Information Security and Cryptography

Colin Boyd
Anish Mathuria
Douglas Stebila

Protocols for Authentication and Key Establishment

Second Edition

Springer

Information Security and Cryptography

More information about this series at http://www.springer.com/series/4752

Colin Boyd • Anish Mathuria • Douglas Stebila

Protocols for Authentication and Key Establishment

Second Edition

Springer

Colin Boyd
Department of Information Security
and Communication Technology
Norwegian University of Science
and Technology
Trondheim, Norway

Anish Mathuria
Dhirubhai Ambani Institute
of Information and Communication
Technology (DA-IICT)
Gandhinagar, Gujarat, India

Douglas Stebila
Department of Combinatorics
and Optimization
University of Waterloo
Waterloo, ON, Canada

Originally published under: Boyd C. and Mathuria A.

ISSN 1619-7100 ISSN 2197-845X (electronic)
Information Security and Cryptography
ISBN 978-3-662-58145-2 ISBN 978-3-662-58146-9 (eBook)
https://doi.org/10.1007/978-3-662-58146-9

To the memory of my father
Thou thy worldly task hast done
– CB

To the memory of my grandmother
– AM

To my parents and teachers
– DS

Preface

The first edition of this book was published in 2003. Inevitably, certain parts of the book became outdated quickly. At the same time new developments have continued apace, including new concrete protocols, new understanding of protocol security properties, and new cryptographic primitives and techniques which can be used in protocol design. Work on a new edition began as early as 2010, but even as it was being written its scope expanded. We are aware that some important protocols have been omitted, and that there are emerging areas which will see considerable activity in the near future (post-quantum secure protocols being one obvious example). However, we still hope that we have been able to capture most of the main developments that have occurred since the first edition.

This second edition has broadly the same purpose and scope as the first edition. We hope to provide a helpful reference for the expert while still being accessible to those newer to the topic, including those trying to obtain a broad overview of the field. In comparison with the first edition, there are three new chapters and all the other chapters (one renamed) have been extensively revised. The new book is around 50% larger with around 225 concrete protocols described and a bibliography with almost twice as many references. Some older material, which we deemed less relevant today, has been removed.

Chapter 1 replaces the first two chapters from the first edition. The chapter is intended to provide the necessary background on cryptography, attack scenarios and protocol goals with an expanded coverage. The initial tutorial introduction has been moved to an appendix, while some parts of Chapter 2 from the first edition were removed as they seemed no longer relevant. An updated, but somewhat shortened, introduction to protocol verification is also included.

Chapter 2 is the first completely new chapter, describing computational models for key exchange and authentication. The purpose of this chapter is not to provide a tutorial on how to read, let alone write, computational proofs, but rather to try to help readers understand what a computational proof provides and how to compare the many different computational models in use. In later chapters we

have freely made reference to the major computational models when discussing specific protocols and their security.

Chapter 3 is an updated chapter covering protocols using shared key cryptography. This includes major updates on the status of the protocols in the ISO 9798-2 and 11770-2 standards.

Chapter 4 is an updated chapter on protocols using public key cryptography. Again, this includes new developments of the ISO standard protocols, this time for the 9798-3 and 11770-3 protocols. Coverage of TLS is moved to the new Chap. 6 which is devoted to TLS.

Chapter 5 on key agreement is, as in the first edition, the longest chapter. There is an amazing diversity of ideas in design of key agreement protocols, even in the simplest case covered in this chapter, which is limited to two-party protocols in the public key setting. Even though several older protocols from the first edition have been omitted, the revised chapter is now more than ten pages longer, illustrating that there are still many new developments occurring.

Chapter 6 is a completely new chapter on the TLS protocol. As the most prominent key establishment protocol in use today, we believe it is more than justifiable to devote a chapter to TLS. The development of the protocol in the past 10 years, culminating in the new TLS 1.3 protocol, provides many lessons for those researching and implementing key establishment protocols.

Chapter 7 is the third new chapter and is dedicated to ID-based protocols. While it gathers in some early ID-based protocols already included in the first edition, the coverage of pairing-based protocols forms the bulk of the chapter and is completely new.

Chapter 8 is an updated chapter on password-based protocols. This is another topic where there has been a great deal of research activity since the first edition leading to a significant expansion in the chapter.

Chapter 9 is an updated (and renamed) chapter on group key establishment. Although group protocols have a long history there has been much recent work to modernise the topic with stronger security properties and formal proofs.

Appendix A covers standards for key establishment and authentication protocols from various standards bodies, updated and expanded from the first edition.

Appendix B consists of a tutorial introduction to protocols for authentication and key establishment. This is unchanged from the corresponding section in the first edition, apart from some small notational revisions.

Acknowledgements

Many people have given their help and advice generously to help us to improve the quality of the book. Different people have reviewed and provided comment on various portions of the book. We thank them all for their generous spirit and are sincerely sorry if we have misinterpreted or forgotten any of their helpful advice. This list includes all those who we can remember; we hope not to have missed anyone.

Craig Costello	Sigurd Eskeland	Håkon Jacobsen
Tibor Jager	Chris Mitchell	Ruxandra Olimid
Kenneth Radke	SeongHan Shin	Berkant Ustaoglu

In addition, Kazuki Yoneyama provided helpful answers to specific questions. We are solely responsible for all errors of fact, presentation style, or just plain typing errors that this book may have.

The team at Springer have been encouraging and helpful throughout. We would particularly like to thank Ronan Nugent for his unfailing confidence and quick action on all matters, even as we delayed time and again. Series editor Kenny Paterson provided enthusiastic and encouraging support.

CB would like to record his personal thanks to his family for bearing with denial of service during the seemingly interminable saga of the second edition. When the work started he was with Queensland University of Technology, particularly the School of Electrical Engineering and Computer Science. More recently he has been at the Norwegian University of Science and Technology in the department now called Information Security and Communication Technology. Both institutions have provided encouragement and resources to support this work.

AM would like to express his gratitude to his wife, Hemal, and daughter, Radhika, for their unfailing love, support, and patience. His work was carried out at the University of Massachusetts at Dartmouth and more recently at the Dhirubhai Ambani Institute of Information and Communication Technology. He would like to thank his colleagues at both of these institutions for their support and encouragement.

DS would like to thank CB and AM for inviting him to join them in updating the fine work they did in the first edition. This work has been carried out over his time at the Queensland University of Technology, McMaster University, and now the University of Waterloo, and colleagues at all of these were supportive of his work on this project.

This book was typeset in LaTeX on various platforms. The style for the protocols and attacks is adapted from Anselm Lingnau's float package.

Trondheim, Gandhinagar, Waterloo
January 2019

Colin Boyd
Anish Mathuria
Douglas Stebila

Contents

List of Protocols

List of Attacks

1

Introduction to Authentication and Key Establishment

1.1 Introduction

Authentication and key establishment are fundamental steps in setting up secure communications. Authentication is concerned with knowing that the correct parties are communicating; key establishment is concerned with obtaining good cryptographic keys to protect the communications, particularly to provide confidentiality and integrity of the data communicated. Because the modern world increasingly relies on digital networks, the security of communications is a critical element in the functioning of society today, and will become only more important in the future.

Protocols for authentication and key establishment (AKE) have acquired a reputation for being difficult to analyse and to design correctly. This may be because of a lack of intuition behind what such protocols are intended to achieve, in contrast to concepts like encryption and data integrity which are more easily compared with familiar real-world situations. Protocols for AKE come in many types according to various criteria. One of the aims of this book is to classify protocols so that they are easier to compare. In this chapter we aim to introduce the main properties that may be used to classify AKE protocols. For the complete newcomer to the subject we include more basic material in tutorial fashion in Appendix B.

We first present in Sect. 1.2 a loose classification of the different *architectural* settings that are commonly encountered in AKE protocols, in terms of the participants, their roles and their initial key material. Section 1.3 outlines cryptographic mechanisms including methods to ensure freshness. Cryptographic primitives are fundamental tools required in almost all practical protocols so we highlight how these are applied in different AKE protocol types. This is followed by a survey of the well-known types of attacks on AKE protocols; understanding protocol failures is essential in understanding how to design and assess protocols. Having understood what may be considered valid attacks, we turn the focus around and define what are the typical *goals* of AKE protocols in Sect. 1.5. Some of these are common to most protocols; others are optional depending on the specific application requirements. Finally, in Sect. 1.6 we discuss some prominent tools for analysis of AKE protocols.

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_1

1.2 Protocol Architectures

Authentication and key establishment typically occur at the start of a *communications session*, which we often call simply a *session*. Authentication allows those parties active in the session to learn the identity of other parties in the session. Key establishment is used to set up a *session key*, used to subsequently protect the data communicated during the session with the help of whatever cryptographic mechanisms are chosen.

There are three features that we regard as architectural criteria to classify different protocols: which keys are already established, how new keys are generated, and how many users a protocol is designed to serve. Note that the second criterion is applicable only to protocols concerned with key establishment in contrast to authentication.

1.2.1 Cryptographic Keys

As a matter of general principle it is not possible to establish an authenticated session key without existing secure channels already being available. In fact this principle can be stated formally and proven to be correct [129]. Therefore, except for the possibility of secure physical establishment of keys, it is essential either that keys are already shared between different principals or that authentic public keys are available. Therefore a key establishment protocol always features two types of keys:

session keys, which are established during the protocol;
long-term keys, which exist before the protocol is run.

Session keys are almost always keys for use with symmetric-key cryptography and are shared between the protocol parties after completion of the protocol. In the protocols we examine throughout this book we almost always assume that there is only one session key defined from a single run of the protocol. In practice, it is common to derive a number of further keys from the session key, for example to obtain different keys for each direction of a bidirectional secure channel. Long-term keys often come in different types; we take particular note of the following options.

Shared keys. Keys for symmetric-key cryptography may be shared by the protocol parties beforehand (and are often called *pre-shared keys*). These may be shared on a pairwise basis between parties, which may include trusted servers as well as ordinary users of the protocol. The protocols in Chap. 3 apply this type of long-term key.

Public–private key pairs. Protocol parties may have long-term public keys for which they hold the corresponding private key. Typically this means that a public key infrastructure (PKI) must be in place so that parties can validate public keys via certificates. It is common to omit the details of certificate communication and verification in AKE protocol descriptions, although there have been some attempts to analyse protocols together with PKI concerns [136]. The protocols in Chaps. 4, 5 and 6 apply this type of long-term key.

Identity-based keys. An alternative to normal public key cryptography is identity-based cryptography in which public keys can be replaced by identity strings and shared public parameters. A key generation server is required in order to distribute corresponding private keys to protocol parties. The protocols in Chap. 7 apply this type of long-term key.

Passwords. Strictly speaking we can regard passwords as a special type of shared key, often shared between a user and a server. There are, however, cases where the server does not keep the plaintext password, but only a one-way image of it. In any case, password-based AKE protocols require special treatment since they cannot resist all attacks that protocols using high-entropy full-length keys can. The protocols in Chap. 8 apply this type of long-term key.

1.2.2 Method of Session Key Generation

There are various ways to generate session keys in an AKE protocol. In the following, we use the term *user* to mean an entity who will use the session key for subsequent communication. We also use the term *principal* or *party* to mean an entity who will engage in the protocol. For example, in a protocol which uses a key server (often called an *authentication server*) there are users who will obtain the session key while the server is a principal but not a user.

Definition 1. *A* key transport *protocol is an AKE protocol in which one of the principals generates the session key and this key is then transferred to all protocol users in that session.*

Definition 2. *A* key agreement *protocol is an AKE protocol in which the session key is a function of inputs from all protocol users in that session.*

Definition 3. *A* hybrid *protocol is an AKE protocol in which the session key is a function of inputs from more than one principal in the session, but not by all users. This means that the protocol is a key agreement protocol from the viewpoint of some users, and a key transport protocol from the viewpoint of others.*

The type of protocols described in Definition 3 are not common in the literature but are easily instantiated. An example is given in Protocol 1.1 below. Protocols using an online key server often use key transport, whereas protocols where users have public keys (often certified by an offline server) often use key agreement. However, this is not always the case, and there are examples of key agreement protocols in which an online key server provides an input to the session key, and key transport protocols using public long-term keys.

We focus on key transport protocols in Chaps. 3 and 4 although they do occur in other chapters too. We focus on key agreement protocols in Chap. 5 but they are also prominent in Chaps. 6, 7, 8 and 9.

1.2.3 Number of Parties

A final component of the protocol architecture is the number of parties, or principals, that are intended to take part in a session of the protocol. The majority of AKE protocols have concentrated on the case where two users wish to establish a session key for point-to-point communications and this case is the focus of most of the chapters in the book. Extending to the case of *group* key establishment, where more than two users wish to establish a joint session key, can complicate matters a great deal. Group protocols are examined in Chap. 9. In most cases group key establishment protocols apply key agreement, but there are also key transport examples; it is quite possible for a group protocol to look like a key transport protocol to some principals (who receive the session key on a cryptographic channel) and a key agreement protocol to other principals (who have an input to the session key).

1.2.4 Example

The three criteria mentioned above may be used to classify key establishment protocols in different ways. We have used all three as criteria for splitting the material in this book into chapters. Nevertheless, it is not always easy to decide where a protocol should lie. We give in Protocol 1.1 an example with unusual properties. This protocol uses an online server, applies hybrid key generation and has two users. As far as we are aware, this simple protocol does not correspond to any protocol published elsewhere.

At the start A and B share long-term keys, K_{AS} and K_{BS} respectively, with S. The session key is calculated as $K_{AB} = f(N_B, N_S)$ for a suitable function f, where N_B and N_S are random values generated by B and S respectively. We use ID_A to denote the identity of party A and ID_B to denote the identity of party B. The notation $\{\ldots\}_K$ indicates encryption with a shared key K.

Goal: Hybrid key establishment of shared key $K_{AB} = f(N_B, N_S)$

1. $A \rightarrow B: ID_A, N_A$
2. $B \rightarrow S: \{N_B, ID_A, ID_B\}_{K_{BS}}, N_A$
3. $S \rightarrow A: \{K_{AB}, ID_A, ID_B, N_A\}_{K_{AS}}, N_S$
4. $A \rightarrow B: N_S, \{ID_A, ID_B\}_{K_{AB}}$
5. $B \rightarrow A: \{ID_B, ID_A\}_{K_{AB}}$

Protocol 1.1: A protocol in an unusual class

Upon receiving message 2, S must check that the value obtained by decrypting the field ID_B is the same as the identity of the principal whose key is used to decrypt the message. On receipt of message 4, B uses N_B to compute $K_{AB} = f(N_B, N_S)$. From B's viewpoint Protocol 1.1 is like a key agreement protocol because B has input to

the key. From A's viewpoint it looks like a key transport protocol. This is why we call it a hybrid protocol according to Definition 3.

1.3 Cryptographic Tools

In this section we highlight the importance of distinguishing the possible different properties that may be provided by cryptographic algorithms. Much of the material in this section is summarised from the classic book of Menezes, van Oorschot and Vanstone [550]. More modern treatments can be found in other textbooks [412, 678].

A good understanding of the algorithms and methods of cryptography is highly beneficial in assessing cryptographic protocols, but it is not the purpose of this section to develop such an understanding, only to give a brief overview. We can identify four fundamental objectives that may be achieved by cryptographic algorithms.

Confidentiality ensures that data is available only to those authorised to obtain it. This is usually achieved through encryption of the data so that only those with the correct decryption key can recover it. In AKE protocols, it is essential that the long-term keys (and possibly some internal values) remain confidential, while the goal is to establish a session key that is itself confidential.

Data integrity ensures that data has not been altered by unauthorised entities. This can be achieved through use of hash functions in combination with encryption, or by use of a message authentication code to create a separate check field. Data integrity is required in many AKE protocols to protect elements such as identity fields and nonces.

Data origin authentication guarantees the origin of data. It is a fundamental step in achieving entity authentication in protocols as well as in establishing keys. Since altering the data must alter its origin, we may say that data origin authentication implies data integrity. Although it is in principle possible to achieve data integrity without origin authentication, they are normally achieved by the same cryptographic mechanisms.

Non-repudiation ensures that entities cannot deny sending data that they have committed to. This is typically provided using a digital signature mechanism. Non-repudiation is rarely a requirement in protocols for authentication and key establishment, but it automatically provides the important data integrity and data origin authentication services.

Some cryptographic transformations can provide more than one of these properties. It is often noted that if the message source has sufficient redundancy, for example a natural language source like English, then encryption for confidentiality automatically provides some degree of data integrity. However, it is a serious error to believe that just because a decrypted message makes sense it must be the same message that was sent. Before looking at the properties of cryptographic algorithms meeting the different objectives, let us consider an example illustrating the importance of identifying which properties are provided.

The *one-time pad* is one of the simplest cryptosystems and it is also provably secure in a very strong sense. If we assume that the message is a string of n bits $b_1, b_2, \ldots, b_n$ then the key is a random string of bits $k_1, k_2, \ldots, k_n$. Encryption takes place one bit at a time and the ciphertext string $c_1, c_2, \ldots, c_n$ is found by adding, modulo 2, each message bit to the corresponding key bit:

$$c_i = b_i \oplus k_i \text{ for } 1 \leq i \leq n.$$

Decryption is the same process as encryption, since adding modulo 2 is a self-inverse operation. Now, suppose that an adversary knows that the one-time pad is used to secure the amount sent in a funds transfer. The adversary may alter any of the bits of the ciphertext and change the amount sent even without finding what the original amount was. If the amounts are usually small then the adversary could alter the most significant bit and expect to increase substantially the amount sent. This simple example shows that the one-time pad on its own provides no data integrity even though it provides perfect secrecy.

We now consider definitions for the cryptographic mechanisms that are typically used to provide the main cryptographic services. Table 1.1 summarises the notation which we will use for these mechanisms throughout this book.

Table 1.1: Summary of notation for cryptographic algorithms

$\mathsf{Enc}_A(M)$	Public key encryption of message M with public key of party A.
$\{M\}_K$	Symmetric encryption of message M with shared key K.
$\mathsf{Encap}_A(\cdot)$	Public key encapsulation of a shared secret with public key of party A.
$\mathsf{MAC}_K(M)$	Message authentication code of M using shared key K.
$\mathsf{Sig}_A(M)$	Digital signature of message M generated by party A.

The definitions we will give are informal, but rely on an intuitive understanding of what it means for a computation to be easy or difficult. In complexity theory a computation is said to be easy (or feasible) if the time it takes to complete increases as a polynomial (which could be a constant) of the input length. If the time required increases faster than any polynomial then we may say that the computation is hard (or infeasible). Sometimes it is more meaningful to have a specific number of operations in mind. Nowadays it is accepted that performing up to around 2^{60} fundamental computational operations is 'easy' while performing 2^{160} such operations is 'hard', with something of a grey area in between.

1.3.1 Confidentiality

Definition 4. *An* encryption scheme *defines four sets: a set of encryption keys K_E, a set of decryption keys K_D, a message set M, and a ciphertext set C, together with three algorithms.*

1. *A key generation algorithm, which outputs a valid encryption key $k \in K_E$ and a valid decryption key $k^{-1} \in K_D$.*
2. *An encryption algorithm, which takes an element $m \in M$ and an encryption key $k \in K$ and outputs an element $c \in C$. The encryption algorithm may be randomised so that a different c will result given the same m.*
3. *A decryption function, which takes an element $c \in C$ and a decryption key $k^{-1} \in K$ and outputs an element $m \in M$ (or possibly a special error symbol). We require that if c is a valid encryption of m, then the decryption of c yields m.*

An encryption scheme is a *symmetric key* algorithm if $K_E = K_D$ and $k = k^{-1}$. In contrast, in an *asymmetric* or *public key* encryption algorithm k and k^{-1} are different and it is computationally hard to obtain the private key k^{-1} from the public key k. As shown in Table 1.1 we use different notation to distinguish between symmetric and asymmetric encryption.

Although all encryption algorithms are intended to hide information from the adversary, different algorithms can provide different properties. The following definition tries to capture the idea that the ciphertext should not be any help to the adversary in learning anything new, including the plaintext.

Definition 5. *An encryption scheme provides* semantic security *if anything that can be efficiently computed given the ciphertext can also be efficiently computed without the ciphertext.*

A useful equivalent characterisation of semantic security is *indistinguishability*. This means that, given the ciphertext corresponding to one out of two chosen messages, the adversary cannot guess with probability greater than 1/2 which message is actually the plaintext. In this challenge the adversary is able to obtain the encryption of any chosen messages; in other words the adversary is allowed a *chosen plaintext* attack.

Definition 6. *An encryption scheme provides* non-malleability *if it is infeasible to take a ciphertext of one message and transform it into the ciphertext of a different related message, without knowledge of the original message.*

The property of non-malleability is strictly stronger than semantic security. Indeed the definition is known to be equivalent to the indistinguishability property mentioned above when the adversary is additionally given the decryption of any chosen ciphertexts; an algorithm with non-malleability is secure against a *chosen ciphertext* attack.

Since non-malleability is a stronger property than semantic security it is not surprising that algorithms providing non-malleability in general have greater computational requirements and message expansion than those providing only semantic security. The extent of the overhead varies from algorithm to algorithm, but where computation and bandwidth are at a premium it is important to know whether an algorithm used in a protocol must provide non-malleability. There are many AKE protocols in which non-malleability is required, although some protocol designers have only implicitly recognised this.

In addition to defining what an adversarial goal is, we often specify elements that an adversary may have access to, in addition to the ciphertext of interest. We have already mentioned the two following types of attack which give such help to the adversary.

Definition 7. *In a* chosen plaintext attack *(CPA) the adversary can obtain ciphertexts of any chosen messages. In a* chosen ciphertext attack *(CCA) the adversary can obtain the decrypted plaintext of any chosen ciphertexts, except for the one under attack.*

The abbreviation IND-CPA is often for an encryption algorithm which provides indistinguishability against chosen plaintext attacks. Similarly, the abbreviation IND-CCA is often used to describe an encryption algorithm which provides indistinguishability against chosen ciphertext attacks.

CCA attacks are sometimes divided into CCA1, where the adversary has access to the decryption function only until it fixes its target ciphertext for attack, and CCA2 where the adversary always has access to the decryption function but cannot use it on the target ciphertext.

An encryption scheme is designed to provide confidentiality to any message, even a single bit. Such a general mechanism is not needed when the goal is to so transmit a random value confidentially. An alternative mechanism is a *key encapsulation mechanism* (KEM), which is designed to generate a new random-looking value, together with an encapsulated version of that value which can only be recovered by the chosen recipient. We can regard a KEM as a kind of encryption scheme which can only be used to encrypt new random values. Since KEMs are often more efficient than general encryption schemes, it is not surprising that they have been used in preference to encryption in some AKE protocol designs. We will see examples of key agreement protocols designed using KEMs in Sect. 5.8.

Definition 8. *A* key encapsulation mechanism *consists of four sets: a public key set K_E, a private key set K_D, a randomness set R, and a ciphertext set C together with three algorithms.*

1. *A key generation algorithm, which outputs a valid public key $K \in K_E$ and a valid private key $K^{-1} \in K_D$.*
2. *An encapsulation algorithm, which takes a public encapsulation key $k \in K$ and outputs a new symmetric key k and an element $c \in C$. The encapsulation algorithm is usually randomised so that a different (c,k) pair is output each time it is called. We normally write $(c,k) = \mathsf{Encap}_K(\cdot)$, ignoring the randomness.*
3. *A decapsulation function, which takes an element $c \in C$ and a private key $K^{-1} \in K_D$ and outputs a symmetric key k. We require that if (c,k) is output by $\mathsf{Encap}_K(\cdot)$, then k is output by $\mathsf{Decap}_{K^{-1}}(c)$.*

1.3.2 Data Origin Authentication and Data Integrity

Data authentication and integrity are essential in most protocols for authentication and key establishment. These two cryptographic services are strongly connected and

are typically both provided together using the same mechanism. This is because the origin of a message can only be guaranteed if the message has not changed since it was formed.

The most common mechanism for providing data origin authentication and data integrity is to append a tag to a message constructed using a message authentication code (MAC). The message may be transmitted either in plaintext or encrypted. On receipt of the MAC tag, a recipient with the correct key is able to recompute the tag from the message and verify that it is the same as the tag received.

Definition 9. *A* message authentication code *(MAC) is a family of functions parametrised by a key k such that* $\mathsf{MAC}_k(m)$ *takes a message m of arbitrary length and outputs a fixed-length value, and satisfies the following properties.*

1. *It is computationally easy to calculate* $\mathsf{MAC}_k(m)$ *given k and m.*
2. *Given MAC values for any number of messages under the given key k (even messages chosen adaptively by an adversary), it is computationally hard to find any valid MAC value for any new message.*

1.3.3 Authenticated Encryption

It is very natural that data may have to be secured in terms of both confidentiality and data integrity at the same time. For example, data sent over a secure channel is routinely protected in both these ways. An *authenticated encryption* algorithm provides both properties together.

Although it is quite possible to build authenticated encryption by combining separate algorithms, such as encryption and MACs, there are potential benefits of using an integrated algorithm; such benefits may be efficiency and less chance to combine the algorithms in a bad way. In Chapter 6 we describe attacks which are possible due to an unfortunate mix of MAC and encryption. Today there are standardised algorithms for authenticated encryption, such as the GCM mode of operation for block ciphers [575].

1.3.4 Non-repudiation

Non-repudiation is usually provided through a digital signature mechanism. Although non-repudiation is not a property that is typically required for authentication or key establishment protocols, nevertheless digital signatures are a common element in their construction. This is because digital signatures also provide authentication and data integrity services; their implementation through public keys makes them useful for providing these essential services.

Definition 10. *A* digital signature algorithm *consists of four sets: a set of signing keys* K_S*, a set of verification keys* K_V*, a message set M, and a signature set S, together with three algorithms.*

1. *A key generation algorithm, which outputs a valid signature key* $k \in K_S$ *and a valid verification key* $k^{-1} \in K_V$.

2. *A signature generation algorithm, which takes an element $m \in M$ and a signature key $k \in K_S$ and outputs an element $s \in S$. We will write $s = \mathsf{Sig}_A(m)$ where K is the signature generation key of party A. The signature generation algorithm may be randomised so that a different output will result given the same m.*
3. *A verification function, which takes a signature $s \in S$, a message $m \in M$, and a verification key $k^{-1} \in K_V$ and outputs an element $v \in \{0,1\}$. If $v = 1$ then we say the signature is* valid *or if $v = 0$ we say that the signature is* invalid.

A digital signature algorithm is regarded as secure if it is computationally hard for the adversary to find a valid signature of any message that has not been previously signed, even given many previously signed messages (chosen adaptively). A signature with this property is often said to be *unforgeable*.

In the above definition we have stated that it is necessary to possess both the signature s and the message m in order to verify the signature. This is sometimes called a signature *with appendix*. In contrast, a signature that can be verified without separate knowledge of the message is called a signature with *message recovery*. Throughout the book we will use the notation $\mathsf{Sig}_A(m)$ to denote a signature with appendix of message m from party A. Notice that even though we assume that m must be available in order to verify $\mathsf{Sig}_A(m)$, it does not follow that an adversary cannot obtain information about m from $\mathsf{Sig}_A(m)$.

1.3.5 Examples of Cryptographic Algorithms

Table 1.2 lists some of the best known cryptographic algorithms which provide the different types of cryptographic properties discussed earlier in this section. Some of these algorithms have been proven secure in the sense that there is a proven reduction to some well-known difficult problem. However, the reader should be aware that these reductions have varying complexities and the underlying problems have no proof regarding their absolute difficulty. The IEEE P1363 standard [372] covers a few public key algorithms and how to implement them. Several books [412, 520, 678] provide detailed descriptions of such algorithms and explain their security proofs.

The Advanced Encryption Standard (AES) is given as an example of a block cipher. The security properties of any block cipher depend critically on the *mode of operation* of the cipher, and so we have just given the generic term 'confidentiality' as its provided security service. One particular mode is GCM (Galois counter mode) which is included in the table as an example of an algorithm for authenticated encryption.

Modern complexity-theoretic definitions are nowadays in frequent use in the literature of cryptography. The definitions given above are informal versions of these. Study of the formal definitions is helpful in gaining a deeper understanding of what algorithms are appropriate to use in a particular protocol, but is outside the scope of this book. Relationships exist between the different formal definitions of confidentiality [73]. Similarly formal definitions of security for digital signatures are available. Certain algorithms have been proven to possess the different cryptographic properties defined above, given certain reasonable assumptions on the underlying mathematical problems, and sometimes about the existence of functions with

Table 1.2: Some well-known cryptographic algorithms and their properties

Algorithm	Type	Cryptographic service
AES [237, 574]	Block cipher	Confidentiality
ElGamal encryption [267]	Public key cipher	Semantic security
Cramer–Shoup [224]	Public key cipher	Non-malleability
RSA-OAEP [77]	Public key cipher	Non-malleability
RSA signature [372, 630]	Digital signature	Non-repudiation
DSS [577]	Digital signature	Non-repudiation
SHA-2 [579]	Hash function	One-way function
HMAC [71]	MAC	Data integrity
GCM [575]	Block cipher mode	Confidentiality and integrity

random-like properties. In many protocols the cryptographic algorithms used are not specified. However, it is helpful to know that appropriate algorithms do actually exist.

1.3.6 Secret Sharing

Secret sharing is a mechanism that allows the owner of a secret to distribute *shares* amongst a group. The owner of the secret is often called the *dealer*. Individual shares are of no help in recovering the secret, but if all shares in some predefined *access sets* are available then the secret can be collectively found. A (t,n) *threshold scheme* is a secret sharing scheme for which n shares are distributed, such that any set of t (or more) shares is sufficient to obtain the secret, while any set of $t-1$ (or fewer) shares is of no help in recovering the secret.

There are some similarities between secret sharing and key establishment for groups since, for both, a group of users cooperates to derive a secret value. However, a secret sharing scheme on its own lacks the means to provide fresh keys, to distribute keys to principals, and to provide key authentication. We look at some specific schemes based on secret sharing in Chap. 9.

The most well-known threshold scheme is due to Shamir [664] and is based on the use of polynomial interpolation. This allows any polynomial of degree d to be completely recovered once any $d+1$ points on it are known. Polynomial interpolation works over any field, but in cryptographic applications the field is typically $\mathbb{Z}_p$, the field of integers modulo p, for some prime p.

In order to share a secret $s \in \mathbb{Z}_p$ in Shamir's (t,n) threshold scheme, the dealer generates a polynomial of degree $t-1$,

$$f(z) = a_0 + a_1 z + \ldots + a_{t-1} z^{t-1},$$

with coefficients randomly chosen in $\mathbb{Z}_p$ except for $a_0 = s$. The shares are values $f(x)$ with $1 \leq x \leq n$. If any t shares are known then s can be recovered. For example, if $f(1), f(2), \ldots, f(t)$ are known then:

$$s = \sum_{i=1}^{t} f(i) \prod_{1 \leq j \leq t, j \neq i} \frac{j}{j-i}.$$

Given any $t-1$ points on the polynomial (excluding the value at 0), all possible values for $f(0)$ can be obtained given one extra point. Consequently, absolutely no information about the secret can be obtained if $t-1$ or fewer shares are known.

1.3.7 Freshness Mechanisms

One of the basic requirements for protocols used to establish a session key is that each user of the key should be able to verify that it is new and not replayed from an old session. This property extends to a variety of other protocol types. For example, protocols designed to achieve authentication in real time also need to ensure that messages sent are not replays. Thus we see that the need to ensure that message elements are new, or *fresh*, is a very common protocol requirement. It is thus worthwhile to look at the typical ways that freshness is achieved in existing protocols.

We can consider two different ways that freshness of a value may be guaranteed to a particular user. The first is that the user has a part in choosing the value (which will often be a new session key), while the second is that the user has to rely on something received with the value that is known to be fresh itself. A typical instance of the first case is in a key agreement protocol. Here two users A and B both choose an input, N_A and N_B respectively, to a new session key K_{AB}. The session key is formed by choosing some function of the inputs:

$$K_{AB} = f(N_A, N_B).$$

A desirable property of the function f is that it should not be possible for A or B to force an old value of K_{AB} even if the other's input is known. This means that each user has independent assurance that K_{AB} is fresh. This property is achieved for A if, once N_A is chosen, B is unable to choose N_B in such a way that K_{AB} is an old value. If we define the function $g(.) = f(N_A, .)$ then this means that g must be a one-way function. In practice it may be necessary to add other conditions to disallow exceptional values. A symmetrical condition must also hold to provide the same property for B. One very common example of such a function is the basic Diffie–Hellman protocol which we examine in detail in Chap. 5.

Let us turn now to the second case, where freshness depends on something received with the message. Suppose that a principal A wishes to verify the freshness of a session key K_{AB} that has been generated by some principal S (which may be a server or perhaps the principal B that shares K_{AB}). Principal A must trust S to freshly generate K_{AB} but needs to be sure that the message received is not an old message that has been replayed by the adversary. Assume that A receives the message field

$F(K_{AB},N)$ which is a function of K_{AB} and a freshness value N. What are the properties required of F to allow the recipient to be sure that the composite message is fresh? F must provide data origin authentication and data integrity so that the recipient can deduce that $F(K_{AB},N)$ was generated by S and has not been altered. If A can be sure that N is fresh then she can also be sure that $F(K_{AB},N)$ is fresh. Since S is trusted to generate and authenticate only fresh keys, A can therefore deduce that K_{AB} is fresh. We next consider the different forms that the freshness value N can take.

A freshness value must have the property that it can be guaranteed not to have been used before. (In some protocols, values used for freshness are also required to have other properties, but this is because they are used for other purposes as well as to ensure freshness.) There are three common types of freshness value used: timestamps, nonces and counters. Gong [317] classified the various ways that these types may be used in a protocol.

Timestamps. The sender of the message adds the current time to the message when it is sent. This is checked by the recipient when the message is received by comparing with the local time. If the received timestamp is within an acceptable window of the current time then the message is regarded as fresh. The difficulty of using timestamps is that synchronised time clocks are required and must be maintained securely. Gong [314] pointed out that if a principal's clock is *advanced* beyond the time in the rest of the system, a vulnerability can exist even after the clock has been corrected. This is because an adversary could have captured, and suppressed, a message that will become fresh in the future. Gong calls this a *suppress relay* attack.

Nonces (random challenges). The recipient A of the message generates a nonce ('number used only once') N_A, and passes it to the sender of the message B. The nonce N_A is then returned with the message after processing with some cryptographic function f as shown in Protocol 1.2. A checks the nonce on receipt and deduces that the message is fresh because the message cannot have been formed before the nonce was generated. A disadvantage of using a challenge is that it requires an interactive protocol which may add to both the number of messages and the number of message exchanges required. Attention must also be paid to the quality of random numbers produced, since if the nonce to be used is predictable a valid reply can be obtained in advance and later replayed (a preplay attack).

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : f(N_A, \ldots)$

Protocol 1.2: Use of a nonce (random challenge)

Counters. The sender and recipient maintain a synchronised counter whose value is sent with the message and then incremented. A disadvantage of counters is that

state information must be maintained for each potential communication partner. Management of counters can also cause problems in the presence of channel errors. Gong [317] pointed out that if a counter is not synchronised with the receiver then preplay attacks are possible.

Mitchell [556] suggested a hybrid between counters and timestamps. His idea is to use a counter based on real time. For example, the counter may be the time in seconds from some starting point: this allows a 32-bit counter to be used for over 136 years before repeating. The motivation for such a suggestion is that a server authenticating multiple clients can easily recover from loss of the counter states by checking its real-time clock. Mitchell discusses an application in mobile telephony in which the mobile handset may not have a reliable clock but is still required to verify freshness of session keys generated by a server.

A different way of obtaining freshness, which is less common, is if an element is transformed with a cryptographic key that is known to be fresh. For this method to work the cryptographic transformation must provide data integrity. Then the recipient can be sure that the fresh key has been used to form the message, and so the message must have been formed after the key was formed.

1.4 Adversary Capabilities

The purpose of this section is to summarise common ways that an adversary may attack a protocol. We can consider these as techniques or strategies that an adversary may use. Before looking at these we sound a couple of notes of caution.

Firstly, the list will not be complete. The ways in which the adversary may interact with one or more protocol runs are infinite. There are almost certainly attack possibilities that we have omitted. Indeed, it could be argued that it is not tremendously helpful to know that a protocol is not vulnerable to a certain list of threats; what is really required is confidence that it meets its security objectives given a known list of assumptions. We consider ways in which we may be able to achieve such guarantees in Sect. 1.6 and Chap. 2. On the other hand, we should not underestimate the usefulness of a list of typical weaknesses to check against.

Secondly, different protocols have different objectives. Some protocols are concerned with key establishment, others solely with entity authentication. There may be additional goals, such as confirmation that the session key was correctly received by the protocol users. We consider different goals for AKE protocols in Sect. 1.5. Naturally, whether or not a protocol achieves particular goals depends on what attacks are deemed possible.

Bearing in mind these caveats, we now consider the most commonly encountered threats to cryptographic protocols. Table 1.3 lists and defines the attacks and they are considered in turn in more detail below. There are certainly other ways to classify attacks; an alternative list, with examples, was given by Carlsen [182].

Table 1.3: Types of protocol attack

Eavesdropping	The adversary captures the information sent in the protocol.
Modification	The adversary alters the information sent in the protocol.
Replay	The adversary records information seen in the protocol and then sends it to the same or a different principal, possibly during a later protocol run.
Preplay	The adversary engages in a run of the protocol prior to a run by the legitimate principals.
Reflection	The adversary sends protocol messages back to the principal who sent them.
Denial of service	The adversary prevents or hinders legitimate principals from completing the protocol.
Typing attacks	The adversary replaces a (possibly encrypted) protocol message field of one type with a (possibly encrypted) message field of another type.
Cryptanalysis	The adversary gains some useful leverage from the protocol to help in cryptanalysis.
Certificate manipulation	The adversary chooses or modifies certificate information to attack one or more protocol runs.
Protocol interaction	The adversary chooses a new protocol to interact with a known protocol.

1.4.1 Eavesdropping

Eavesdropping is perhaps the most basic attack on a protocol. Nearly all protocols address eavesdropping by using encryption. It is obvious that encryption must be used to protect confidential information such as session keys. In certain protocols there may be other information that also needs to be protected. An interesting example is that protocols for key establishment in mobile communications usually demand that the identity of the mobile station remain confidential. Eavesdropping is sometimes distinguished as being a *passive* attack since it does not require the adversary to disturb the communications of legitimate principals. The other attacks we consider all require the adversary to be *active*. It should be remembered that many sophisticated attacks include eavesdropping of protocol runs as an essential part.

1.4.2 Modification

If any protocol message field is not redundant then modification of it is a potential attack. Use of cryptographic integrity mechanisms is therefore pervasive in protocols for authentication and key establishment.

Whole messages, as well as individual message fields, are vulnerable to modification. Many attacks do not alter any known message field at all, but split and

reassemble fields from different messages. This means the integrity measures must cover all parts of the message that must be kept together; encryption of these fields is not enough. Examples of attacks on protocols in which encryption does not provide the required integrity properties were given by Stubblebine and Gligor [699] and by Mao and Boyd [521].

1.4.3 Replay

Replay attacks include any situation where the adversary interferes with a protocol run by insertion of a message, or part of a message, that has been sent previously in any protocol run. Replay is another fundamental type of attack which is often used in combination with other attack elements. Just as almost all protocols address eavesdropping and modification attacks by using cryptography, almost all protocols include elements to address possible replay attacks. Various means to combat replay were discussed in Sect. 1.3.7.

It is possible for the replayed message in an attack to have been originally part of a protocol run that happened in the past. Alternatively the replayed material may be from a protocol run that takes place at the same time as the attacking run. Syverson [703] produced a taxonomy of replay attacks based upon this distinction.

1.4.4 Preplay

Preplay might be regarded as a natural extension of replay, although it is not clear that this is really an attack that can be useful on its own. The distinction is that, in a preplay attack, the adversary is *active* in the earlier protocol run, with the aim of setting up the correct conditions for an attack on the later run. An interesting example of an attack that employs preplay is the so-called *triangle attack* of Burmester [167] which will be presented in Sect. 5.3.5.

1.4.5 Reflection

Reflection is really an important special case of replay. In a typical scenario a principal engages in a shared key protocol and the adversary simply returns a challenge to the originating party. This attack may only be possible if parallel runs of the same protocol are allowed but this is often a realistic assumption. For example, if one principal is an Internet host, it may accept sessions from multiple principals while using the same identity and set of cryptographic keys. The possibility of instigating several protocol runs simultaneously is another common and realistic strategy for the adversary.

Consider Protocol 1.3, which gives a very basic example. Suppose A and B already share a secret key K and choose respective nonces N_A and N_B for use in the protocol. The protocol is intended to mutually authenticate both parties by demonstrating knowledge of K.

On receipt of message 2, A deduces that it must have been sent by B since only B has K. However, if A is willing to engage in parallel protocol runs then there is

1. $A \to B : \{N_A\}_K$
2. $B \to A : \{N_B\}_K, N_A$
3. $A \to B : N_B$

Protocol 1.3: A protocol vulnerable to reflection attack

another possibility, namely that message 2 was originally formed by A. An adversary C can successfully complete two runs of the protocol, as shown in Attack 1.1.

1. $A \to C : \{N_A\}_K$
1′. $C \to A : \{N_A\}_K$
2′. $A \to C : \{N'_A\}_K, N_A$
2. $C \to A : \{N'_A\}_K, N_A$
3. $A \to C : N'_A$
3′. $C \to A : N'_A$

Attack 1.1: Reflection attack on Protocol 1.3

Immediately after receiving the first message, C starts another run of the protocol with A, and reflects back the message received from A. The reply allows C to respond to the first message and then both runs of the protocol can be completed. In fact all cryptographic processing has been performed by A, while A believes two protocol runs have been completed with B.

These sorts of attacks are sometimes called *oracle attacks* since A acts as an oracle to C by presenting the required decryption. An extensive treatment of reflection attacks was given by Bird *et al.* [103].

1.4.6 Denial of Service

In a denial of service attack (often contracted to *DoS attack*) the adversary prevents legitimate users from completing the protocol. Denial of service attacks in practice take place against servers which are required to interact with many clients. Attacks can be divided into those that aim to use up the computational resources of the server (*resource depletion attacks*) and those that aim to exhaust the number of allowed connections to the server (*connection depletion attacks*).

As a matter of principle it seems that it is impossible to prevent denial of service attacks completely. Any attempt to establish a connection must either result in allocation of a connection or use some computational work to establish that the attempt is invalid. Nevertheless there are certain measures that may be taken to reduce the impact of denial of service attacks and some protocols are much more vulnerable to this sort of attack than others, so it is important not to ignore this issue.

- Aura and Nikander [45] suggested using *stateless connections* to protect against connection depletion attacks. Their idea is to make the client store all the state information required by the server and return it to the server as necessary with each message sent. In this way the server need not store any state information. Of course it is necessary that the state information returned to the server can be verified by the server to be authentic and it may also need to be confidential. Therefore there is an overhead in both communication and computation incurred by transforming protocols to provide stateless connections.
  A practical mechanism for delaying the need for state at the server is the use of *cookies*, which first seems to have been suggested by Karn and Simpson for their Photuris protocols (most recent version 1999 [411] but originally published in 1995). When a client attempts to make a connection the server sends back a cookie. This procedure is similar to the familiar use of cookies by web servers, but here the cookies take a special form: they are a function of a secret known only to the server and other information unique to the particular connection. At this stage the server stores no state for this request. The client needs to return the cookie in the next message and its validity can be checked by the server from the information sent and its secret. The idea is to ensure, before investing significant resources, that the client is making a unique request for connection. This technique prevents denial of service attacks in which the adversary sends random connection requests. By changing the server secret on a regular basis (perhaps every 60 seconds) even the same client can be prevented from making unlimited connection requests.
- Meadows [537] suggested that in order to protect against connection depletion each message in a protocol must be authenticated. However, to minimise possibly wasted computation the authentication can be weak at the start of the protocol and increase in strength with subsequent messages. Cookies formed by the server, and which must be returned by the client, may form the weak authentication. Meadows developed a formal framework based on the idea of *fail–stop protocols* introduced by Gong and Syverson [320] which abort as soon as a bogus message is discovered.
- Juels and Brainard [404] proposed a mechanism that they called *client puzzles* to form stronger authentication than that provided by cookies. Their idea is that when the load on a server becomes high (possibly as a result of a denial of service attack) the server will send a 'puzzle' of moderate computational difficulty to each client which must be solved before a new connection is made. Genuine clients will be only mildly inconvenienced by this demand but an adversary trying to make multiple connections will have to solve many puzzles. Formal models exist to measure the effectiveness of client puzzles for denial of service resistance in AKE protocols [334, 686].

1.4.7 Typing Attacks

When a protocol is written on the page its elements are clearly distinct. But in practice a principal receiving a message, whether encrypted or not, simply sees a string

of bits which have to be interpreted. Typing attacks exploit this by making a recipient misinterpret a message, accepting one protocol element as another one (that is, a message element of a different type). For example, an element which was intended as a principal identifier could be accepted as a key. Such an attack typically works with replay of a previous message.

An example can be seen in the well-known protocol of Otway and Rees [597] shown in Protocol 1.4 (see also Sect. 3.4.2). Principals A and B, with identities ID_A and ID_B, share long-term keys, K_{AS} and K_{BS} respectively, with the server S. S generates a new session key K_{AB} and passes it to both A and B. M and N_A are nonces chosen by A and N_B is a nonce chosen by B.

Goal: Key transport of K_{AB} from S to A and B

1. $A \rightarrow B : M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}$
2. $B \rightarrow S : M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}, \{N_B, M, ID_A, ID_B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Protocol 1.4: Otway–Rees protocol

The typing attack works because of the similarity in the encrypted parts of the first and last messages – they start with the same message field and are encrypted with the same key. As usual for this kind of attack we need to make some extra assumptions if the attack is to succeed. The attack depends on the length of the composite field M, ID_A, ID_B being the same as that expected for the key K_{AB}. This may be a quite reasonable assumption; for example, M may be 64 bits, and ID_A and ID_B could be 32 bits, so that K_{AB} would have to be of length 128 bits, which is a popular choice of symmetric key size. With these assumptions, an adversary C is able to execute Attack 1.2. Here we introduce the notation C_B to indicate that the adversary C is masquerading as principal B.

1. $A \rightarrow C_B : M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}$
4. $C_B \rightarrow A : \{N_A, M, ID_A, ID_B\}_{K_{AS}}$

Attack 1.2: Typing attack on Otway–Rees protocol

C masquerades as B and intercepts the message from A. C then returns the encrypted part of this message to A, which is interpreted by A as message 4 of the protocol. With the assumptions mentioned above, A will accept the composite field M, ID_A, ID_B as the shared key K_{AB}. Of course C knows the values of M, ID_A and ID_B from message 1, and so is able to continue masquerading as B for the duration of the session.

Typing attacks can be countered by various measures. Ad hoc precautions include changing the order of message elements each time they are used, and ensuring that each encryption key is used only once. More systematic methods are to include an authenticated message number in each message or an authenticated type field with each field. Naturally these come at a cost in computation and bandwidth. Aura [44] has considered systematic methods to avoid typing-based replay attacks.

Chen and Mitchell [197] defined *parsing ambiguity attacks*, a related notion to typing attacks in the sense that both attack types depend on principals misinterpreting message fields. However, rather than re-using protocol messages in 'the wrong place', parsing ambiguity attacks simply expect two or more concatenated fields to be parsed wrongly. Chen and Mitchell illustrated such attacks with several examples from international standard protocols. As with typing attacks, these attacks can be prevented by using appropriate coding methods to avoid any possibility of misinterpreting message types and their encoding.

1.4.8 Cryptanalysis

Cryptographic algorithms used in AKE protocols are often treated abstractly and considered immune to cryptanalysis. However, there are some exceptions that should be mentioned. The most important exception is when it is known that a key is weak and is (relatively) easy to guess once sufficient evidence in available. This 'evidence' will normally be a pair of values, one of which is a function of the key; examples are a plaintext value and the corresponding ciphertext, or a plaintext value and its MAC.

The most common example of use of a weak key is when the key is formed from a password that needs to be remembered by a human. In this situation the effective key length can be estimated from the set of values that are practically used as passwords, and is certainly much smaller than would be acceptable as the key length of any modern cryptosystem. A number of protocols have been designed specifically to hide the evidence needed to guess at weak keys. These are examined in some detail in Chap. 8.

1.4.9 Certificate Manipulation

In public key protocols the *certificate* of a principal acts as an offline assurance from a trusted authority that the principal's public key really does belong to that principal. Other principals which make use of a certificate are trusting that the authority has correctly identified the owner of the public key at the time that the certificate was issued. However, it is not necessarily expected that the authority is provided with evidence that the corresponding private key is actually held by the principal claiming ownership of the key pair. This leads to potential attacks in which the adversary gains a certificate asserting that a particular public key is its own, even though the adversary does not know the corresponding private key. If this public key is a function of an existing public key some undesirable consequences may arise.

An example of a certificate manipulation attack was given by Menezes *et al.* [551] on a key agreement protocol of Matsumoto *et al.* [526]. (This protocol, and

related ones, will be examined in some detail in Sect. 5.3.) Principals A and B possess public keys $y_A = g^{x_A}$ and $y_B = g^{x_B}$ respectively, and corresponding private keys x_A and x_B. Here g generates a suitable group in which the discrete logarithm problem is hard. Each public key is certified and so A and B possess certificates $Cert(A)$ and $Cert(B)$ respectively which contain copies of their public keys g^{x_A} and g^{x_B}. A normal protocol run proceeds as shown in Protocol 1.5, where r_A and r_B are random values chosen by A and B respectively.

Goal: Key agreement

1. $A \to B : g^{r_A}, Cert(A)$
2. $B \to A : g^{r_B}, Cert(B)$

Protocol 1.5: A protocol vulnerable to certificate manipulation (MTI protocol)

The shared key is $K_{AB} = g^{x_A r_B + x_B r_A}$, calculated by A as $(g^{r_B})^{x_A} y_B^{r_A}$ and by B as $(g^{r_A})^{x_B} y_A^{r_B}$. The adversary C engineers an attack by choosing a random value x_C, claiming that $g^{x_A x_C}$ is its public key, and obtaining a certificate for this public key. (Notice that C cannot obtain the corresponding private key $x_A x_C$.) C then masquerades as B in Protocol 1.5, and completes two runs of the protocol, one with A and one with B, as shown in Attack 1.3.

---

1. $A \to C_B : g^{r_A}, Cert(A)$
1′. $C \to B : g^{r_A}, Cert(C)$
2′. $B \to C : g^{r_B}, Cert(B)$
2. $C_B \to A : g^{r_B x_C}, Cert(B)$

Attack 1.3: Certificate manipulation attack on MTI protocol

After the attacking run is complete, A will calculate the key

$$K_{AB} = (g^{r_B x_C})^{x_A} (y_B)^{r_A} = g^{x_A x_C r_B + x_B r_A}$$

and B will calculate the key

$$K_{CB} = (g^{x_A x_C})^{r_B} (g^{r_A})^{x_B} = g^{x_A x_C r_B + x_B r_A}.$$

Thus A and B have found the same key, but A believes this key is known only to A and B while B believes it is known only to C and B. This is an example of an *unknown key-share attack*, which will be discussed in more detail in Section 5.1.3.

Attacks of this sort can be avoided by demanding that every principal demonstrates knowledge of the private key before a certificate is issued for any public key.

Such a demonstration is ideally achieved using zero knowledge techniques so that the trusted authority gains nothing useful about the private key. A more convenient method may be to have the private key owner sign a specific message or a challenge. More generally this process is part of *public key validation* which provides assurance that the public and corresponding private key have been properly generated as specified in the protocol. It is possible to give a formal treatment of security incorporating certificate manipulation attacks [136].

1.4.10 Protocol Interaction

Most long-term keys are intended to be used for a single protocol. However, it could be the case that keys are used in multiple protocols. This could be due to careless design, but may be deliberate in cases where devices with small storage capability are used for multiple applications (smart cards are the obvious example), or where a single certificate is used in multiple protocols, in multiple versions of a protocol, or in multiple ways within the same version of a protocol.

It is easy to see that protocols designed independently may interact badly. For example, a protocol that uses decryption to prove possession of an authenticating key may be used by an adversary to decrypt messages from another protocol if the same key is used. Kelsey *et al.* [423] gave several examples of how things can go wrong, and discussed the *chosen protocol attack*, in which a new protocol is designed by the adversary to attack an existing protocol. In Sect 6.10.2 we look at cross-protocol attacks on TLS where long-term keys are shared across different protocol versions.

Apart from limiting keys to be used in unique protocols, one method to prevent such attacks is to include the protocol details (such as a unique identifier and the version number) in an authenticated part of the protocol messages. Protocols with a security proof in the universal composability framework [181] are immune to such attacks.

1.5 Goals for Authentication and Key Establishment

Any attack on a protocol is only valid if it violates some property that the protocol was intended to achieve. In other words, all attacks must be considered relative to the protocol goals. Experience has proven that many protocol problems result when designers are unclear about the protocol goals they are trying to achieve. This in turn leads to disputes about whether protocol attacks are valid, since designers may regard the goals differently from analysers. Gollmann [311] recognised that it is a difficult matter to decide exactly what is meant by commonly used words such as ‘authentication’; even if everyone has a general idea of the meaning of such a word, the interpretation may vary with the protocol. It turns out that although most authors can agree on general definitions, their ideas diverge when precision is required.

Clarity in describing protocol goals is desirable for all parties concerned. Designers have to make use of the protocol goals to justify each message field and all cryptographic processing. Experience shows that protocols with well-defined goals

are streamlined and transparent to analyse. Analysers make use of protocol goals to direct their attempts to find attacks or prove they do not exist.

Many authors have considered the question of what are the appropriate goals for cryptographic protocols, mainly in the context of protocol analysis. Definitions of various goals also appear in a number of standards for cryptographic protocols. However, there is a lack of agreement as to what are the desirable goals for authentication and key establishment, as well as precise definitions for these goals.

Let us consider an initial example to illustrate the divergent paths that may be taken in assessing protocol goals and attacks. In Protocol 1.6 principals A and B wish to authenticate each other, using an initially shared key K_{AB}. We will discuss below some possible meanings of *authenticate*, but for now we will assume that both users wish to know that their communicating peer is in possession of K_{AB}.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : \mathsf{MAC}_{K_{AB}}(ID_B, ID_A, N_A), N_B$
3. $A \rightarrow B : \mathsf{MAC}_{K_{AB}}(ID_A, ID_B, N_B)$

Protocol 1.6: Example protocol

Both A and B choose nonces, N_A and N_B respectively, and generate a tag from a message authentication code using the shared key. Protocols similar to this one have been published in the literature (although we do not believe this exact one has been suggested before). An attack on Protocol 1.6 is possible which is very similar to some previously published attacks [103, 253]. In Attack 1.4, principal A is used as an 'oracle' by the adversary C.

1. $C_A \rightarrow B : N_C$
2. $B \rightarrow C_A : \mathsf{MAC}_{K_{AB}}(ID_B, ID_A, N_C), N_B$
1′. $C_B \rightarrow A : N_B$
2′. $A \rightarrow C_B : \mathsf{MAC}_{K_{AB}}(ID_A, ID_B, N_B), N_A$
3. $C_A \rightarrow B : \mathsf{MAC}_{K_{AB}}(ID_A, ID_B, N_B)$

Attack 1.4: Attack on Protocol 1.6

From the view of B the protocol has completed normally. However, the protocol has not run correctly and it is certainly not the case that A is the communication peer of B. On the other hand, if the protocol goal for B was to establish that A is ready and willing to communicate with him then the protocol has not failed. Indeed we may note that A was sent a challenge by someone purporting to be B and replied with a message to the effect that she was prepared to communicate with B. This example

illustrates how careful we must be to evaluate attacks against definitions of protocol goals.

In the next few sections we will consider various possible definitions for the fundamental goals of authentication and key establishment protocols and explain the reasons for our choice of definitions that we will use in subsequent chapters. This will lead to further goals which are desirable in many AKE protocols. This chapter concentrates on goals for protocols where the session key is shared between two principals; additional goals which may be useful in multi-party protocols are discussed further in Chap. 9.

1.5.1 Models of Security

As well as deciding what are the required goals of a protocol, we need to agree on the attack model that will be used to decide whether a goal is achieved. This concerns what actions we allow to the adversary. Generally models allow at least two general capabilities to the adversary.

- The adversary controls the communications between all principals, which means that the adversary can observe all messages sent, alter messages, insert new messages, delay messages or delete messages. This may be more than is achievable by an adversary in practice, but by assuming a more powerful adversary we achieve a stronger form of security.
- The adversary can obtain any session keys used in different runs of the protocol. This reflects the typical requirement that session keys should be independent of each other.

In addition to allowing the adversary to obtain session keys unrelated to the session key in current use, some models also allow the adversary to obtain long-term keys. This is often called *corruption* of principals since it allows the adversary to participate in protocol runs as a normal participant. Of course an adversary in possession of the long-term key of Alice can authenticate as Alice, so we need to restrict what is a successful attack in such circumstances. Often we need to prevent corruption of any of the principals involved in the protocol run which is the target of the attack. However, this restriction is not necessary as we will see when we consider forward secrecy and key compromise impersonation a little later.

Later, in Sect. 1.6 and Chap. 2, we will look in more detail at formal models of security for AKE protocols. In this section we continue with an informal look at protocol goals.

1.5.2 Key Establishment or Authentication?

In the early literature on cryptographic protocols it was common to refer to all protocols concerned with setting up session keys as ‘authentication protocols’. This is not entirely satisfactory because some protocols that set up session keys provide no authentication of one party to the other, while other protocols designed to provide entity

authentication involve no session key. Therefore it has become usual to distinguish between two types of protocols. We will use the term *entity authentication protocols* for protocols that provide only authentication while using the term *key establishment protocol* (also often called a *key exchange protocol*) for protocols that involve setting up a new key, typically for a communications session.

Gollmann [311] put forward a number of different options for what could be meant by authentication. The first one is as follows.

Gol1. The protocol establishes a fresh session key, known only to the participants in the session and possibly some trusted third parties.

This goal may be achieved even though each party knows nothing about even the existence of the other party, let alone whether the other party is willing to engage in a session. Thus this is a goal about key establishment rather than entity authentication.

The second goal suggested by Gollmann is as follows, in which A and B are the protocol principals.

Gol2. A cryptographic key associated with B was used in a message received by A during the protocol run. The protocol run is defined by A's challenge or a current timestamp.

This is a goal concerning entity authentication. It says nothing about a new session key and can be satisfied by a protocol that is not concerned with key establishment.

There appears to be more dissent in the literature regarding the nature of entity authentication than there is with regard to key establishment. One reason for this may be that it is difficult to be clear about the purpose of entity authentication in the absence of key establishment. Diffie *et al.* [253] say that it is 'accepted that these topics should be considered jointly rather than separately', while Bellare and Rogaway [78] go further in stating:

> ...entity authentication is rarely useful in the absence of an associated key distribution, while key distribution, all by itself, is not only useful, but it is not appreciably more so when an entity authentication occurs along side. Most of the time entity authentication is *irrelevant*: it doesn't matter if you have been speaking to a given communication partner, in that by the time you become aware of [an authenticated entity] there will be no particular reason to believe that the partner is still 'out there' anyway.

In our view there are situations when entity authentication by itself may be useful, such as when using a physically secured communication channel. But it is important to appreciate exactly what it provides.

Syverson and van Oorschot [705] identified what they termed six 'generic formal goals'. These are expressed in English in Table 1.4; for formal statements readers should refer to their paper. There are clearly dependencies between various of these goals. For example, **SVO2** is a stronger property than **SVO1**. Furthermore, it is not clear why these particular goals are important; for example, it might be questioned whether secure key establishment is useful without key freshness. To be fair to these

Table 1.4: Syverson and van Oorschot's generic formal goals for protocols

SVO1	**Far-end operative** *A* believes *B* recently 'said' something.
SVO2	**Entity authentication** *A* believes *B* recently replied to a specific challenge.
SVO3	**Secure key establishment** *A* has a certain key *K* that *A* believes is good for communication with *B*.
SVO4	**Key confirmation** In addition to SVO3, *A* has received evidence confirming that *B* knows *K*.
SVO5	**Key freshness** *A* believes a certain key *K* is fresh.
SVO6	**Mutual understanding of shared key** *A* believes that *B* has recently confirmed that *B* has a certain key *K* that *B* believes is good for communication with *A*.

authors, they state that it is *not* intended as a 'definitive list of *the* goals that a key agreement or key distribution protocol should meet'.

The least of these goals, **SVO1**, simply says that *B* has recently done something, independent of any other entities or keys. We will refer to this goal a few times later, and sometimes use descriptions such as *B* is *alive* to describe the assurance that *A* achieves, or simply say that a protocol which reaches this goal provides *liveness*.

So far we hope to have convinced the reader that there is no unanimity, either on what the goals of authentication and key establishment protocols should be or on how to define those goals. We will now look in more detail at different classes of goals which can be considered in three categories: those concerning entity authentication; those concerning key establishment and those which are optional additions to key establishment.

1.5.3 Entity Authentication

The ISO Security Architecture [375] defines entity authentication as 'the corroboration that an entity is the one claimed'. This is not as precise a definition as one might like since it does not explain which entity is the subject. Menezes *et al.* [550] gave a more comprehensive definition as follows.

Definition 11. Entity authentication *is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e. is active at, or immediately prior to, the time the evidence is acquired).*

Protocol 1.7 is an example that seems to provide entity authentication of *B* to *A* satisfying this definition. *A* sends her nonce to *B*, who replies by signing it. It seems clear that *A* knows that *B* must have engaged in this protocol and that the signature is fresh.

Definition 11 is a clear explanation, but does not go as far as is possible or perhaps even desirable. Imagine user *A* having received some messages in an entity authentication protocol. What is it that she can hope to have learnt from those messages?

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : \mathsf{Sig}_B(N_A)$

Protocol 1.7: A simple authentication protocol

One aspect is that user B is really out there now, somewhere on the network. This is the far-end operative property (**SVO1**) that we have already seen. The only other assurance that seems relevant is to know that B is ready to engage in communication with A.

Attack 1.5 on Protocol 1.7 shows why this extra assurance may be desirable. In this protocol run A can verify that the signature received was formed by B, yet B has not indicated that he is aware of A. In some sense we may even accept that the adversary C has provided assurance that he is B. On the other hand C does not appear to be doing anything other than faithfully replaying messages between A and B. The next definition tries to capture the notion of one principal being prepared to communicate with another principal.

1. $A \rightarrow C_B : N_A$
1′. $C \rightarrow B : \ N_A$
2′. $B \rightarrow C : \ \mathsf{Sig}_B(N_A)$
2. $C_B \rightarrow A : \mathsf{Sig}_B(N_A)$

Attack 1.5: An attack on Protocol 1.7

Definition 12. *A principal A is said to have* knowledge of B as her peer entity *if A is aware of B as her claimed peer entity in the protocol.*

Considering again the fundamental elements used in authentication protocols this seems to be all that can be achieved. Messages can convey either freshness, or principals with which communication is desired. Combining these leads to a strong definition of entity authentication. (There are several alternative ways of expressing this property which all indicate that A is authenticated to B only if A is prepared to engage in communications with B.)

Definition 13. Strong entity authentication *of A to B is provided if B has a fresh assurance that A has knowledge of B as her peer entity.*

An enhanced version of Protocol 1.7 can provide this stronger assurance. Protocol 1.8 provides strong entity authentication of B to A. It may be checked that an adversary C cannot use Attack 1.5 to convince A that B is aware of A as his peer entity.

According to Definition 13, there are two subgoals of entity authentication:

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : \mathsf{Sig}_B(ID_A, N_A)$

Protocol 1.8: Another simple authentication protocol

- A (once) has had knowledge of B as her peer entity;
- A is operative.

The latter of these is the far-end operative property discussed before in goal **SVO1** of Table 1.4. Notice that it is straightforward to extend Definition 13 to a multi-party goal of entity authentication of a group of users U to A: the principal A is freshly aware of the principals in U as her peer entities.

Entity authentication is a service that is provided by one entity to one or more other entities. Most often we are concerned with the interaction between two entities and then it is common to differentiate between two situations.

Definition 14. Mutual authentication *occurs if both entities are authenticated to each other in the same protocol.* Unilateral authentication *(sometimes called* one-way authentication*) occurs if only one entity is authenticated to the other.*

When multiple entities are involved there are many possibilities for different combinations of entity authentication; in principle a protocol can authenticate any subset of the entities to any other subset. In practice there seem to be few situations where a complex rule for who should be authenticated to whom is useful.

1.5.4 Key Establishment

Menezes *et al.* [550] gave the following definition for key establishment.

> *Key establishment* is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use.

This definition can be extended and made more specific. One way to understand the possible goals for key establishment is to consider what may be achieved with typical message components. There are three types of message components that are conventionally used in cryptographic protocols for key establishment and entity authentication. These are:

1. **keys**, which may be long-term keys or session keys;
2. **identifiers** for protocol principals;
3. **nonces**, which may be random values, timestamps or counters.

These components are combined and processed with cryptographic mechanisms to provide confidentiality and/or authentication. For key establishment a new session key may be associated with a nonce, or with identifiers of protocol principals. In practice a session key is not of any use unless it is known to be fresh and it is known

which other entities may possess it. Most authors agree that secure key establishment should require the two goals that the key is known to be fresh and is known only to the other protocol participant(s), possibly including trusted third parties. This is often referred to as establishing a *good key*.

Definition 15. *A shared session key is a* good key *for A to use with B only if A has assurance that:*

- *the key is fresh (*key freshness*);*
- *the key is known to at most A and B and any mutually trusted parties (*key authentication*).*

The second of these properties is often also called *implicit key authentication*. (As pointed out by Gollmann [312], this property may equally be regarded as being about confidentiality of the key.) It can be argued that key authentication must imply key freshness, since a key that is not fresh cannot be guaranteed to be kept confidential. From this viewpoint a separate requirement for key freshness is not required.

Although not a common requirement, public session keys are certainly possible. The above definition is easily extended to this case.

Definition 16. *A public session key is a* good key *for A to use with B only if:*

- *the key is fresh (*key freshness*);*
- *the corresponding private key is known only to B (*key authentication*).*

An interesting additional goal has been considered by some authors, including Janson and Tsudik [394].

Definition 17. Key integrity *is the property that the key has not been modified by the adversary, or equivalently only has inputs from legitimate principals.*

- *For a key transport protocol, key integrity means that if the key is accepted by any principal it must be the same key as that chosen by the key originator.*
- *For a key agreement protocol, key integrity means that if a key is accepted by any principal it must be a known function of only the inputs of the protocol principals.*

Note that there is no contradiction if a key establishment protocol provides the good key property but fails to provide key integrity. It is quite conceivable that an adversary may be able to disturb a protocol, whether it is a key transport or a key agreement protocol, in such a way that the key has been changed from its 'correct' value but is still fresh and unknown to the adversary. Protocol 3.15 is an example which provides key integrity.

1.5.5 Key Confirmation

Definition 18. Key confirmation *of B to A is provided if A has assurance that key K is a good key to communicate with B, and that principal B has possession of K.*

Key confirmation provides evidence that the partner has the same key but leaves open the possibility that the key is intended by the partner for a different communication session (with the assumption that the partner may be engaged in several conversations). Key confirmation provides evidence that the partner wishes to communicate with some entity, and so implies the far-end operative property, but may not imply entity authentication. Key confirmation is typically achieved by having both parties send each other some fresh data using a cryptographic function depending on the key; this is often referred to as a *handshake*.

Shoup [674] has put forward the idea that key confirmation is not a valuable security property. His point is that it is not really useful for a principal to know that the partner has, or can obtain, possession of the session key, but rather that the partner has *accepted* the session key. It is never possible to guarantee this for both parties, since one party must always finish, and therefore accept, without the other party knowing. Nevertheless, the property as stated may or may not be achieved, perhaps even mutually. As with entity authentication, we prefer not to judge whether this property is a useful one.

It can be seen that Definition 18 requires that the identified other party has received the session key. Not all authors use this definition of key confirmation. For example, the following definition is given in the *Handbook of Applied Cryptography* [550]:

> *Key confirmation* is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

This contrasts with the definition in the ISO/IEC 11770-2 key management standard [376].

> *Key confirmation:* the assurance for one entity that another identified entity is in possession of the correct key.

The following definition is also taken from the *Handbook of Applied Cryptography*.

Definition 19. Explicit key authentication *is the property obtained when both (implicit) key authentication and key confirmation hold.*

Notice that it does not matter for this definition whether or not key confirmation includes identification of the other party in possession of the key. This is because implicit key authentication assures B that only A may have the key, so any party that shows possession of the key must be entity A.

Mutual belief in the key, following **SVO6** in Table 1.4, adds to key confirmation that the key is known by the partner to be good. (Actually, **SVO6** does not require the good key property, but seems of little value if it does not also hold.) It provides both key confirmation and entity authentication since if the partner has acknowledged that the key is good for the communication this can be taken as a confirmation that the partner is willing to communicate.

Definition 20. Mutual belief in the key *K is provided for B only if K is a good key for use with A, and A wishes to communicate with B using key K which A believes is good for that purpose.*

1.5.6 Example: STS Protocol

We turn to an example to focus discussion on the subtleties of assessing protocol attacks against published goals. The station-to-station (STS) protocol [253] uses a digital signature in the exchanged messages to add authentication to the well-known Diffie–Hellman protocol [252]. This uses arithmetic in a multiplicative group with generator g. Exponents x and y are chosen randomly by A and B respectively and are used to form the session key $K_{AB} = g^{xy}$. Protocol 1.9 shows the messages in a successful protocol run.

1. $A \rightarrow B : g^x$
2. $B \rightarrow A : g^y, \{\mathsf{Sig}_B(g^y, g^x)\}_{K_{AB}}$
3. $A \rightarrow B : \{\mathsf{Sig}_A(g^x, g^y)\}_{K_{AB}}$

Protocol 1.9: STS protocol

Here $\mathsf{Sig}_X(.)$ represents the signature by the principal X on the string in the brackets, while $\{M\}_K$ denotes symmetric encryption of message M using key K. The particular signature algorithm chosen does not matter for the protocol. Consider how the good key goal is achieved for A.

1. The signature in message 2 can only be formed by B.
2. It is not a replay from an old protocol run since A knows that g^x was fresh.
3. The signature alone does not imply that B knows K_{AB}. Therefore the encryption with K_{AB} is necessary to provide assurance that B really knows K_{AB}.

Thus it appears that A gains key confirmation, as well as a good key with B, from message 2. With regard to authentication goals, it seems clear that both users achieve liveness of the other, since each receives a signed message containing a value they know to be fresh. Strong entity authentication, in the sense of Definition 13, is more problematic since there is no explicit inclusion of identifiers in the signed messages which could be used to deduce the identity of the desired communications partner.

Lowe [502] has proposed an attack on the STS protocol. To be quite precise, the protocol analysed by Lowe is slightly different in that principal identifiers are added to each message to give the modified version shown in Protocol 1.10. The addition of the identifiers appears to make no material difference to the protocol since they are attached as plaintext and so are vulnerable to both eavesdropping and modification. However, their addition is critical to the interpretation of the attack. Lowe [502] states that the identifiers were ‘included to make the subsequent explanations clearer’.

1. $A \rightarrow B : ID_A, ID_B, g^x$
2. $B \rightarrow A : ID_B, ID_A, g^y, \{\mathsf{Sig}_B(g^y, g^x)\}_{K_{AB}}$
3. $A \rightarrow B : ID_A, ID_B, \{\mathsf{Sig}_A(g^x, g^y)\}_{K_{AB}}$

Protocol 1.10: STS protocol modified to include identifiers

Lowe's attack does not affect the key establishment properties but is addressed at whether entity authentication is achieved. Suppose that C is an adversary who wishes to attack the protocol. C intercepts a protocol run started by A and masquerades as B. In parallel, C starts a protocol run with B. Attack 1.6 shows an attacking run, where C_B denotes C masquerading as principal B.

1. $A \rightarrow C_B : ID_A, ID_B, g^x$
1′. $C \rightarrow B : ID_C, ID_B, g^x$
2′. $B \rightarrow C : ID_B, ID_C, g^y, \{\mathsf{Sig}_B(g^y, g^x)\}_{K_{AB}}$
2. $C_B \rightarrow A : ID_B, ID_A, g^y, \{\mathsf{Sig}_B(g^y, g^x)\}_{K_{AB}}$
3. $A \rightarrow C_B : ID_A, ID_B, \{\mathsf{Sig}_A(g^x, g^y)\}_{K_{AB}}$

Attack 1.6: Lowe's attack on Protocol 1.10

The attack is very simple: C is doing little more than relaying each message that passes between A and B. What is the result? In the attacking run B has no indication that A has engaged in the protocol and yet A has completed a successful run and accepted that her partner is B. Is this a successful attack on the STS protocol? If we apply the same attack to the original STS protocol (Protocol 1.9) without identifiers, we see that C does nothing more than relay messages between A and B, so how can this constitute an attack?

Diffie *et al.* [253] defined security based on matching conversations. In other words, for a secure protocol the accepting parties should agree on the messages exchanged in the protocol run. (Section 2.2 provides a detailed discussion of matching conversations.) When applying Attack 1.6 to their original protocol the conversation of A does indeed match that of B and so the attack does not violate their definition of security. A reasonable conclusion may be that the attack is invalid on the STS protocol as specified by its authors and in accordance with their definition of security. But what about the modified protocol? Is it really different from the original and is the attack valid in that case? The answer must depend on the intended goals.

- After the attacking run it is clear that the good key goal has not been broken.
- Key confirmation has indeed been achieved: A can be sure that B knows the shared key.
- A does not know that B knows the key is good for use with A. In other words the *mutual belief in key* goal (Definition 20) is not achieved for A.

- The attack shows that A would be wrong to conclude, after a successful run, that B wishes to communicate with her. Thus strong entity authentication (using Definition 13) is not achieved.

We conclude that the attack would be valid if either mutual belief in the key or strong entity authentication were protocol goals. However, it is clear from the paper of Diffie *et al.* that they did not regard these as goals of their protocol. The insight gained from the attack is therefore that the protocol does not meet extended goals that could be desired by some users.

Lowe [502] proposed that the identity of the other party be included in the signatures in order to prevent the attack. This also allows an informal argument that strong entity authentication is achieved, if the included identifier is interpreted as the name of the entity with which communication is desired.

1.5.7 Forward Secrecy

The idea of forward secrecy is that when a long-term key is compromised, session keys that were previously established using that long-term key should not be compromised too. Key agreement protocols in which the long-term key is used only to authenticate the exchange provide typical examples of protocols with forward secrecy. Key transport protocols in which the long-term key is used to encrypt the session key cannot provide forward secrecy.

Definition 21. *A key establishment protocol provides* forward secrecy *if compromise of the long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals.*

Definition 22. *A protocol provides* partial forward secrecy *if compromise of the long-term keys of one or more specific principals does not compromise the session keys established in previous protocol runs involving those principals.*

If a protocol does not provide (full) forward secrecy then partial forward secrecy may still be useful if there is an asymmetry in the roles of the principals involved. For example, in a client–server protocol it may be deemed more likely that a client long-term key will be compromised than that the server key will be. In this situation partial forward secrecy, in which compromise of client long-term keys does not compromise old session keys, is a useful property.

The term ‘forward secrecy’ seems to have been coined by Günther [336]. In fact he used the term *perfect forward secrecy* but, in common with other authors, we have dropped the word ‘perfect’; this is not only for the sake of brevity, but also because it gives connotations with the term ‘perfect secrecy’ which refers to unconditional (information-theoretic) security which is not relevant here.

The critical concept in providing forward secrecy is an *ephemeral public key*; this is a public key that is used only for the duration of the key establishment protocol and is then destroyed along with the corresponding private key. If the long-term public key is used only for authenticating the session key then the session key cannot be

recovered without the ephemeral private key. The most commonly used ephemeral keys are of the type needed in the Diffie–Hellman protocol which is examined in detail in Chap. 5. This is not the only type of ephemeral public key however; ephemeral keys can be used for any public key cryptosystem. Furthermore, it is a common misconception that forward secrecy can be achieved *only* with key agreement.

As an example consider Protocol 1.11 which provides key transport between A and B. Here K_T is an ephemeral public key chosen by A uniquely for this session. This key is sent to B and signed by A together with a nonce N_A chosen by A. B then uses this ephemeral key to transport the session key K_{AB} confidentially back to A. Here $\mathsf{Enc}_T(.)$ denotes encryption with K_T and h is a one-way hash function.

1. $A \rightarrow B: K_T, N_A, \mathsf{Sig}_A(K_T, ID_B)$
2. $B \rightarrow A: \mathsf{Enc}_T(K_{AB}), \mathsf{Sig}_B(h(K_{AB}), ID_A, N_A)$

Protocol 1.11: Key transport protocol providing forward secrecy

The private key corresponding to the ephemeral public key should be destroyed by A immediately after the session key is recovered. It can be seen that compromise of the long-term signature keys will not help an adversary in obtaining the session key.

The long-term keys used in a protocol providing forward secrecy may be either shared or public. Consider Protocol 1.12, in which A and B share long-term keys K_{AS} and K_{BS} with server S. Random values r_A, r_B and K_S are chosen by A, B and S respectively. The protocol includes Diffie–Hellman-like key agreement, using the generator g of some multiplicative group, together with encryption using the long-term keys.

1. $A \rightarrow S: ID_A, ID_B$
2. $A \rightarrow B: ID_A, g^{r_A}$
3. $S \rightarrow B: \{ID_A, ID_B, K_S\}_{K_{BS}}$
4. $S \rightarrow A: \{ID_A, ID_B, K_S\}_{K_{AS}}$
5. $B \rightarrow A: ID_B, g^{r_B}$

Protocol 1.12: Server-based protocol providing forward secrecy

The session key K_{AB} is calculated by A as $K_{AB} = (g^{r_B})^{r_A K_S}$ and by B as $K_{AB} = (g^{r_A})^{r_B K_S}$. Once the ephemeral values r_A and r_B are destroyed the session key is protected against compromise of the long-term keys shared with S.

1.5.8 Weak Forward Secrecy

Protocols which provide forward secrecy are often more expensive, either computationally or with regard to communications complexity, than those without it. Therefore there is value in considering a weakened security property which may be less costly. Bellare *et al.* [74] and later Krawczyk [453] defined *weak forward secrecy* to be similar to normal forward secrecy but where the adversary is forbidden to take an active part in the protocol run which is being targeted.

Definition 23. *A protocol provides* weak forward secrecy *if compromise of the long-term keys of one or more specific principals does not compromise the session keys established in previous protocol runs involving those principals when the adversary did not take an active part in the session under attack.*

This means that in a protocol providing weak forward secrecy the victim principal executes the session under attack with a legitimate party whose messages are transmitted correctly to the victim. The adversary is not allowed to interfere in the session. Such a restriction could make sense in a scenario where the adversary is eavesdropping on a large number of sessions and has not yet decided which ones to attack.

In order to differentiate from weak forward secrecy we will sometimes use the term *strong forward secrecy* to denote the normal version of forward secrecy where the adversary is allowed to be active in the target session. Krawczyk [453] and Boyd and González Nieto [142] provided generic attacks which show that strong forward secrecy is not possible if either of the following applies:

- the protocol messages are independent of the long-term key of the sender;
- the adversary is allowed to reveal ephemeral secrets of the partner party to the test session.

An example of a protocol with only weak forward secrecy is a key agreement protocol of Matsumoto *et al.* [526]. (This protocol will be examined in Sect. 5.3.) Principals A and B possess public keys $y_A = g^{x_A}$ and $y_B = g^{x_B}$ respectively, and corresponding private keys x_A and x_B. Here g generates a suitable group in which the discrete logarithm problem is hard. A normal protocol run proceeds as shown in Protocol 1.13, where r_A and r_B are random values chosen by A and B respectively.

Goal: Key agreement. Shared key K_{AB} is (derived from) shared secret $g^{r_A r_B}$.

1. $A \rightarrow B : y_B^{r_A}$
2. $B \rightarrow A : y_A^{r_B}$

Protocol 1.13: A protocol with weak forward secrecy (MTI protocol)

The shared key is $K_{AB} = g^{r_A r_B}$, calculated by A as $(y_A^{r_B})^{x_A^{-1} r_A}$ and by B as $(y_B^{r_A})^{x_B^{-1} r_B}$. The active adversary C engineers an attack by choosing a random value

r_C, and replying to the message of A with the response $y_A^{r_C}$. Then A completes the protocol normally and computes the shared secret as $K_{AB} = g^{r_A r_B}$. Once A has sent some data, encrypted with K_{AB}, to what A assumes is B, C can close the connection. If this protocol provided strong forward secrecy then C would be allowed to obtain the long-term key of B, x_B. Then C can compute the shared key as $(y_B^{r_A})^{x_B^{-1} r_C}$ and thus recover the secrets of A. Note that this attack no longer applies if C is forbidden from being active in the session under attack since it needs to choose its r_C value during the protocol run.

Many well-known two-message protocols provide only weak forward secrecy. There has been some misunderstanding in the literature regarding whether two-message protocols can achieve strong forward secrecy at all. In fact two-message protocols proven to have strong forward secrecy have been known since at least 2004 [397] (see Protocol 5.39). Indeed, there is even a known way to achieve forward secrecy in *one-message* protocols [337], although this requires a long-term key which is updated over time.

One generic way to ensure that a secure protocol with weak forward secrecy actually provides strong forward secrecy is to add key confirmation. This is because key confirmation requires an active adversary to know the session key before being allowed to obtain the long-term keys, so if the protocol can be broken then it is broken even without giving the long-term keys to the adversary.

Another way to ensure strong forward secrecy is to add explicit authentication to the messages exchanged. Boyd and Gonzalez Nieto [142] provided a generic method to add strong forward secrecy to any protocol by adding a MAC tag to the messages. A similar method using digital signatures was provided by Cremers and Feltz [235] while a protocol using a specific signature was designed by Huang [366].

1.5.9 Key Compromise Impersonation

When an adversary learns the long-term key of Alice the adversary can impersonate Alice to other principals until the compromise is detected and the long-term key is revoked.[1] Key compromise impersonation refers to an attack in which the adversary uses Alice's compromised long-term key to masquerade *to Alice* as another user.

Definition 24. *A protocol provides* resistance to key compromise impersonation *if compromise of a long-term key of a principal A does not allow the adversary to masquerade to A as a different principal.*

The typical situation in which key compromise impersonation is possible is when the protocol gives assurance only that each entity has any one of a pair of long-term keys. This situation is commonly found in key agreement protocols where the security is often based on the property that a particular value can be calculated with knowledge of the long-term key of either of the two principals.

[1] However, there are measures that can be taken to protect compromised signature keys against abuse, as discussed by Just and van Oorschot [406].

A simple example is the static Diffie–Hellman value $g^{x_A x_B}$ which can be computed by either of A, with public key $y_A = g^{x_A}$, or B, with public key $y_B = g^{x_B}$. A protocol which we will look at later (Protocol 5.12) combines this static value with an ephemeral Diffie–Hellman value to obtain the session key. Since these values can be computed with the long-term private key of either A or B it is vulnerable to key compromise impersonation.

Protection against key compromise impersonation seems to require use of asymmetric cryptography. If each party can verify that the correct private key was used then the other party must be present. For example, if each party receives a digital signature from the other, the adversary cannot forge the signature from B if it only has A's private key. Several examples of protocols secure against key compromise impersonation are presented in Chap. 5.

1.5.10 Deniability

Deniability is a privacy property of security mechanisms that is desirable in certain circumstances. The idea is that it should be possible for a user employing such a mechanism to later deny taking part in the communication. Of course it is possible to conduct the communication without any cryptographic mechanisms at all, and then the communication can always be denied. However, it may be desirable to provide authentication to the receiver and to set up a secure channel to protect the confidentiality and integrity of the information being sent. A typical scenario for such communication is an 'off-the-record' disclosure from an insider, I, in some organisation to a news reporter, R. The reporter may want to verify the source of the data and the disclosing insider may want to use a secure channel for the communication. Yet I would like to ensure that if R later tries to implicate I in the communication, then I can deny having taken part.

Definitions for deniability in key establishment were preceded in the literature by definitions of deniable encryption [176] and deniable authentication [43]. Deniability of communications can be achieved if a key establishment protocol is used with the deniability property. Once I is able to convincingly deny having taken part in establishing the key, I can also deny having used the key to form a secure communications channel with R. Both of these properties can be achieved by first establishing a session key in a deniable key establishment protocol and then using standard encryption and authentication schemes keyed by the session key.

Informally, deniability should prevent anybody from convincing an impartial judge that a particular protocol principal took part in the protocol. We will call the adversary attempting to prove participation the *accuser*. The accuser may be an outsider, not taking part in the protocol run, or an insider who is running the protocol with I. In the former case it is much easier to achieve deniability; in particular, any protocol using protocol messages which depend only on public information is deniable against outsiders. This is because the transcript of the protocol could always have been produced by the accuser alone and so the judge will not be convinced. An example is Protocol 1.13 above, since the protocol messages are simply random values in a group. Many existing protocols achieve this property of *outsider deniability*.

When the accuser is allowed to run the protocol with I, deniability is much harder to achieve – in this case the accuser has the opportunity to try to construct messages in such a way that the key could only have been constructed by the victim. If the accuser can link the session key to the protocol session and show that only the victim could have formed the key then the judge can be convinced.

In most solutions for deniability it is assumed that there is no special communication between the accuser and the judge prior to the protocol run; there is only a generic set-up process. We can envisage stronger adversaries who can communicate with the judge prior to the protocol run, or even during the protocol itself. It is not clear that these stronger adversaries are realistic. A judge who is also a protocol participant will always be convinced and this is basically the situation when the judge can communicate directly with the accuser during the protocol run.

Mao and Paterson [522] seem to have been the first to discuss deniability for key establishment and considered a variety of informal definitions. The first is similar to what we called outsider deniability above, while the second and third are a form of insider deniability.

- For *weak deniability* there should be no values, such as digital signatures, which can be used to identify the protocol principals.
- For *strong deniability* the accuser is a peer of the victim willing to divulge ephemeral secrets used in the protocol run.
- For *complete deniability* the accuser is willing to divulge long-term secrets used in the protocol run.

Mao and Paterson provided variants of the Internet Key Exchange (IKE) protocols which intuitively satisfy these definitions using identity-based keys.

Di Raimondo *et al.* [248] provided the first formal definitions for deniable key establishment. Their definitions are based on the idea of *simulatability*, similar to the notion of zero knowledge. The intuition is that if any one party, who may be an insider, could alone have produced a protocol transcript that is indistinguishable from a normal protocol run, then the judge should never be convinced that the victim was active in the protocol run. There are some subtleties regarding the precise restrictions on the simulator which are important, at least in theoretical terms [602].

A simple and efficient deniable key exchange protocol was proposed by Jiang and Safavi-Naini [400] shown in Protocol 1.14. The protocol uses a trapdoor permutation, such as RSA, to send a random value from each party to the other together with a hashed value to prove knowledge of the value. We denote the trapdoor permutation simply as public key encryption in Protocol 1.14, although simpler instantiations may be possible. The session key, K_{AB}, is then a hash of the random values from each party and the party identifiers. Although the protocol is very efficient it does not provide forward secrecy because the session key is computed from a hash of the random values chosen by each party.

Yao and Zhao [753] designed a protocol suitable for use with Internet Key Exchange. This protocol uses MACs instead of signatures where the MAC keys can be fully simulated by either party. Their security proof uses the random oracle model and depends on the so-called *fresh-challenge knowledge of exponent* assumption.

Shared information: security parameter k; random oracle H.

A		B
$s \in_R \{0,1\}^k$		
	$\xrightarrow{\mathsf{Enc}_B(s), H(s, ID_A, ID_B)}$	$r \in_R \{0,1\}^k$
	$\xleftarrow{\mathsf{Enc}_A(r), H(r, s, ID_A, ID_B, 0)}$	
$K_{AB} = H(s, r, ID_A, ID_B, 2)$	$\xrightarrow{H(s, r, ID_A, ID_B, 1)}$	$K_{AB} = H(s, r, ID_A, ID_B, 2)$

Protocol 1.14: Protocol of Jiang and Safavi-Naini [400]

They also proved both forward secrecy and key compromise impersonation (KCI) resistance.

Cremers and Feltz [227] proposed a one-round protocol which provides full forward secrecy as well as deniability. Since their protocol includes signatures of both parties, strong deniability is not possible. However, they achieve instead a property which they call *peer-and-time deniability*; similar to the peer independence property of Di Raimondo *et al.* [248], this property allows users to deny communication with a particular party, in addition to denying the time when communication may have taken place. This latter property prevents an accuser from showing that a particular party was active after a certain time.

Bohli and Steinwandt [120] defined deniability for group key exchange and provided a four-round protocol which satisfies their definition. Following this other authors have proposed other constructions but with varying formal definitions of deniability. Zhang *et al.* [773] propose a three-round deniable group key exchange protocol. Neupane *et al.* [583] defined a compiler to convert any unauthenticated group key exchange protocol into one providing deniability as well as standard security properties, at the cost of one additional round.

Deniability has not been a property of prime interest to AKE protocol designers. Perhaps this is because it is not concerned at all with keeping the identity of the protocol participants hidden, only that they can plausibly claim that they were not participating. When privacy is of high concern to protocol principals they may be more interested in hiding their identities altogether. We look at this goal next.

1.5.11 Anonymity

It may appear to be a contradiction to consider anonymity when our focus is on protocols to provide authentication or authenticated key exchange. However, there are situations where both can make sense together. One is where the goal is to remain anonymous only to outsiders, while the legitimate parties authenticate only between themselves. Another is where two parties communicate with one party, remaining anonymous while the other authenticates. Following the terminology of Goldberg *et*

al. [308] we call these external and internal anonymity and will consider these two situations separately below. First we note a couple of different flavours of anonymity.

- It is possible that all protocol principals remain anonymous to each other. For example, it is simple to use plain Diffie–Hellman without any authentication. However, it is not clear what security can be provided in such a situation.
- A way to provide limited anonymity is for users to authenticate as being a member of some well-defined group. Special digital signatures, such as group signatures [190] and ring signatures [631], have been designed for this purpose. They could be used to authenticate key exchange messages, for example to sign Diffie–Hellman key exchange. There are also anonymous password-based key exchange protocols [723] which allow users to authenticate as being a member of a group by using a low-entropy password.

External Anonymity

A protocol achieves *external anonymity* if an adversary observing the protocol is unable to determine the identity of at least one of the protocol principals. This form of anonymity is not difficult to achieve in two-party protocols if it is desired to hide the identity of only one of the principals. In many applications, an AKE protocol is run between a client and a server; there may be no need to hide the identity of the server but the client identity may be more sensitive. For example, if the client is a mobile device it is often desired to hide its identity to prevent tracking of its physical location.

An example of a protocol designed to provide external anonymity is the Oakley protocol, examined later in this book (see Protocols 5.28 and 5.29). The client is assumed to have the public key of the server and can use it to encrypt its identity in addition to a secret input to the session key computation. Once the server has received the client identity, it can obtain the public key of the client (that too could have been sent encrypted by the client) and use that to hide its own input to the session key. Thus both client and server ultimately authenticate each other, but the client identity is never sent in clear text. Note that for this idea to work properly the encryption scheme used should provide *key privacy* [69].

It seems intuitively that external anonymity cannot be obtained for both principals in a two-party protocol, since one party must reveal its identity first. However, this is not always the case as shown by the idea of *secret handshake* protocols [51] where principals only authenticate each other if they both possess matching credentials. This, however, seems to require a special set-up phase before the protocol starts to set up mutually trusted groups.

Internal Anonymity

A protocol achieves *internal anonymity* if an adversary actively participating in the protocol is unable to determine the identity of at least one of the protocol principals. Protocols of this type are required as part of anonymous communications services

such as the Tor anonymity network [255]. Each client needs to set up a secure session key with multiple communications servers; the servers have known public keys, but the client wishes to remain anonymous. Note that in this scenario the server may be the adversary attempting to learn the identity of the client.

Since the client need not even have a long-term key in this scenario it may seem as first that any protocol will work. However, there are some important security properties that must hold. Firstly, the protocol must still provide confidentiality, so the session key must be shared only with the correct server. Secondly, it must not be possible for the adversary to distinguish between protocol runs given any two different clients.

Goldberg *et al.* [308] designed a protocol called ntor, shown as Protocol 1.15, which provides internal anonymity. Only principal B, the server, has a public key. The principals exchange ephemeral Diffie-Hellman values, but compute two shared elements which are combined as a shared secret, $\mathbf{Z}$, which is used as to derive the session key.

Shared information: public key of B, y_B.
Information known to B: private key of x_B, with $y_B = g^{x_B}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{r_A}, y_B^{r_A}$	$\xleftarrow{t_B}$	$\mathbf{Z} = t_A^{r_B}, t_A^{x_B}$

Protocol 1.15: Protocol ntor of Goldberg, Stebila and Ustaoglu [308]

Goldberg *et al.* [308] provided a formal analysis of Protocol 1.15 and showed that it provides anonymity as well as a session key secure against outsiders. They gave a formal definition of one-way anonymity which, roughly speaking, says that an adversary cannot distinguish which of any two chosen clients took part in a protocol run. The ntor protocol is now deployed in the Tor software.

A number of variants to the ntor protocol have been proposed. Backes *et al.* [47] proposed a protocol called Ace which is slightly more efficient than Protocol 1.15. Ghosh and Kate [303] proposed a one-way anonymous protocol based on the learning with errors (LWE) problem and designed to be secure against adversaries equipped wth quantum computing.

1.5.12 Protocol Efficiency

It is usually desirable to make protocols as efficient as possible. At the same time, it is not unusual that stronger security goes along with lower efficiency. Since all

protocols involve multiple parties, we may need to consider the overall efficiency as well as the efficiency with regard to individual parties. There are usually two main concerns with regard to efficiency: *computational efficiency* and *communications efficiency*. We should also not forget *storage efficiency*: how big are the required long-term keys and the working memory required during a protocol run? This can be relevant for devices with limited storage capacity.

Computational efficiency is concerned with the computations that the protocol principals need to engage in to complete the protocol. The amount of computation required in cryptographic protocols will depend on the algorithms used to provide the cryptographic services, such as encryption and decryption functions, generation and verification of digital signatures and message authentication codes, and calculation of hash functions. These can vary considerably between specific algorithms and, in particular, computations required for public key algorithms are usually much greater than those for symmetric algorithms. Furthermore, there has been, and continues to be, considerable research into improvements of implementations of different fundamental cryptographic operations, such as multiplication in finite fields. Advances in implementations can make a significant difference to the relative merits of different cryptographic algorithms. It is not always easy to compare the computational efficiency of protocols when they use different cryptographic settings, but often protocols have the same or a similar setting and a simple count of the number of arithmetic operations can be useful.

Communications efficiency is concerned with the number and length of messages that need to be sent and received during the protocol. We will sometimes use the term *flow* to describe a single set of message components which can all be sent together. As well as minimising the size and number of message flows, it can be important to have as few *rounds* as possible in the protocol. Gong [316] gives the following definition.

Definition 25. *One protocol* round *includes all messages that can be sent in parallel from any point in the protocol.*

Many protocols can be run in a smaller or larger number of rounds by grouping messages together in different ways. Minimising the number of protocol rounds is not necessarily the most efficient choice. In Chap. 5 we will describe many key agreement protocols with two message flows which can be run together in one round. As explained in Sect. 5.4.13, such protocols can be extended to provide key confirmation by adding two message components to be sent in each direction. This naturally results in protocols which can be run in two rounds with two message flows in each round. However, the same protocols can also be run in three rounds (one more round), each of one message flow, so three flows in total (one less flow). This works by piggybacking the first key confirmation message onto the second protocol flow (see Protocol 5.22). Which way to run the protocol depends on whether it is better to minimise the number of rounds or the number of flows. Chapter 9 discusses group key establishment where the number of rounds can be large (for example, proportional to the size of the group). For such protocols, reducing the number of rounds can significantly improve the delay in completing a protocol run.

1.6 Tools for Verification of Protocols

The sheer complexity of behaviour that security protocols can exhibit makes their verification no small task. It has long been recognised that informal arguments about protocol correctness are not reliable. Therefore, much research has been devoted to ways of gaining assurance that the security goals are satisfied using formal mathematical methods. It is beyond the scope of this book to cover such techniques in detail, but we believe it is worthwhile to give a flavour of how they work. We aim to make practitioners aware of what it means to conduct an analysis or obtain a proof with these approaches. In this section we focus on the *tools* available for protocol analysis, including a brief overview of some of the methods.

Formal methods for analysing security protocols are often divided into two major categories.

Symbolic methods treat the cryptographic primitives in a protocol as symbols which can be manipulated deterministically using specified rules. For example, a ciphertext could be modelled as a symbol which can be used as an input to a decryption algorithm. If the correct decryption key is used then the corresponding plaintext symbol is returned, but otherwise nothing can be gained from the ciphertext. An adversary can only use the defined interfaces to compute with symbols. One way to differentiate different symbolic methods is by the cryptographic primitives which they support. For example, many tools today support Diffie–Hellman operations by symbolically defining the relevant computations.

Computational models allows the adversary to compute whatever can be efficiently computed with the known inputs. For example, a ciphertext could be a bit string output by a probabilistic algorithm which can be used to help an adversary to compute something about the plaintext. Whether the computations of the adversary are useful depends on what the defined security goals are.

Chapter 2 is devoted to examining computational models for authentication and key exchange, which have been extensively developed in the cryptographic research community since 1993. We will therefore not describe them further in this section. We focus more on symbolic models here. For these there exist some automatic tools which can be applied without a deep knowledge of the theory behind them.

Symbolic methods are often said to use the *Dolev–Yao model*. This is in recognition of a pioneering paper by Dolev and Yao [256] which has a simple model in which the adversary mainly uses the principals as oracles to try to decrypt things which it could not do otherwise. Only public key encryption is involved in the original model. Although there is no limit on the number of times a public key can be applied, and therefore infinitely many cases to consider, Dolev and Yao provide theorems which allow certain designs to be proven secure. No tools or automatic checking was involved in their work. Later, symbolic methods are often said to use the Dolev–Yao model even though they may typically include many cryptographic primitives other than public key encryption and consider much more complex security properties than secrecy of specific values.

The general idea behind tools using symbolic models is to show that bad states, representing successful attacks, cannot be reached. Usually we do not wish to explicitly bound the scope of an adversary in terms of how many runs of the protocol it can observe since that could exclude certain attacks. Indeed, Millen [554] showed that there exist protocols which require an unbounded number of parallel sessions in order to find a certain attack. However, since no tool can use infinite resources all tools have some limitations on what they can achieve. For example, some tools do not guarantee to terminate, while others always terminate but perhaps only with a partial result that no attacks exist up to a certain horizon.

In the following subsections we highlight the applicability and main features of a range of tools supporting security analysis. We focus on tools which are currently available at the time of writing. This means that we do not cover historically important tools such as Murϕ [561] and AVISPA [38]. Some useful sources of further information are the surveys of Basin *et al.* [60] and Meadows [540].

1.6.1 FDR

Lowe [501] developed a method for verifying security protocols using FDR, a model checker for the process algebra CSP [636]. This method was used to find a previously unknown attack on the Needham–Schroeder public key protocol (see Sect. 4.3.3). A comprehensive introduction to the method, including background on CSP, is contained in the book of Ryan and Schneider [638]. Although it has been around for quite a long time, FDR has been regularly updated and continues to be used for security protocol analysis.

Each principal taking part in the protocol is modelled as a CSP process representing the protocol steps performed by the principal. In CSP, communication is modelled by the notion of *channels*. In Lowe's formulation of the Needham–Schroeder public key protocol the following channels are defined:

- *comm*, which carries messages sent by honest principals;
- *fake*, which carries messages introduced by the intruder;
- *intercept*, which carries messages that are killed by the intruder.

In CSP, a communication is an event of the form $c.v$ where c is the name of the channel on which the communication takes place and v is the value of the message that passes along the channel. Message components are put together using a dot. For example, message 1 of the Needham–Schroeder public key protocol is expressed by the event *comm.Msg1.a.b.Encrypt.kb.na.a*. Here the letters a and b stand for variables denoting principal identities; *na* stands for a variable denoting a nonce; and *kb* stands for a variable denoting a public key. The simplest way of constructing processes in CSP is by prefixing. A process that performs an action x and then behaves like process P is denoted $x \rightarrow P$ (pronounced 'x then P'). The $\rightarrow$ operator is right associative, so $x \rightarrow y \rightarrow P = x \rightarrow (y \rightarrow P)$. The initiator in the Needham–Schroeder public key protocol is defined by the CSP process $INITIATOR(a, na)$ below.

$$
\begin{aligned}
&INITIATOR(a,na) = \\
&\quad user.a?b \rightarrow I\_running.a.b \rightarrow \\
&\quad comm!Msg1.a.b.Encrypt.kb.na.a \rightarrow \\
&\quad comm.Msg2.b.a.Encrypt.ka?na'.nb \rightarrow \\
&\quad \text{if } na = na' \\
&\quad \text{then } comm!Msg3.a.b.Encrypt.kb.nb \rightarrow \\
&\quad I\_commit.a.b \rightarrow session.a.b \rightarrow Skip \\
&\quad \text{else} STOP
\end{aligned}
$$

The question marks model inputting of data. The initiator waits for the value for b from the channel *user* and then sends message 1. The event $I\_running.a.b$ indicates that a is taking part in a protocol run with b. The initiator then waits for a corresponding message 2, decrypts this message and checks that the value for na' matches the same value sent in message 1 (the initiator will accept any value for nb). If the nonce matches, then the initiator sends message 3 and commits to the session. The event $I\_commit.a.b$ represents the fact that the initiator is committing to a session with b; the event $session.a.b$ represents the fact that a carries out a session with b; and *Skip* represents a process that completes its task. If the nonce does not match then the initiator halts; this is represented by the CSP process *STOP*.

A CSP process $RESPONDER(b,nb)$ that captures the steps performed by the responder is defined similarly.

The intruder is modelled using the CSP choice operator $\Box$. The choice operator can be applied to any number of processes. The resulting process can choose to act like any one of the processes. For the Needham–Schroeder public key protocol the intruder is modelled with the choice operator applied to 12 processes: three for overhearing the messages sent by the honest principals, three for intercepting messages by the honest principals, three for replaying overheard messages, and three for generating messages using the known nonces and injecting them. The intruder is defined by the process $INTRUDER(m1s,m2s,m3s,ns)$, where the sets $m1s$, $m2s$, $m3s$ collect the undecrypted messages 1, 2 and 3 the intruder has overheard so far and ns is the set of nonces the intruder has learnt. The part of the intruder process involving message 1 only is as follows; the parts involving messages 2 and 3 are similar.

$$
\begin{aligned}
&INTRUDER(m1s,m2s,m3s,ns) = \\
&\quad comm.Msg1?a.b.Encrypt.k.n.a' \rightarrow \\
&\quad \text{if } k = Ki \text{ then } I(m1s,m2s,m3s,ns \cup \{n\}) \\
&\quad \text{else } I(m1s \cup \{Encrypt.k.n.a'\},m2s,m3s,ns) \\
&\quad \Box\ intercept.Msg1?a.b.Encrypt.k.n.a' \rightarrow \\
&\quad \text{if } k = Ki \text{ then } I(m1s,m2s,m3s,ns \cup \{n\}) \\
&\quad \Box\ fake.Msg1?a.b?m : m1s \rightarrow I(m1s,m2s,m3s,ns) \\
&\quad \Box\ fake.Msg1?a.b!Encrypt?k?n : ns?a' \rightarrow I(m1s,m2s,m3s,ns)
\end{aligned}
$$

To analyse the protocol, we need to restrict the CSP specification of the protocol and intruder so that the resulting system is finite. For example, Lowe's CSP model of the Needham–Schroeder public key protocol is limited to a single initiator A, a single responder B, and a single intruder. Despite this restriction, Lowe found an attack on this protocol. This finite CSP model of the protocol, represented by the process *SYSTEM*, is compared against other CSP processes that capture the desired security properties. Lowe formalised two authentication properties as the following CSP processes:

$$AI = I\_running.A.B \rightarrow R\_commit.A.B \rightarrow AI$$
$$AR = R\_running.A.B \rightarrow I\_commit.A.B \rightarrow AR$$

The first process, AI, has the behaviour that the responder B commits to a session with initiator A only if A took part in the protocol run. The second process, AR, has the behaviour that the initiator A commits to a session with responder B only if B took part in the protocol run.

To check whether a given property holds for a protocol, one tests for refinement between the CSP processes representing the protocol and the property in question. In CSP, a process P refines process Q if every trace of P is also a trace of Q. Intuitively, a trace is a sequence of events. The first authentication property amounts to checking that *SYSTEM* refines AI; the second property amounts to checking that *SYSTEM* refines AR. The FDR tool is used for testing these refinements. It turns out that the refinement check involving AR succeeds, while the one involving AI does not. FDR produces a trace in the latter case, which shows that B commits to a session with A, although A never attempted to interact with B. This trace is the famous attack published by Lowe.

Lowe proposed a correction to the protocol to prevent the attack found on the original protocol. No attack on the corrected protocol was found using the model checking technique. However, since the model checking is carried out on a small system running the protocol, the question remains whether there is an attack on an arbitrary system (with an arbitrary number of initiators and responders). To answer this question, Lowe offered a proof that any attack on the general protocol implies an attack on the smaller protocol. The proof is by hand and runs to several pages. Since no attack is found on the smaller system, he concludes that there can be no attack on an arbitrarily sized system.

Lowe [503] wrote a program called CASPER to make his model checking technique accessible to protocol designers and implementers lacking specialist skills in CSP. CASPER automatically produces a CSP model of a protocol as well as the intruder using a more abstract description of the protocol, similar to the conventional description. The CASPER input language is machine-readable and provides special syntax to make explicit certain aspects of protocols. For example, the first message of the Needham–Schroeder public key protocol is represented as:

```
<pkb := PK(B)>
1. A -> B: {na, A}{pkb}
```

The first line represents that principal *A* uses a function *PK* to look up *B*'s public key. As well as the definition of the sequence of messages passed between the principals, CASPER requires the definition of the actual system to be checked. To define the actual system, one has to instantiate the parameters appearing in the protocol definition to actual values. Consider a system with a single initiator, Alice, and a single responder, Bob, each of whom has a public key, a secret key and a nonce. This is defined by writing the following lines within the ♯ system heading in the CASPER input file.

```
INITIATOR(Alice, Na, SKa)
RESPONDER(Bob, Nb, SKb)
```

The size of the system to be model-checked can be changed very easily.

CASPER provides a concise notation for specifying a range of security properties, including a number of authentication properties described in another paper by Lowe [504].

1.6.2 NRL Analyzer and Maude-NPA

As part of a long-term project on cryptographic protocol analysis, the US Naval Research Laboratory (NRL) developed a special-purpose software tool known as the Analyzer [535]. It was one of the first tools developed for this purpose and was improved over many years to provide increased automatic support for the user. The original tool is a Prolog program with several thousand lines of code.

The Analyzer is a hybrid that possesses features of both a model checker and a theorem prover. Searching begins from an insecure state, looking backwards to see if that state can be reached from the initial state. If so then an explicit attack has been found. Lemmas may be proven to show that infinite classes of states are unreachable. These may lead to a proof that all paths to the insecure state start in unreachable states.

The specification of a protocol consists of several elements such as the following.

System state, which includes the values known by the adversary and the protocol principals, as well as event sequences that have occurred.

Protocol rules, which state how honest principals behave and what is learnt by the adversary after each protocol step. The adversary may encrypt or decrypt with known keys and concatenate known values. It is assumed that the adversary is able to recognise which key was used to encrypt any ciphertext.

Rewrite rules, which define the cryptographic properties, such as that encryption and decryption are inverse operations.

Protocol goals are defined by the insecure states. As well as the values known to the adversary (such as keys) the state can include a notion of local variables (such as a value that some principal believes is the key) or conditions on sequences of events. In common with other tools, cryptographic properties are defined implicitly by what values may be found by the adversary using known keys. However, some extensions

take into account the specific properties of certain algorithms, in particular cipher-block chaining for block ciphers [700].

The Analyzer was used to analyse a large number of protocols and to reproduce many known faults and find new ones. Initially the subjects were restricted to relatively simple key establishment protocols, but the tool was also used to analyse complex protocols such as the (IKE) protocol [536] (see Chap. 5 for a description of IKE).

A successor to the original Analyzer, rewritten in the logic Maude, was developed and released as Maude-NPA in 2007 [271]. (NPA stands for NRL Protocol Analyzer.) Version 3 of Maude-NPA became available in 2017. Maude-NPA puts the tool on a more formal foundation; in particular, if the analysis terminates without finding an attack then this constitutes a proof that no attack exists in the model. The new tool retains many of the features of the Analyzer, including the backward searching technique. Maude-NPA has been used to analyse a variety of cryptographic protocols incorporating broader concepts such as homomorphic encryption [270] and security API protocols [321, 322].

1.6.3 ProVerif

ProVerif is an open source tool, first developed around 2003 by a team led by Bruno Blanchet. A detailed description, a comparison with other tools and a survey of applications were given by Blanchet [115].

Inputs to ProVerif consist of protocol specfications using an adaptation of the pi calculus together with a choice of the security property to be analysed. The output can be one of three possibilities: confirmation of the property, an attack showing that the property is false, or a 'false attack' which means that the tool has given up. It is also possible that ProVerif will not terminate at all, because in general the problem of determining security with an unbounded number of sessions is undecidable [115]. ProVerif is designed to be an automatic tool so that it will run without further intervention from the user.

The tool is extensible by users through definition of equations for additional cryptographic properties. For example, this allows coverage of properties such as Diffie–Hellman [464] and bilinear pairings [599].

Proverif has been applied to a wide range of protocols, both by the tool designers and by others. These include protocols in different application areas such as electronic voting, RFID, and trusted platform modules. Example analyses related to authentication and key establishment have been provided for JFK [3], Kerberos [116] and Diffie–Hellman-based protocols such as draft TLS 1.3 [93].

1.6.4 Scyther and Tamarin

Scyther, scyther-proof and Tamarin are three open source tools developed by researchers at ETH and Oxford.[2] Although Scyther is an older tool than Tamarin, support for use of Scyther is still widely available at the time of writing. In particular,

[2] https://people.cispa.io/cas.cremers/tools/index.html.

tutorial materials are available for Scyther, making it a good choice for learning about protocol security. Scyther also features a graphical interface to help users understand how a protocol works and to illustrate attacks.

Scyther was first made available around 2008 and advertised as a 'push-button tool' [231] for analysis and verification of security protocols. Scyther has been applied to analysis of many of the protocols which are presented later in this book including HMQV, KEA+, NAXOS and JKL. In an extended study, Scyther was used to analyse all of the protocols in the ISO/IEC 11770 standard [228, 229] and those in the first four parts of the ISO/IEC 9798 standard [61, 64] (see Sections 3.2.3, 3.3.4, 3.4.4 and 5.6 for details of these protocols). Scyther identified a number of problems with both standards and was also used to demonstrate the absence of problems in revised versions.

The original Scyther tool is suited to finding typical attacks in symbolic models. While it provides an unbounded search, the absence of an attack does not formally constitute a proof of security. Later a version of Scyther known as Scyther-proof [541] was developed which can output a proof in a logic known as Isabelle/HOL. Scyther-proof was used to provide proofs to repaired versions of protocols in the ISO/IEC 9798 standard [61, 64].

In 2010, Basin and Cremers [62, 63] designed extensions to Scyther to model compromise of parties by the adversary. This *compromising adveraries* version of Scyther allows attacks such as weak forward secrecy and key compromise impersonation to be captured, which at the time was not possible with other symbolic tools. Although this extension does not allow protocols to be proven secure in computational models, it does allow *separation* of computational models by finding symbolic attacks on some protocols which shows that they cannot be secure in the corresponding computational model. Basin and Cremers used their analysis to provide hierarchies of protocols such that certain protocols can be shown to be better than others (in a partial order) at avoiding attacks from certain types of adversaries.

The Tamarin prover [542] is a successor to Scyther. Tamarin allows for more faithful representation of cryptographic models and has built-in support for Diffie–Hellman and bilinear pairings. It provides a security proof in the symbolic model as long as the analysis terminates without an attack. Although it is not guaranteed to terminate, experience shows that on well-known protocol examples it typically does so within a few seconds [656]. Unlike Scyther, Tamarin allows user extensions which gives much more flexibility to the user. Like Scyther, Tamarin has been applied to many well-known protocols such as KEA+, NAXOS, UM, JKL, STS-MAC [656] and draft TLS 1.3 [230].

Dreier *et al.* [259] reported on extensions to the scope of Tamarin which previously had been unable to deal with some cryptographic primitives such as blind signatures. By overcoming these limitations they were able to provide Tamarin analyses for protocols in applications such as digital cash and e-voting.

1.6.5 Tools for Computational Models

This section has so far looked at tools using symbolic methods based on the model of Dolev and Yao. In the cryptographic research community the modern standard for proofs of protocols (and primitives) is a reductionist *computational* proof which shows a protocol to be secure as long as some computational assumption holds. We look in some detail at computational security models for key establishment protocols in Chap. 2. In some restricted cases proof in the Dolev–Yao model can imply a computational proof [48], but in general this is not the case.

Reductionist proofs have, up to today, normally been performed by hand, without the help of tools, in the same way as traditional proofs in mathematics. When the model and proof are simple this has been a satisfactory situation, but increasingly problems have arisen due to complexity when more advanced security properties are considered or where complex protocols are involved. One particular problem is that real world protocols tend to be much more complex than those traditionally studied in the research literature. Therefore there has significant interest to provide machine support for computational proofs [56].

As far back as 2005, Halevi [340] proposed a way to develop a tool to remove much of the routine checking that is typically necessary in reductionist proofs and based on the well-established technique of game-hopping [675]. Today there are tools available which aim to achieve exactly this task, although in our (limited) experience current tools are not easy to use for the non-expert. They usually require interaction with the user and their output can be difficult to interpret. We mention here two prominent examples.

EasyCrypt [58] was developed by researchers in Spain (at IMDEA) and France (at INRIA) and originates from 2009. At the time of writing the tool is still under development. EasyCrypt has been applied to provide proofs for several cryptographic primitives and protocols. In particular it was used to verify one part of a proof of the TLS handshake protocol [99] in an implementation known as miTLS. EasyCrypt has also been used [57] to provide verified proofs of some well-known key agreement protocols such as NAXOS (Protocol 5.15). Interestingly, the proof for NAXOS [57] allowed reduction to a more standard computational problem than that used in the original handwritten proof.

Cryptoverif [113, 114] was developed by Blanchet, originally around 2005, but at the time of writing is still under development. The tool is specifically based on the technique of using a sequence of games. It has been used to provide computational proofs for many protocols which we examine later in this book, including the Needham–Schroeder shared key protocol [113], Kerberos [116], SSH [173] and draft TLS 1.3 [93].

It seems likely that in the future tools to support computational proofs will be developed much further and may even replace the handwritten proofs in common use today. This would certainly help to make such proofs more reliable and perhaps more widely used.

1.6.6 Comparison of Tools

In this section we have briefly introduced some of the most prominent tools available for formal analysis of authentication and key establishment protocols. Although there are a number of related ideas, each of these tools has both strengths and weaknesses. Our brief survey has certainly not been exhaustive and research in this area continues to be active. Table 1.5 summarises some of the properties of several of the analysis tools.

Table 1.5: Summary of some tools used for protocol analysis

Properties → ↓ *Tool*	Type	Usage
FDR	Symbolic	Automatic
Maude-NPA	Symbolic	User-guided
ProVerif	Symbolic	Automatic
Scyther	Symbolic	Automatic
Tamarin	Symbolic	Automatic/user-guided
EasyCrypt	Computational	User-guided
CryptoVerif	Computational	User-guided

All of the tools in Table 1.5 have been mentioned in previous subsections. The properties in Table 1.5 do not include any measure of how useful each tool is in providing assurance about protocol security. This is very difficult to compare, especially when tools use different techniques. However, roughly we can divide the tools into those that are automatic and those which require user intervention. It seems reasonable to suggest that an automatic tool should be used in the early protocol design stage to filter out any simple errors while the more complex tools, which generally provide proofs of some sort, should be used at a later stage.

Cremers *et al.* [236] performed a comparison of four tools: AVISPA, ProVerif, Scyther and FDR. They examined two main features.

1. How much of the state space was explored in each tool. They noted that different tools often explore quite different states.
2. How efficiently each tool performed its analysis. They concluded that ProVerif was the fastest of the four, with Scyther coming a close second.

Cremers *et al.* noted that there have been very few studies comparing different tools, a situation which does not seem to have changed much recently. This does not make selection of the right tool easy for the non-expert.

In 2003, Meadows [538] discussed various promising research directions which the formal methods community could follow to extend existing approaches to protocol security analysis. Twelve years later [540] she followed up on this with a new

paper where she examined how far these directions had developed, concluding that the area is still vibrant with many exciting new directions.

1.7 Conclusion

This chapter has introduced important background concepts needed to understand the many protocols examined later in the book. Seeing the large variety of protocol goals, the many different cryptographic primitives that can be used, and the different kinds of adversaries, may help to explain why there exist so many different protocols for authentication and key establishment. Later in the book we will often refer to the concepts we have introduced here.

The focus of this chapter has been on providing informal, intuitive understanding of the concepts. However, to obtain high assurance of security a formal approach is needed. The final main section of this chapter looked briefly at different tools available for protocol analysis. This remains an active research area and we cannot say that there is a universal method to ensure that any chosen protocol has no flaws, particularly if we demand an automatic tool. The next chapter looks in more detail at formal models designed by cryptographers for use with computational proofs.

2

Computational Security Models

2.1 Introduction

During the early years of open academic research in cryptography it was commonplace to see research papers following a sequence of break, fix, break, fix ...: a scheme would be proposed and then others would analyse it, often finding an attack. The scheme was then patched up and subjected to further scrutiny, and so the cycle would continue. Although this pattern applied to many different kinds of cryptographic schemes, it was nowhere more true than for protocols for authentication and key exchange.

Starting in the 1980s research began into providing more formal approaches to protocol specification and analysis. While a formal security definition allows the possibility of a mathematical proof of security, no less important is the ability to give a specific and unambiguous definition of what it means to be secure. Even without a proof, a formal definition allows designers to specify what a protocol is intended to achieve so that claimed attacks can be judged by whether or not they violate what the protocol is actually intended to achieve.

Initial models came from outside the cryptographic research community and treated any cryptographic primitives as a black box [538]. Such approaches have been summarised in Sect. 1.6. This chapter is about the computational approach favoured in the cryptography research community, often called *provable security* or *reductionist security*.

The provable security approach is used in the cryptographic research community to prove security of a variety of cryptographic schemes. Such proofs are complexity-theoretic reductions; the security of the subject algorithm or protocol **S** is related to the security of another better understood problem **P** in the sense that if there is an efficient algorithm that can break **S** then there is an efficient algorithm to solve **P**. Note that we usually have no guarantee that **P** really is hard to solve; for example, **P** might be the integer factorisation problem, whose absolute difficulty is not known.

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_2

2.1.1 The Significance of a Computational Proof of Security

It is important to understand the limitations of a formal computational model and of any proof within such a model. A security proof is always relative to the model used. Any model may or may not capture what was intended and models often simplify real-world complexities. A security proof says nothing about attacks which are not captured within the model used.

We examined many different potential security goals for key establishment and authentication in Sect. 1.5. Different computational security models aim to capture different subsets of these goals. Indeed, with the exception of protocol efficiency, all of the protocol goals discussed in Sect. 1.5 have been defined in some computational models. Since there often are different interpretations of the informal meaning of such goals, it is perhaps not surprising that there are also varying formal definitions. Thus even with a formal definition it remains impossible to say unequivocally that a protocol achieves a particular goal, but it is at least possible to say whether a goal is reached as defined in the chosen model.

The notion of provable security has received some criticism, notably by Koblitz and Menezes [438]. Some potential limitations are the following.

- Current provable security techniques do not help in protocol design. A small change in the protocol will require a new proof to be constructed.
- Choosing the right model, and obtaining a correct proof within the model, is often difficult. Suboptimal solutions can be adopted because proofs for better solutions cannot be found.
- Security proofs tend to be difficult to understand for the average practitioner. They typically run to several pages of mathematical reasoning and there are few people who check such proofs in any detail. Sometimes proofs have turned out to be wrong [211], whether or not the protocol is secure in the model.

Even if you agree with these points, they do not imply that a proof of security is not worthwhile. On the contrary, our view is that a security proof is a very valuable attribute of any cryptographic protocol. Formal security models allow us to unambiguously decide whether or not a protocol meets its security design goals. A proof within such a model undoubtedly increases confidence that security errors in the design have been avoided. At the same time, a security proof is a theoretical object which, in the case of real-world systems, should be supplemented by informal scrutiny, machine analysis, and any other approaches available to gain assurance.

Many complexity-theoretic proofs for protocols rely on the so-called *random oracle model* [76]. The random oracle model assumes that a hash function works in an idealised manner: it returns truly random values each time it is used, except that it always returns the same output when the input is the same. No such function exists in reality, but a random oracle seems to be representative of the way that we want hash functions to work. The research community has differing views on the reasonableness of the random oracle model, but everyone agrees that it is better if a proof does not rely on it.

2.1.2 Elements of Computational Models

The driving force behind any computational security model is the adversary. While the adversary represents our concept of an attacker with a will to break the protocol, formally it is simply an algorithm. The adversary's abilities are constrained only by its computational power: typically we require that the adversary is reasonably efficient but otherwise the strategy it uses is unconstrained.

The adversary is given access to a number of *oracles* (or *queries*) which allow it to control all the messages sent to protocol principals and often give the adversary access to various secret information. For example, insider attacks are modelled by allowing the adversary to corrupt principals and obtain their stored values. Also the adversary is usually given access to session keys from sessions other than the one it is targeting. Cryptographic algorithms may be modelled either with generic properties (for example, an encryption algorithm secure against chosen plaintext attacks) or as specific transformations (for example, a Diffie–Hellman operation in a particular group).

Security of protocols is usually defined in terms of a *security game* played by the adversary in combination with its environment. In Fig. 2.1 the environment is denoted as a *challenger* with the job of presenting to the adversary **A** any elements required by the model. This may include public keys and parameters at the start of the game and may require the challenger to compute random values. The adversary's queries must be consistently answered as they would be if the adversary were running against a real implementation of the protocol. The challenger is responsible for answering the queries from the adversary and is often said to *simulate* the environment.

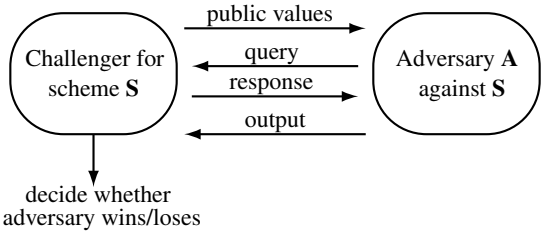


Fig. 2.1: Security game for scheme **S**

The game ends when the adversary halts its computations and gives its output. There must be a formally specified condition, applied at the end of the game, to decide whether or not the adversary has won the security game. Security is then defined based on limiting the success of the adversary in winning the game. For some given scheme **S**, we would like to ensure that no efficient adversary **A** is able to win the security game with a probability which is non-negligible in the length of the long-term keys. Often we need to settle for a bit less than this.

Once a security model is in place, it is possible to try to achieve a proof of security by a reduction to an existing scheme **T** as shown in Fig. 2.2. To achieve this, the proof constructs a new adversary against **T** which uses the adversary **A** rather like a sub-routine in a program. If **A** is successful, the proof tries to use this success to solve the given instance of **T**. Here scheme **T** could be a simple problem (like the computational Diffie–Hellman problem or factoring) or it could be a full security property of another scheme such as one of those defined in Sect. 1.3. We may, for example, achieve a proof that a protocol (scheme **S**) is secure if an encryption scheme (scheme **T**) which it uses is secure.

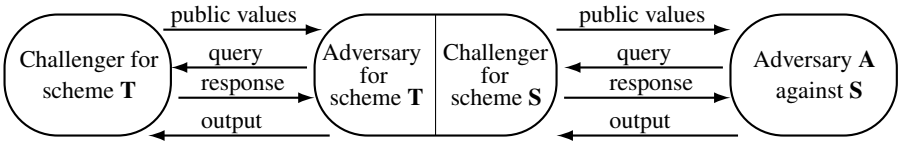


Fig. 2.2: Reduction from scheme **S** to scheme **T**

We can summarise the main elements in a provable security analysis as follows:

- the security definition, including the specification of the adversary's capabilities and a winning condition;
- the specification of the protocol to be analysed; and
- a theorem and its proof bounding the probability that an adversary can win the security game against the protocol.

The security goals captured in a particular protocol security model vary, and can include entity authentication, session key confidentiality, or both. Models vary according to what queries are allowed to the adversary as well as the winning condition of the security game. Moreover, terminology and notation can vary greatly between models, even when the basic ideas are the same. Table 2.1 summarizes some common concepts present in most computational models.

Table 2.1: Common terminology in computational models for key exchange

Party	A protocol principal possessing a long-term key and which usually takes part in multiple runs of the protocol
Instance	The actions of a specific protocol run at some party
Partnering	Rules to specify when the model regards two instances as being linked
Freshness	Rules to specify whether a session is a valid target for the adversary
Key derivation	Application of a key derivation function (KDF) to obtain a session key from a shared secret

One important concept in all of the models is that of an *instance*, often called a *session*. In informal descriptions, a session is commonly regarded as a protocol run executed between two parties, but in computational models a session is almost always located at a *single* party. Instead of thinking about parties sharing a session we rather think of parties who may have matching sessions which share the same session key. Usually an instance shares state (specifically a long-term key) with other instances at the same party. Instances also have a local state which is updated as the protocol is run. Different models can have very different notation for sessions, even when the concept is the same, which can be a source of confusion. We return to this issue in Sect. 2.5.

Another tricky notion is that of a *partner* of an instance. The idea is to identify which instances can be expected to share the same session key (or state related to the session key). Models usually do not allow the adversary to win against a chosen instance if the session key from a partner of that instance has been obtained. Therefore it is very important to precisely define which instances can be partners. Different models use different ways to define partners. We will mention two common ways later in this chapter: *matching conversations* between instances and *session identifiers*.

In the remainder of this chapter we take a mainly historical perspective on the development of security models. Generally speaking, older models are simpler than more modern models so this can help ease the exposition. In addition, there is no model which encompasses all other models and it is not uncommon to see older approaches used in recent papers. The next three sections cover three major families of models: the Bellare–Rogaway (BR) family, the Canetti–Krawczyk (CK) family, and the extended CK (eCK) family. Section 2.5 compares some of the main features of different models and discusses which models are stronger than others. Most of the chapter deals with models for two-party protocols with a single long-term key per party, but Sect. 2.7 looks at models which go beyond this boundary. Finally in Sect. 2.8 we look at models for defining secure channels established using key exchange protocols, which use a weaker security definition than most of the models.

Conceptually, the goal of secure key exchange is for the principals to complete the protocol in possession of a session key which is random from the viewpoint of the adversary. Consequently, most models for key exchange follow the *key indistinguishability* approach in which the goal of the adversary is to distinguish the agreed session key from a random string. This is by far the most common approach in the literature. An alternative is the *simulation-based* approach in which the goal of the adversary is to distinguish between interactions with a real system and interactions with an ideal system, with the latter being secure by definition. The most prominent of the simulation-based approaches is the *universal composability* (UC) framework [181]. An earlier related approach is Shoup's model discussed in Sect. 2.6. Models for secure channels, explored in Sect. 2.8, cannot achieve indistinguishability of session keys because they allow the adversary to observe key usage on the channel. Therefore they use models related to authenticated encryption where the adversary's goal is to distinguish ciphertexts or to forge valid messages.

2.2 Bellare–Rogaway Model

Bellare and Rogaway pioneered the study of authentication and key establishment protocols using computational cryptographic models. Their paper on entity authentication and key distribution from 1993 [75] is one of the earliest papers in the area of *practice-oriented provable security* which they established; it was presented even before their famous paper on the random oracle model. Many of the elements which they introduced are present in most of the models which have been proposed subsequently. In addition there are many refinements and extensions to their original ideas which were developed by themselves and others.

In this section we start by describing the 1993 model and then follow its developments during the 1990s through Bellare and Rogaway's own work in 1995, the extensions to public-key based protocols by Blake-Wilson and Menezes, and then the password-based model of Bellare, Pointcheval and Rogaway of 2000.

2.2.1 BR93: The First Computational Model

By 1993 there were already formal methods tools to analyse security with the Dolev–Yao approach and the BAN logic had recently become popular. Within the cryptography research community efforts had started to understand how to design robust protocols and classify attack types. Bellare and Rogaway specifically acknowledge the papers of Diffie *et al.* [253] and Bird *et al.* [103] as logical precursors to their work.

Communication Model

Consider a set of identifiers which represent protocol principals. Each principal has a long-term secret key and an internal state which updates as the protocol runs.

A protocol is a function Π which represents the specification of the protocol by stating what the output of a specific principal will be if it is given a specific input message in a specific protocol state. More precisely, Π is a sequence of *transitions*, each with five inputs and three outputs as shown in Table 2.2.

Table 2.2: Protocol inputs and outputs in the BR93 model

	Inputs
i	the identity of the sender of the current message;
j	the identity of the intended recipient of the current message;
a	the long-term secret of the sender;
κ	the transcript of the current protocol run for the current session.
	Outputs
m	the output message from the current transition (which can be empty);
δ	the output decision which can have values in {accept, reject, none};
α	the updated local state of the sender of the message.

While the abstract definition of the protocol allows for multiple principals to be engaged in multiple runs of the protocol, we can also consider a single run, or a *session* at a particular party. We call this an *instance* of the party and denote an instance by $\Pi^s_{i,j}$, where i is the party at which the session takes place, j is the intended partner of i, and s is an index that is unique for a particular pair (i, j). We say that an instance *accepts* if and when its output decision becomes "accept".

Matching Conversations

In all the computational models, a critical issue is how to identify partners of protocol instances. To achieve authentication instances should only accept when the protocol is run with their intended partners. For key establishment the adversary should also be prevented from getting (too much) information about the session key of an instance also from its partner. For example, although the adversary is allowed to obtain session keys from independent sessions, this should not include the partner session.

BR93 uses the notion of *matching conversations* to define partners. The idea is that the communicating entities should agree on the messages that have been sent and received in the protocol run they are involved in. Of course the entity sending the last message in the protocol cannot know if that message was ever received (during the protocol in question) and so we need to relax the requirement for the party sending the last message.

To capture what this means we have to specify what a conversation is. Atomic events in the BR93 model are transitions at one instance where a message is received, the state is updated, and a message is sent. From the external view all that can be seen is a message sent and received at a certain time as shown in Fig. 2.3. A conversation is therefore a finite sequence of events at one instance $\Pi^s_{i,j}$ where each event is of the form (τ, r, m) for a time τ, an incoming message r and an outgoing message m.

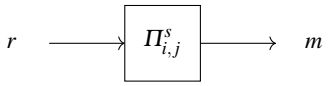


Fig. 2.3: Atomic event (τ, r, m) in the BR93 model

In the BR93 model, and in all other similar models we look at in this chapter, the adversary is free to choose the incoming message r in any way. Indeed the only way that instances communicate in the model is through the adversary. The adversary is free to simply relay messages between instances so that the conversations proceed exactly as they would in a real run of the protocol without the adversary being present. Such an adversary is known as a *benign adversary*. But equally the adversary can choose to fabricate messages, using any efficient algorithm, to change, delay or reorder messages, or simply to delete them.

Another matter which we have to take care of is that the first message in a protocol occurs with no received message to cause it. We use the BR93 notation by writing

(τ, λ, m) for the event at an instance which initiates the protocol run – here λ denotes the empty string, τ is the time of the event and m is the first protocol message. We call the instance where this event occurs the *initiator*. Similarly, when the final protocol message is received it causes no output message in response. Again we use the BR93 notation (τ, r, λ) for the event at an instance which ends the protocol run by receiving message r at time τ with an empty output. We call the instance where this event occurs the *terminator*.

Consider any two conversations at instances $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, so that one instance is at party i intending to communicate with party j, while the other is at party j intending to communicate with party i. Depending on the protocol specification, the instance which is the initiator (sends the first message) may, or may not, also be the terminator (receives the last message). Suppose that protocol Π has R messages in total. If R is odd, say $R = 2\rho - 1$, then the initiator is not the terminator; moreover the initiator sends ρ messages and receives $\rho - 1$ messages while the terminator sends $\rho - 1$ messages and receives ρ messages. If $R = 2\rho$ is even then the initiator *is* the terminator; moreover both the initiator and the terminator send and receive ρ messages.

Case 1: R Odd

To be concrete we look first at the case where $R = 2\rho - 1$ is odd. Since exactly one instance can be the initiator we shall assume that $\Pi^s_{i,j}$ is the initiator which means that $\Pi^t_{j,i}$ is the terminator. Then we will denote the conversation of $\Pi^s_{i,j}$ as a sequence of atomic events,

$$(t_0, r_0 = \lambda, m_0), (t_1, r_1, m_1), \ldots, (t_{\rho-1}, r_{\rho-1}, m_{\rho-1})$$

and the conversation of $\Pi^t_{j,i}$ as another sequence of atomic events,

$$(u_0, s_0, n_0), (u_1, s_1, n_1), \ldots, (u_{\rho-1}, s_{\rho-1}, n_{\rho-1} = \lambda).$$

In order for these two conversations to be matching we will require that $m_0 = s_0$, $n_0 = r_1$ and so on. However, at the end of the matched messages we have a slight difference depending on which party we are looking at: only when matching to the terminating party do we include the last message in the conversation. This is because in the protocol we must allow the non-terminating party to accept once it has sent its last message even though the adversary can trivially delete this last message. This leads to the following definition.

Definition 26. *For the case of an odd number of protocol messages, we say that the conversation at* $\Pi^s_{i,j}$ matches *the conversation at* $\Pi^t_{j,i}$ *when:*

1. $t_0 < u_0 < t_1 < u_1 < \ldots < u_\rho$
2. $m_0 = s_0, n_0 = r_1, \ldots, m_{\rho-1} = s_{\rho-1}$.

Similarly the conversation at $\Pi^t_{j,i}$ matches *the conversation at* $\Pi^s_{i,j}$ *when:*

1. $t_0 < u_0 < t_1 < u_1 < \ldots < u_\rho$
2. $m_0 = s_0, n_0 = r_1, \ldots, n_{\rho-2} = r_{\rho-1}$.

Case 2: R Even

We now look at the case $R = 2\rho$ is even. Again we shall assume that $\Pi^s_{i,j}$ is the initiator so that the first event at $\Pi^s_{i,j}$ has $r_0 = \lambda$. Since R is even this means that $\Pi^s_{i,j}$ is also the terminator so that the last event at $\Pi^s_{i,j}$ has $m_\rho = \lambda$.

Then $\Pi^s_{i,j}$ has conversation,

$$(t_0, r_0 = \lambda, m_0), (t_1, r_1, m_1), \ldots, (t_\rho, r_\rho, m_\rho = \lambda)$$

and $\Pi^t_{j,i}$ has conversation,

$$(u_0, s_0, n_0), (u_1, s_1, n_1), \ldots, (u_{\rho-1}, s_{\rho-1}, n_{\rho-1}).$$

Now we can adjust the definition of matching based on the different conversation formats. Again, only when matching to the terminating party do we include the last message in the conversation.

Definition 27. *For the case of an even number of protocol messages, we say that the conversation at* $\Pi^s_{i,j}$ matches *the conversation at* $\Pi^t_{j,i}$ *when:*

1. $t_0 < u_0 < t_1 < u_1 < \ldots < u_{\rho-1} < t_\rho$
2. $m_0 = s_0, n_0 = r_1, \ldots, m_{\rho-1} = s_{\rho-1}$.

Similarly the conversation at $\Pi^t_{j,i}$ matches *the conversation at* $\Pi^s_{i,j}$ *when:*

1. $t_0 < u_0 < t_1 < u_1 < \ldots < u_{\rho-1} < t_\rho$
2. $m_0 = s_0, n_0 = r_1, \ldots, n_{\rho-1} = r_\rho$.

Mutual Authentication

We are now ready to discuss the first of the two security properties analysed in the BR93 paper. This is *mutual authentication*, which informally means that both parties should gain assurance that they are in conversation with each other. Many protocols for key establishment do not require, and do not provide, this property but in the BR93 model it is a requirement for secure key establishment.

In the security game for mutual authentication the adversary interacts with the instances by sending a message r of its choosing at time τ, and receiving a response m, as depicted in Fig. 2.3. Since this is the only interaction that the adversary is allowed to perform, there may not seem any need to give it a specific name. In later models this interaction became known as a send query to distinguish it from other adversarial queries. In addition to the message r chosen by the adversary, the inputs to such a query must include (i, j, s) values to identify the specific instance to receive the query. The output of the query is the output message m specified by the protocol definition.

There are two requirements in the security definition for mutual authentication. The first is that instances will accept when they have engaged in matching conversations. This is a correctness notion since is it how we expect the protocol to work in the

absence of a malicious adversary. Remember that in the BR93 model the adversary actually transports messages even if only in a benign way. The second requirement says that if an instance accepts then there must have been another instance with a matching conversation.

Definition 28. *A protocol Π is a* secure mutual authentication protocol *in the BR model if, for any efficient adversary* **A***, both of the following hold.*

1. *If the conversations at $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ match each other then both instances will end in the accept state.*
2. *The probability that there exists any instance $\Pi^s_{i,j}$ in the accept state but without an instance $\Pi^t_{j,i}$ which has a matching conversation is negligible.*

Bellare and Rogaway went on to provide examples of protocols satisfying Definition 28.

Key Establishment

The second security property defined in the BR93 model is key establishment, or *authenticated key exchange*. In the BR93 model secure key establishment is only defined for protocols which provide mutual authentication. This results in a significant limitation in the set of protocols which can be proven secure in the strict BR93 model. Subsequent developments from the BR93 model, which we explore later in this chapter, do not have such a restriction.

In a successful run of a key establishment protocol, instances will eventually have a session key as part of their local states (α as defined in Table 2.2). Since one of the main reasons to have session keys is to keep sessions independent, the adversary should be allowed to obtain session keys for some sessions while the protocol still remains secure for the remaining sessions. In order to capture this property, in the BR93 model the adversary is allowed to obtain the session keys from any instances that it chooses. To this end, an adversary query called reveal is introduced with inputs (i, j, s) to identify the specific instance to receive the query, and output equal to the session key. The adversary is allowed to ask reveal queries only to instances which have output an accept decision. (In the BR93 formalism this is ensured by assuming that the private state of each instance is empty until the instance has accepted.)

We do not require the adversary to output the session key of its targeted session in order to win its security game – we require only the much weaker condition that the adversary can reliably distinguish the session key from a random string of the same length. This is in line with the principle that we should make the security game as easy for the adversary as we can without allowing the adversary to win trivially. Note that an adversary which can reliably predict, say, the least significant bit of the session key can reliably win the security game. The requirement is formalised through the test query.

At some point the adversary must decide to issue a test query whose inputs (i, j, s) specify the instance to receive the query. This instance is often known as the *target session*. Only one test query is allowed during the whole game. To answer

the test query we imagine a challenger who flips a fair coin to define a bit b; if $b = 0$ then the response is the session key at that instance, but if $b = 1$ then the response is a completely random string of the same length as a session key. The adversary's final output is a bit b' which is its own guess at the value of b. Then we say that the adversary wins the security game if and only if $b' = b$.

There is one important restriction which we have ignored so far. We allowed the adversary to obtain any session key by asking reveal queries but this will trivially allow the adversary to win by revealing the session key of the target session or its partner. Therefore we must restrict the adversary access to the reveal queries by introducing the notion of *freshness*. The adversary will only be allowed to ask the test query to a fresh session. In particular, this prevents the adversary from asking a reveal query to the target session and any matching session.

Definition 29. *We say that an instance $\Pi^s_{i,j}$ in the BR93 model is* fresh *if:*

1. *$\Pi^s_{i,j}$ has accepted;*
2. *$\Pi^s_{i,j}$ has not been asked a* reveal *query;*
3. *if $\Pi^t_{j,i}$ has a matching conversation with $\Pi^s_{i,j}$ then $\Pi^t_{j,i}$ has not been asked a* reveal *query.*

We summarise the whole BR93 security game in Table 2.3. As outlined in Sect. 2.1.2 we can regard this game as being played with a challenger which is responsible for choosing the long-term keys of the parties and choosing the random bit b (Step 1). Since the game is a guessing game, the adversary can always win with probability 1/2 by simply outputting a random guess for b. Therefore we are only interested in how much better the adversary can do than guessing so we define the adversary's *advantage* as $\Pr(b' = b) - \frac{1}{2}$. We are now able to give the BR93 security definition for authenticated key exchange.

Table 2.3: Key exchange security game in the BR93 model

1. Long-term keys are generated for all parties. In the BR93 paper only symmetric keys are used and each party starts with a symmetric key shared with every other party. A random secret bit b is chosen by the challenger (this can happen at any time before step 3).
2. The adversary can interleave in any chosen way the following actions:
   - send messages to any instance (including starting new instances by sending the empty string) and receive the correct response;
   - ask reveal queries to any instance to obtain the session key.
3. The adversary asks a test query to any fresh instance as defined in Definition 29.
4. The adversary can continue to send messages and ask reveal queries as long as the tested session remains fresh.
5. Eventually the adversary outputs its guess bit b'.

Definition 30. *A protocol Π is a* secure authenticated key exchange protocol *in the BR93 model if it is a secure mutual authentication protocol and both of the following hold.*

1. *For any benign adversary* **A** *who relays conversations between $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, both instances will end in the accept state with the same session key.*
2. *The advantage of any efficient adversary* **A** *in the security game is negligible.*

Timing of test Query

In contrast to what is shown in Table 2.3, the original BR93 model required the test query to be the adversary's final query in the game. (This also applies to the original BR95 model discussed in Sect. 2.2.2.) Although this may seem a natural and correct assumption, it is not necessary and there is no reason to prevent the adversary from continuing with other queries as long as they do not trivially allow the adversary to win the game. At the least, allowing the adversary to make extra queries cannot make the adversary any weaker and this is always desirable if we can still obtain a security proof. Moreover, we can think of protocols, albeit somewhat contrived, that would be secure if the adversary is restricted to make test its final query, but insecure otherwise.

Consider a protocol, otherwise secure, in which instances will accept further queries after accepting the key, and if that query includes the accepted session key the instance will return a special flag. Such a protocol, in other words, allows the adversary to test guesses of the session key within the protocol. Such a protocol is insecure when we allow the adversary to continue querying after the test session since the adversary can simply test whether the key it received in response to the test query receives the special flag. As later reported by Bellare and Rogaway [74] they were alerted to this issue in a private communication by Petrank in 1995 and thereafter the adversary's power was increased to allow further queries after the test session. Canetti and Krawczyk [178, Appendix A] also discussed this issue.

Limitations of the BR93 Model

The BR93 paper was a pioneering publication and the basic ideas are still in wide use today. With hindsight we can see a few limitations of the original model which later models have sought to overcome. We summarise these here before going on to look at other models.

- The cryptographic setting in BR93 is very simple. Each party already shares a long-lived key with every other party. This is not realistic in many distributed communication settings. Furthermore, public keys play a crucial role in modern cryptography and they are not considered at all.
- There is a strong coupling between authentication and key exchange. While many protocols do provide both services, many others do not and these cannot be properly analysed in the BR93 setting. Even if both are desired, it can be useful to be able to analyse key exchange independently of authentication.

- There are limited adversarial capabilities. The adversary in BR93 is not allowed to obtain long-term keys of principals and therefore the model does not capture malicious insiders which are common in practice.

Such issues may not be important within the limited cryptographic setting of BR93, but we will see later that they can be important when we want to capture security properties of more diverse protocols.

2.2.2 BR95: Server-Based Protocols

Two years after their first model was published, Bellare and Rogaway [78] presented a new model which we will refer to as the BR95 model. It is aimed at a different situation: key establishment using an online server. Bellare and Rogaway called this the *three-party case*. We look at protocols in this category in Chap. 3. The basic ideas of the model they used for this purpose are the same as in the BR93 model, but there were also some changes. Perhaps the most prominent change is the de-coupling of key establishment and entity authentication. It is no longer necessary for a secure key exchange protocol to be one which also provides mutual authentication. Indeed, the protocol which they prove secure in their paper does not provide entity authentication – no party gets any assurance that its intended partner has taken part in the protocol at all when the protocol is completed.

Without entity authentication there is a need for a different mechanism to define partnering in the BR95 model. Some way of defining partners is always required since the partner of the test query cannot be subject to a reveal query otherwise the adversary can always win the security game. The BR95 model introduced the idea of a *partner function* which takes as inputs (i, j, s) specifying the instance, $\Pi^s_{i,j}$, to receive the query and a transcript of the protocol run so far. The output of the partner function is a value t which specifies the partner instance $\Pi^t_{j,i}$. No specific partner function is defined for any protocol, but protocol security is defined by requiring the *existence* of a partner function which ensures that the adversary's advantage in winning the security game is negligible. The effect of this is to change the definition of freshness in the BR95 model; in comparison with Definition 29 for the BR93 model, the third condition must now require that the partner of $\Pi^s_{i,j}$, as defined by the partner function, has not been asked a reveal query.

An explicit send query is defined in the BR95 model to allow the adversary to interact with honest parties running the protocol. (To be precise, there are two such queries in the BR95 paper, one for messages to clients and one for messages to the server. Here we conflate the two as is done in later models.) The send query has an instance and a protocol message (possible empty) as input and provides the adversary with the output which would be computed by the honest party running the protocol (see also Table 2.4 later). The adversary can freely choose the input message and instance.

In addition to the reveal query available in the BR93 model, the BR95 model has a corrupt query which allows the adversary to obtain the long-term key of any party. Long-term keys are shared with the server, but the server itself cannot be corrupted. All instances at parties which have been corrupted through this query are no

longer fresh, so they cannot be used as the target of the test query by the adversary. The corrupt query also has an input which allows the adversary to specify a new long-term private key for the corrupted party. This does not seem an unreasonable expectation for a real-life adversary who corrupts a party.

Later Shoup and Rubin [676] adapted the BR95 model in order to model server-based key exchange with the aid of smart cards for the user machines. The idea is that the long-term key will never leave the smart card and can only be accessed through a defined interface. The difference in the model is that the adversary is able to additionally obtain any of the state information in a non-target session except what is stored in the smart card. In addition the adversary is able to query the smart card interface at any time. Finally smart cards themselves can also be corrupted to obtain the long-term key, but then sessions using that card can no longer be fresh. Shoup and Rubin designed an efficient protocol secure in this extended model.

Limitations of the BR95 Model

Some of the innovations of the BR95 model have stood the test of time; most models today include some kind of corrupt query as well as explicit queries for send and reveal. However, the introduction of the partner function has proved to be problematic. Choo *et al.* [209] demonstrated that the partner function used in the BR95 paper is incorrect – with the function specified, the adversary is easily able to win the game because instances which might be expected to be partners are not according to the definition. However, they found an alternative partner function which does not have the same problem. This illustrates a potential problem with the partner function idea – different partner functions are possible for the same protocol and the right function to use is not necessarily obvious from the protocol. Later Rogaway [634] discussed the limitations of the partner function definition as an example of the difficulty of getting security definitions correct. Free choice of the partner function is not normally allowed in more recent models, but the model rather defines the partner function, for example through matching conversations.

2.2.3 The Public Key Setting: The BWM and BWJM Models

In a natural progression, Blake-Wilson and Menezes [108] in 1997 extended the BR93 model to the public key setting. We refer to this as the BWM model. Instead of each pair of users initially sharing long-term symmetric keys as in BR93, the BWM model assumes that each user has a public–private key pair for signing messages and a public–private key pair for encryption. The main ideas of the model are largely the same as in the BR93 model. In particular mutual authentication is a requirement for secure key exchange. As in BR95, a corrupt query is available to the adversary allowing recovery and replacement of long-term private keys, in addition to a reveal query to obtain accepted session keys.

The BWM model uses matching conversations to define mutual authentication as in BR93. However, Blake-Wilson and Menezes noticed that the standard security definition for digital signatures causes a slight problem. The adversary may be able

to alter each signature output by an instance to form new valid signatures, something not ruled out by the standard security definition for signatures. This means that the peer instance will accept using the altered signature even though matching conversations have not occured, violating the second requirement of Definition 28. Therefore a slightly altered version of the matching conversations definition is provided which allows malleable tags to be added to the messages which match. Blake-Wilson and Menezes point out an alternative solution to this problem, which is to strengthen the security definition of signatures to prevent malleability. Signatures secure in this definition later became known as *strongly unforgeable signatures* [32].

In the same year, Blake-Wilson, Johnson and Menezes [107] extended the BWM model to consider key agreement. We call this the BWJM model. They differentiate between authenticated key agreement (which they abbreviate as AK) and authenticated key agreement with key confirmation (which they abbreviate as AKC). The AKC definition follows closely the BR93 model by requiring a secure AKC protocol to already provide mutual authentication as in Definition 28. This means that a secure AKC protocol must have at least three flows. Two examples, with proofs, are provided by Blake-Wilson *et al.* using Diffie–Hellman constructions and secure message authentication codes.

Blake-Wilson *et al.* [107] also considered how to decouple key establishment from mutual authentication to provide AK, still using the notion of matching conversations to define partnering. However, the only protocol considered uses a weakened model in which reveal queries are forbidden to the adversary. Although these authors suggest that 'no key agreed in an AK protocol should be used without key confirmation', later results by Jeong, Katz and Lee [397] and by Kudla and Paterson [459] show that one-round protocols without any key confirmation (or mutual authentication) can provide security in models allowing reveal queries.

2.2.4 BPR00: Forward Secrecy and Passwords

In 2000, Bellare and Rogaway, this time together with Pointcheval, proposed the latest refinement of their key exchange model [74] which we refer to as the BPR00 model. The model incorporates a number of enhancements, notably allowing capture of the forward secrecy property. However, the main motivation of the BPR00 model was to include analysis of password-based protocols (see Chap. 8).

Recall that the security game for key establishment, originally defined in the BR93 model, requires the adversary to distinguish the session key from a random string with non-negligible probability. Security in such a game is essentially unachievable for a password-based protocol where users have low-entropy passwords. This is because the adversary will have a non-negligible probability of correctly guessing a user's password which will always allow the adversary to win the security game. Therefore the security definition in the password-based setting has to be relaxed by requiring a successful adversary to win with an advantage that is significantly better than the probability of simply guessing the password correctly. Moreover, the adversary can always use a send query to test the correctness of a password guess and therefore it is necessary to limit the number of active attack messages

while at the same time allowing a large number of passive eavesdropping events. In order to accommodate this, a new execute query was made available to the adversary. In a good password-based protocol, the adversary's advantage can be kept small by severely limiting the number of send queries; this corresponds to the real-world technique of locking out a user after a small number of password failures.

Partnering in the BPR00 model is captured in a new way, different from the use of matching conversations in BR93 and partner functions in BR95. The BPR00 model introduces the notion of *session identifiers* (SIDs), which have been used in many later models. The exact form of session identifiers is not specified but should uniquely define a session between two parties – it is suggested that the concatenation of protocol messages can be used to define the SID. BPR00 also uses the notion of *partner identifiers* (PIDs) at each instance to specify the identity of the *principal* which that instance intends to communicate with. When an instance accepts a session key it must have also decided on a SID and PID. In the BPR00 model partners must have the same SID and session key, and record each other's principal as the PID.

Another innovation in the BPR00 model is the consideration of forward secrecy. This is captured by modifying the definition of freshness. If forward secrecy is required then corrupt queries only affect the freshness of an instance if it occurs *before* the test query is issued. The corrupt query in the BPR00 model has two variants: one which returns the state of the party it is sent to, including any randomness but not the session key, and one which only returns the long-term key. The first kind of corrupt query was used in the BR95 model. The second kind allows BPR00 to formally capture the property of *weak forward secrecy* which forbids the adversary from being active in the test session.

Table 2.4 summarises the adversarial queries available in the BPR00 model. This table applies also to the BR93 and BR95 models for the subset of queries that are defined in those models.

Table 2.4: Queries in the BPR00 model

Query	Inputs	Outputs	Purpose
send	Instance + input message	Output message	Active attacks
execute	Instance pair	Protocol transcript	Passive attacks
reveal	Instance	Accepted session key	Compromise of session keys
corrupt	Principal	Long-term key	Compromise of long-term keys
test	Fresh instance	Session key or random string	Adversary challenge on indistinguishability

Limitations of the BPR00 Model

One curious feature of the BPR00 model is that the definition of session freshness does not differentiate between which parties have been corrupted. Thus, when modelling a protocol which does not provide forward secrecy, all sessions are no longer

fresh if *any* party is corrupted and consequently the adversary cannot ask the test query to any valid party. This means that corruption is disallowed completely which seems a quite unnecessary restriction. Choo *et al.* [208] showed that this limits the ability of the BPR00 model from capturing certain realistic attacks. However, this does not seem to be a fundamental limitation of the model, and can be fixed by making the freshness definition more precise by only limiting corruption with regard to the target session and its partner.

2.2.5 Summarising the BR Model Variants

Table 2.5 summarises the development of the Bellare–Rogaway models from 1993 to 2000. Here we can see how the models evolved to capture different kinds of protocols and security properties. For the earlier models with only symmetric key cryptography the forward secrecy property is not applicable (marked N/A' in the table).

Table 2.5: Comparison of the main Bellare–Rogaway model versions

Model	Setting	Partnering mechanism	Forward secrecy
BR93	Two-party shared key	Mutual authentication	N/A
BR95	Server-based	Partner function	N/A
BWM	Public key	Mutual authentication	No
BWJM	Key agreement	Matching conversations	No
BPR00	Password-based	Session identifiers	Yes

2.3 Canetti–Krawczyk Model

In 1998 a different direction in formal modelling of key exchange was started by Bellare, Canetti and Krawczyk [72]. Initially the idea was to take a very general approach incorporating not just key exchange but also authentication of exchanged data. Another feature of the new approach was an attractive ability to design protocols in a modular way. In 2001, Canetti and Krawczyk [178] made fundamental changes to the model, avoiding some shortcomings of the 1998 security definition, but retaining the modular features. This model is ofter referred to as the CK01 model. In 2005, Krawczyk made the model much more specific and concrete in order to analyze the HMQV protocol [453]. While these three stages have significant differences, and the HMQV model is finally quite close to the BR model, there is a logical development between them which we follow in this section.

2.3.1 BCK98 Model

A central concept introduced in the BCK98 model is the *authenticated links model*, abbreviated as AM. In the AM the adversary **A** has restricted capabilities compared

with the usual key exchange models. In particular **A** is not able to fabricate messages but can only activate instances using messages output by legitimate parties. To differentiate the usual kind of stronger model, the latter model is called the *unauthenticated links model*, and abbreviated as UM.

The main purpose of introducing the AM is that secure protocols can generally be much simpler while at the same time there is a general method to promote a protocol which is secure in the AM to one that is secure in the UM. This is achieved by applying a transformation, or *compiler*, whose input is a protocol π secure in the AM and whose output is a new protocol π' secure in the UM. Such a transformation is called an *authenticator*. Secure protocols in the AM can be defined independently of authenticators but then combined with any authenticator to obtain secure protocols in the UM.

The BCK98 model follows an approach to defining security of key exchange which is quite different from the BR indistinguishability approach – their definition is instead *simulation-based*. This means that they consider an ideal secure key exchange protocol and compare this with the real protocol π. Roughly speaking, if π is secure then for any efficient adversary running π there is an efficient adversary against the ideal protocol such that the output of the real and ideal systems are indistinguishable.

Problems in the BCK98 Model

It turns out that the simulation-based security definition in BCK98 is too strong. The result is that the basic protocols which were originally claimed to be secure in the model do not in fact satisfy the definition. (This is acknowledged in the CK01 paper [178].) The main issue, as discussed by Shoup [674], is that the corruption allowed in BCK98 is too strong. The BCK98 model only considers corruption in which the adversary obtains session keys as well as long-term keys and can choose which parties to corrupt at any time: Shoup calls this *strong adaptive corruption*. It turns out that in this situation some basic protocols, such as Diffie–Hellman key exchange, are not secure in the AM. This is unsatisfactory since Diffie–Hellman is a basic building block against which we have no realistic attacks in the AM.

To fix these problems it is natural to make a weaker security definition. Shoup [674] also used a simulation-based approach but allows for weaker version of corruption so that security can still be obtained for protocols which we believe should be secure. Later Canetti and Krawczyk considered definitions in the universal composability (UC) model [181] which can be considered to be a generalisation of the security definition of BCK98. However, the CK01 model takes a pragmatic approach by using an indistinguishability definition, like that of the BR models, while retaining the AM and UM and the authenticators to map between them.

2.3.2 CK01 Model

The CK01 model [178] contains more than just a model for defining security of key exchange protocols. There are two other significant contributions, which together

were highlighted by Canetti and Krawczyk as the main motivation for their work. The first of these is an extension of the modular approach to design of secure protocols already started in the BCK98 model as mentioned above. The second is a model for secure channels and an analysis of how these can be achieved from a secure key exchange model. In this section we focus on the model and its differences from the BR models but we will also say something about the modular approach. We defer a discussion of secure channels in the CK01 model until Sect. 2.8.

Adversary Capabilities in CK01

The adversary in the CK01 model works in the same basic manner as in the BR models. Specifically it controls the communications between all parties. The notation used in CK01 is rather different from that in the BR models and in particular there is no separate notation for instances. Instead sessions can be identified as instances and are named using *session identifiers* which are distinct at any party. More specifically, a session can be identified by a tuple (P_i, P_j, s) for a principal P_i intending to communicate with a principal P_j using a session identifier s. Note that the session identifier s has a completely different role from the instance number s in the BR notation. This is discussed further in Sect. 2.5 and in Table 2.8.

As well as scheduling message interactions with parties (sessions) and observing the response, the adversary has available some specific queries which go beyond what is in the BR models.

Party corruption. The corrupt query allows the adversary to obtain the long-term key of a party exactly as in the BR95 and BPR00 models. The query returns the long-term key of the party and also all the memory which may include ephemeral keys or session keys.

Session key reveal. Just as in the BR models, the adversary can obtain the session key of any completed session by asking a reveal query.

Session state reveal. Except in corrupt queries, the BR model does not allow the adversary to obtain information which may be stored during (or after) the session key computation. The CK01 sessionstate query can be asked of an incomplete session and receives the internal state in return. The model allows the protocol to specify what is included in the session state; a typical example would be an ephemeral Diffie–Hellman exponent.

Session expire. CK01 models forward secrecy by allowing the adversary to expire sessions. The effect of this query is to delete the session key from the session specified as input to the query. This means that party corruption at the target session can occur after the session has expired without trivially giving away the session key.

Security in the CK01 Model

The definition of security in the CK01 model follows very closely the BR definitions we have seen in previous sections, but with appropriate adjustments to the new

queries. In particular the definition is based on the absence of an efficient adversary who can distinguish the session key in a fresh session from a random string in the session key space. In order to highlight the similarity we continue to use the same terminology as in the BR model to describe the security game, instead of changing to the terminology used in the CK01 paper. Thus Table 2.6 describing the security game and the security definition in Definition 32 can be observed to be similar to Table 2.3 and Definition 30.

Table 2.6: Security game in the CK01 model

1. Long-term keys are generated for all parties using a protocol-dependent function called initialization which is run before the protocol starts.
2. The adversary can interleave in any chosen way the following actions.
   a) The adversary invokes a new protocol instance with a specified partner principal, a session identifier s and a role. There must be no other session between the same parties with the same value of s.
   b) The adversary sends a message to any session from a specified principal and observes the response. The response can include an indication that the session is *complete* in which case all memory is erased except for the session key.
   c) The adversary can ask reveal queries to any instance to obtain the session key.
   d) The adversary can ask sessionstate queries to any incomplete instance to obtain the session state.
   e) The adversary can ask corrupt queries to any instance to obtain the state of the principal.
   f) The adversary can ask expire-session queries to any completed instance which erases the session key from the session state.
3. At some point the adversary asks a test query to a fresh instance. The adversary can continue to send messages and ask other queries in step 2, as long as the tested session remains fresh.
4. Eventually the adversary outputs its guess bit b'.

Partnering in the CK01 model is defined through session identifiers. More specifically, any party P_i starts a protocol run when it receives an input of the form $(P_i, P_j, s, \mathsf{role})$ for a session identifier s and $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$.

Definition 31. *Two sessions* $(P_i, P_j, s, \mathsf{role})$ *and* $(P_j, P_i, s', \mathsf{role}')$ *in the CK01 model are said to be* matching *if* $s = s'$ *and* $\mathsf{role}' \neq \mathsf{role}$. *A session* $(P_i, P_j, s, \mathsf{role})$ *is* fresh *as long as:*

- *it has not been asked a* sessionstate *query;*
- *it has not been asked a* reveal *query;*
- *if* P_i *was asked a* corrupt *query then the session was first asked an* expire-session *query;*
- *the above three conditions also hold for any matching session* $(P_j, P_i, s, \mathsf{role}')$.

The third condition captures forward secrecy by allowing the adversary to ask a corrupt query to the owner of the test session, or the owner of a matching session, as long as the session has been expired. A variant of the definition without forward secrecy prevents any corrupt query to the test session or its partner.

Definition 32. *A protocol Π is a* secure authenticated key exchange protocol *in the CK01 model if both of the following hold:*

1. *if two uncorrupted parties complete matching sessions then both instances will end in the accept state with the same session key;*
2. *the advantage of any efficient adversary* **A** *in guessing the correct bit in the security game is negligible.*

It is interesting to compare this definition with the earlier BR model definition (specifically Definition 30). The second part of both definitions deals with indistinguishability of the target session key from random and is essentially the same in both cases. However, the first parts are a little different. The BR93 definition says that parties running the protocol with a benign adversary will always accept with the same session key. This is only a functional requirement which does not seem to have any security implications. However, the first part in the CK definition is rather different and *does* have security implications. Indeed we will see later that some natural ways of designing protocols can end up with matching sessions which do not have the same session key.

Modular Design

The CK01 model inherited the idea of an ideal AM world and a real UM world as defined in the BCK98 model outlined in Sect. 2.3.1. Indeed, Definition 32 specifies security in the UM and a similar definition applies in the AM where the only difference is that the adversary is restricted from fabricating any messages.

A key result of the CK01 paper is that a protocol π which is secure in the AM can be transformed into a protocol $\mathscr{C}(\pi)$ which is secure in the UM by applying a valid authenticator $\mathscr{C}$. Furthermore, a method of building valid authenticators from basic authenticators is defined that can be applied to single messages. This allows protocols to be designed which automatically inherit a security proof by combining a basic protocol secure in the AM with a secure authenticator.

The CK01 paper defined a few building blocks which can be used with the modular method. Two basic protocols secure in the UM are basic Diffie–Hellman key exchange and simple key transport using encryption. These can be combined with two valid authenticators from the BCK98 paper, one using signatures and one using CCA-secure encryption. These authenticators transform each protocol message into an interactive pair of messages. As an example, Canetti and Krawczyk showed that signed Diffie–Hellman (Protocol 5.25) can be derived as a combination of the basic Diffie–Hellman protocol and the signature-based authenticator. However, in order to obtain an efficient three-message protocol they had to apply optimisations without formal justification.

Later, Hitchcock *et al.* [359] showed that the kind of optimisations used in the CK01 paper can be formally justified and also showed that it is permissible to 'mix-and-match' authenticators for extra flexibility. They also pointed to a few additional building blocks providing the beginnings of a library for designing secure protocols with a variety of properties. For example, with the four AM-secure protocols and the five valid authenticators they obtained 60 distinct UM-secure protocols.

Despite the attractiveness of this approach it has not seen any significant further development. One reason for this may be that the most efficient protocols known today cannot be split in any obvious way into an authenticator and a more basic protocol. In fact it is impossible to achieve any secure two-message protocol using the known authenticators, since authenticators always add extra messages when applied to a protocol in the AM. Thus any secure UM protocol with only one message per party results in a protocol in the UM with at least three messages. For similar reasons, one-pass protocols (with limited security properties) can also not be reached using authenticators.

Post-specified Peers

Canetti and Krawczyk designed a variant of the CK01 model the following year [179] in which the instances in a protocol run may not be aware of the identity of the peer party in the run until the session is completed. This turns out to be a relatively common situation in practice; an example is where it may be desirable for a mobile device to reveal its identity only to a trusted server once it is confident that the party it is communicating with is indeed the correct server. The model was used by Canetti and Krawczyk to analyse the IKE protocol (see Sect. 5.5.5).

The implications of this change mainly relate to the definition of matching instances in the model. In Definition 31 above it was required that each instance knows the identity of its matching partner and that the partners must agree on the pairing. One way to avoid the problem is to delay partnering by saying that a session can only have a matching session if both have completed. However, this allows the adversary to ask a sessionstate query to incomplete sessions, allowing the adversary to win the security game against protocols which seem naturally secure. Therefore Canetti and Krawczyk revised the definition of matching to allow a completed session to have a matching partner which has not completed. In the post-specified peer model a session is identified by a pair (P_i, s) and the intended partner P_j can be considered as an output of a completed session. Then a session (P_j, s) is the partner of a completed session (P_i, s) as long P_j is the intended partner of (P_i, s) and either

- (P_j, s) is not complete, or
- (P_j, s) is complete with intended partner P_i.

Canetti and Krawczyk pointed out that the post-specified model gives a relaxed definition of security so that a protocol proven secure in the usual (pre-specified) model may not be secure in the post-specified model. Later, Menezes and Ustaoglu [546] applied this observation to show that the HMQV protocol (Protocol 5.14) is not secure in the post-specified setting. They also described a combined

model so that protocols secure in the post-specified setting can be run in either variant and still be secure.

Problems in the CK01 Model

The CK01 model has received some significant criticism. A lot of this can be simply addressed by using techniques that are widely used in more recent models. However, we summarise some of the issues here.

Session identifiers. The usage of session identifiers has turned out to be one of the most controversial aspects of the CK01 model because there is no concrete definition of how they are obtained. Instead, the CK01 paper states that session identifiers are:

> ...chosen by a 'higher layer' protocol that 'calls' the protocol. We require the calling protocol to make sure that the session id's of no two [protocol] sessions in which the party participates are identical. Furthermore, we leave it to the calling protocol to make sure that the two parties that wish to exchange a key will activate matching sessions [178, Section 2.1].

Using examples later in their paper, Canetti and Krawczyk suggest concrete ways that session identifiers may be established, for example by prior agreement or by deriving them from signed messages exchanged during the protocol. However, this still leaves open exactly what is allowed and falls short of a practical generic method. This issue was addressed in the HMQV model.

Session state query. Some authors have criticised the lack of a concrete definition of what constitutes session state. The question of exactly what values should be available to the adversary continues to be an area where new models are developing.

Restrictions on queries. In the CK01 definition of freshness, no session which has had a sessionstate query can be fresh. This rules out some attacks which are captured in other models since the test session cannot have its ephemeral keys revealed. Moreover, since the corrupt query gives away all session state, it cannot be used to model only corruption of the long-term key.

2.3.3 HMQV Model

Analysis of the HMQV protocol (Protocol 5.14) was performed by Krawczyk [453] in an updated version of the CK01 model. This model uses the same basic format as CK01 but addresses many of its criticisms. To differentiate it from the CK01 model we will call it the *HMQV model*. The model is tailored to protocols which exchange Diffie–Hellman protocols in two message passes and combine these with the long-term keys. Many other modern protocols fit this pattern.

As in the CK01 model, partnering is defined using session identifiers, but session identifiers in the HMQV model are defined concretely as 4-tuples. A session at principal A with intended partner B has a session identifier (ID_A, ID_B, Out, In)

where *In* and *Out* are the messages received and sent by the session. Matching is then defined the same way as in the original CK01 model so that completed sessions are matched if and only if their identifiers are of the form (ID_A, ID_B, Out, In) and (ID_B, ID_A, In, Out).

As well as capturing (weak) forward secrecy, the HMQV model is also adapted to capture key compromise impersonation (KCI) attacks. This is achieved by allowing the adversary to obtain the private key of the owner of the test session. In other words, the definition of what it means to be fresh is adapted so that a session is still fresh if the long term key of the owner is compromised.

Krawczyk also made a concrete assumption about what is available to the adversary using the sessionstate query. To be precise he looks at two different situations. At first he assumes that the session state is empty except for the session key which can be obtained with a normal reveal query. Later he looks at an alternative where the sessionstate query gives away the ephemeral secret key chosen for that session. The reason for allowing both variants is that the security proof for the HMQV protocol requires a stronger computational assumption for the case where the sessionstate query gives away the ephemeral secret. This ability to define the contents of the sessionstate query can be regarded as a flexible feature of the CK model.

2.4 eCK Model

By 2005 it was widely accepted that the CK and BR models had adequately captured the main security issues for key exchange. There were some options available for exactly how freshness and adversary queries were defined but these could be seen as nuances which could be matched to specific protocols and computational assumptions. Therefore it was a surprise to many that a new model with a rather different idea was proposed in 2007 by LaMacchia, Lauter and Mityagin [470]. They called their model an extended Canetti–Krawczyk model and it is now widely referred to as the eCK model.

The eCK model tackles directly some of the perceived weaknesses in the CK and BR models. Specific advantages are:

- the adversary can obtain ephemeral secrets which belong to the test session;
- the adversary can obtain the long-term key of the test session and of its partner even before the session is completed.

Due to the above observations it was widely believed that the eCK model was strictly stronger than the CK01 (or HMQV) model, but this is not true since there are other features which the eCK model does not capture.

The general idea behind the eCK model is simple and appealing. Each party in a protocol run has two secrets – a long-term secret and an ephemeral secret, the latter chosen for this particular protocol run. If the two principals are A and B, let us denote their long-term secrets by x_A and x_B, and their ephemeral secrets by r_A and r_B. An adversary who can obtain both of x_A and r_A can compute the session key in the same way as the principal A. Similarly an adversary who obtains x_B and r_B can obtain the

session key. However, a priori there is no reason why an adversary who obtains any of the other pairs of secrets should be able to obtain the session key (or even distinguish it from a random string); the adversary should be able to obtain the long-term keys or the ephemeral keys of both parties, even for the test session. There is one more restriction which must always apply to the adversary – if the adversary is active in the test session then it can choose the ephemeral secret and therefore must be prevented from obtaining the long-term key of the partner of the test session. This leads us to the definition of freshness in the eCK model. First we define some notation for the adversary queries as shown in Table 2.7.

Table 2.7: Queries in the eCK model

Query	Inputs	Outputs	Purpose
send	Parties and message	Protocol message	Control message flow
reveal	Session identifier	Accepted session key	Compromise session keys
ephemeral	Session identifier	Ephemeral key	Leak short-term keys
longterm	Principal	Long-term key	Compromise long-term keys
test	Fresh instance	Session key or random	Adversary challenge

As in the HMQV model, partnering is defined through session identifiers defined from the transcript of the messages exchanged. However, there is a subtle difference which can become important as discussed in Sect. 2.5: sessions are only partners if they agree on which one of them takes the initiator and which takes the responder role. Thus (for one-round protocols) sessions are matched if and only if their identifiers are of the form $(\mathsf{role}, ID_A, ID_B, Out, In)$ and $(\mathsf{role}', ID_B, ID_A, In, Out)$ with $\mathsf{role}' \neq \mathsf{role}$.

A session with identifier sid at party P_i with intended partner P_j is *fresh* as long as:

- the session was not asked a reveal query;
- if a matching session exists with session identifier sid' then:
  - not both of $\mathsf{ephemeral}(sid)$ and $\mathsf{longterm}(P_i)$ queries were asked;
  - not both of $\mathsf{ephemeral}(sid')$ and $\mathsf{longterm}(P_j)$ queries were asked;
- if no partner exists then:
  - not both of $\mathsf{ephemeral}(sid)$ and $\mathsf{longterm}(P_i)$ queries were asked;
  - $\mathsf{longterm}(P_j)$ was not asked.

LaMacchia *et al.* [470] designed a protocol called NAXOS (Protocol 5.15) which can be proven secure in the eCK model. Their idea is to use Diffie–Hellman where the ephemeral exponent r is combined with the long-term secret x using a hash function H. Thus a principal with long-term key x will choose random r for a new session, but sends $g^{H(x,r)}$ instead of sending g^r as we normally expect. The point of this is that an $\mathsf{ephemeral}(sid)$ query then only returns r which does not allow the adversary to learn the Diffie–Hellman exponent actually used. Since the adversary is never allowed to ask for both r and x in a fresh session we can hope that $H(x,r)$ will never

be available to the adversary. This so-called *NAXOS trick* has been used in several subsequent protocols to achieve security in the eCK model.

2.4.1 MU08 Model

Menezes and Ustaoglu [547] adapted the eCK model in order to analyse the Unified Model protocol (Protocol 5.12) in a formal way. They mentioned that their model, which we will refer to as the MU08 model, was 'a weakening of the extended Canetti–Krawczyk (eCK) definition' because it does not capture KCI attacks. This was a deliberate choice, necessary because the UM protocol is insecure against KCI attacks. Therefore in order to obtain a proof it is necessary to weaken the security model. This process of matching the model with the expected security properties of the protocol is a common device, and one reason for the proliferation of security models.

The weakening of security in the MU08 is achieved by preventing the adversary from asking the longterm query to the owner of the target session. However, while the MU08 model is weaker than eCK in this sense, it can also be seen as *stronger* in another sense. This is because it allows both longterm and ephemeral queries to be made to the target session or its partner (when it exists), but *only* when the longterm query occurs after the session at that party expires.

2.4.2 eCK-PFS Model

Motivated by the need to better understand when full forward secrecy (as opposed to weak forward secrecy) can be obtained, Cremers and Feltz [234, 235] defined two variants of the eCK model which both give the adversary greater powers. This difference can be defined in terms of which sessions remain fresh (and so available for the adversary to choose as the test session). In order to describe these models they introduced the notion of an *origin session*.

Definition 33. *A session with identifier sid' is an* origin session *for a completed session with identifier sid if the output messages (one or more) from session sid' equal the input messages (one or more) for session sid.*

Note that if a session sid has a partner session sid', then sid' is an origin session for sid and also sid is an origin session for sid'; this is because partner sessions agree on the messages sent and received. The first extension of Cremers and Feltz for the eCK model, which they called eCK^w, allows the adversary to replay the message from an origin session sid' as well as obtain the long-term key of the partner to the test session, as long as $\mathsf{ephemeral}(sid')$ has not been asked. So here the adversary has a partial ability to be active by replaying messages from other sessions, but does not have the ability to choose new messages (and thereby know the ephemeral secret).

The second extension of Cremers and Feltz is called eCK-PFS and differs from eCK^w only in that the adversary is now allowed to obtain the long-term key of the peer to the test session after the test session is complete, even if there is no origin

session. This is similar to the MU08 model, in that the adversary can be fully active in the test session and obtain the long-term key of the peer to the test session later. A little more formally, a session in the eCK-PFS model with identifier *sid* at party P_i with intended partner P_j is *fresh* as long as:

- the session was not asked a reveal query;
- if an origin session exists with session identifier sid' then:
  - not both of ephemeral(sid) and longterm(P_i) queries were asked;
  - not both of ephemeral(sid') and longterm(P_j) queries were asked;
- if no origin session exists then:
  - not both of ephemeral(sid) and longterm(P_i) queries were asked;
  - longterm(P_j) was not asked until after session *sid* was complete.

2.4.3 seCK Model

Saar *et al.* [653] proposed the seCK model as a strengthened version of the eCK model. The seCK model allows the adversary to obtain *intermediate results*, in particular the exponent of the Diffie–Hellman messages sent. For example, in the case of the NAXOS protocol this would allow the adversary to obtain the exponent $H(x,r)$ where the first message sent is the value $g^{H(x,r)}$. The motivation for this is that some implementations could compute such values in secure memory (such as on a smart card) and export them to main memory from where they may become exposed. Although Saar *et al.* also proposed a protocol which they claimed secure in the seCK model, Yoneyama and Zhao [767] later showed that this protocol is not actually secure in either the seCK or the eCK model. Yoneyama and Zhao also provide evidence that it is difficult to find any protocol which is secure in the seCK model.

2.5 Comparing Computational Models for Key Exchange

As we have seen in this chapter so far, there are many different computational models for key exchange. While they all can be seen as developments from the BR93, there have been a few different directions. It is a natural question to ask whether there is a best model to be used in some sense, for example which model most closely captures reality or which model is strongest. While there has been some significant work in examining such questions, in the current situation we cannot give very strong answers. There are a number of factors which we can use to compare individual models.

What the adversary is allowed to obtain. As models have developed, the adversary has generally been made stronger by giving it more access to secrets. From the initial BR93 model which gave the adversary session keys and long-term keys from non-target session, this has grown to include session state and/or ephemeral keys in target and non-target sessions. Since BR93 it has always been assumed that the adversary obtains transcripts.

What the adversary is allowed to change or choose. Models normally assume an active adversary, but this is not always the case depending on what else the adversary is expected to know. Some early models allow the adversary to actively choose long-term keys. An aspect that has not been widely considered is whether an adversary can actively choose randomness or ephemeral keys. Models in which the adversary is passive (a pervasive wire-tapper) when the protocol is run may still be interesting to explore.

When the adversary is allowed to obtain things. With the introduction of forward secrecy, models had to take into account whether compromise of long-term keys happens before or after the target session is completed. There may be interesting cases where the adversary obtains ephemeral data only at a later time.

How partnering is defined/what freshness means. Partnering can be very subtle. We discuss this more below.

What it means for the adversary to win. Since BR93 most models have demanded indistinguishability of the session key from a random string. Many real-world protocols cannot achieve this (see Sect. 2.8 below) and weaker security notions may be sufficient in many cases.

2.5.1 Comparing the BR and CK Models

Choo *et al.* [208] made a comparison of the three BR model variants (BR93, BR95 and BPR00) and the CK01 model, being four of the main indistinguishability models known at the time of their paper.[1] They compared each of the six pairs of models to see if one was stronger than the other. Here model X is said to be stronger than model Y if a protocol that is secure in model X is always secure in model Y. There are some difficulties in making a direct comparison between these different models.

- The BR93 and BPR00 models consider the goal of mutual authentication while BR95 and CK01 consider only key exchange (with implicit authentication).
- Partnering is defined differently in each of the four models. Particularly, in the BR93 model instances can only be partners if they have matching conversations. This is arguably stronger than necessary and a protocol that is secure in BR93 can be transformed into one that is technically insecure in CK01 simply by adding random fields which are ignored by protocol participants. Since CK01 does not restrict how session IDs are defined the random fields may not affect partnering, but matching conversations are easily violated by an adversary who changes the random fields. Choo *et al.* [208, Section 3.5] use such a trick to show that protocols secure in the CK01 model need not be secure in the BR93 model.

In order to avoid these difficulties, Choo *et al.* apply two conditions in making most of their comparisons.

1. Only the key exchange goal is considered; entity authentication is ignored.

[1] Although the original BR93 model omits the corrupt query and is applied only to shared-key protocols, Choo *et al.* assumed later versions of the model, such as the BWM (see Sect. 2.2.3).

2. The CK01 model uses the concatenation of protocol messages as the session identifier.

Given these two conditions, Choo *et al.* found that the CK01 model is strictly stronger than all three BR model variants. They also showed that BR93 is strictly stronger than both BR95 and BPR00. For the other pairs they found that BR93 and CK01 are incomparable (neither one is stronger than the other). These relationships are summarised visually in Fig. 2.4.

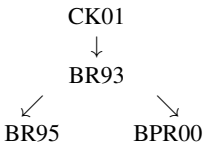


Fig. 2.4: Hierarchy of models according to Choo *et al.* [208] assuming CK01 uses protocol transcripts for session IDs

The main reason that the CK01 model is typically stronger is the addition of the sessionstate query which is absent in all the BR variants. As mentioned in Sect. 2.2.4, BPR00 has a coarse interpretation of the corrupt query, which leads to its potential weakness even though it can capture forward secrecy.

2.5.2 Comparing eCK and Other Models

Cremers [226] performed a careful comparison of the CK01, HMQV and eCK models. He pointed out that they are all strictly incomparable – protocols secure in one of the models can be insecure in the other two.

Specifically, Cremers pointed out that any role-symmetric protocol which includes the identities of the parties in the key derviation function cannot be secure in the CK or HMQV models. The reason for this is that in these models sessions can match even if they disagree on the roles of the parties. (For CK01 there is no requirement for the session identifier to depend on the party identities. In HMQV the roles are explicitly excluded from the session identifier.) When identities are included in the KDF this means that matching session can both accept with *different* session keys thus violating the first requirement of Definition 32. Arguably this is not a serious security issue since the adversary does not again any advantage in knowing the test session key this way. Cremers also pointed out that protocols which do not include the session identifiers (or some other way of differentiating the roles) cannot be secure in the eCK model. Such protocols cannot be matching in the eCK model since partners are required to agree on their respective roles.

By the above method Cremers showed that these three models are incomparable. However, this comparison relies only on the way that matching is defined. This seems not to capture the fundamental differences between the models since matching could have been specified equally in all three models without changing things too much.

(A similar remark could be made regarding the comparisons of Choo *et al.* [208], discussed above, between the CK and BR models.)

Other differences were also discussed by Cremers [226]. Since the CK01 model does not allow sessionstate queries on the test session it cannot be stronger then either eCK or HMQV. Also, neither CK nor HMQV allow the long-term key of the test session and the ephemeral key of its partner to be revealed, as eCK does, so they cannot be stronger than eCK. In the other direction, eCK does not allow full forward secrecy to be modelled since it does not restrict long-term key corruption based on expiration of the test session, so eCK cannot be stronger than CK. In an earlier paper [232], Cremers showed an elegant attack on the NAXOS protocol with a suitable definition of session state. This attack can also be described in HMQV. Since NAXOS has a security proof in the eCK model this is another way to show that eCK cannot be stronger than CK or HMQV. Ustaoglu [719] showed that a similar attack is also possible on the HMQV protocol. This seems a paradox since HMQV protocol has a proof in the HMQV model. However, the HMQV analysis places restrictions on what can be obtained from the sessionstate queries [453, Sections 5.1 and 7 in full version]. Therefore it may be more accurate to say that the attack illustrates that the HMQV model *can* be stronger than the eCK model based on the precise definition of the sessionstate query.

Another way to illustrate the separation between the eCK and HMQV models is consideration of forward secrecy. In the eCK model there is no notion of whether compromise of long-term keys takes place before or after a session is complete. The adversary is free to reveal long-term or ephermal keys at any time as long as the combination of revealed keys is within the allowed set. This means that it is impossible to model forward secrecy in the eCK model since an adversary who knows the long-term key of a party and then is active in that session has as much information as a legitimate party. Therefore the most that can be acheved in the eCK model is weak forward secrecy. Since the CK and HMQV models can capture behaviour where a session is expired and only then is the long-term key revealed, they do not share this restriction. The eCK-PFS model [234, 235], mentioned in Sect. 2.4, extends the eCK model to include consideration of the timing of the long-term key compromise and therefore captures forward secrecy.

2.5.3 Sessions and Session Identifiers

The notion of sessions, or instances, is an important concept in all the computational models for key exchange. Unfortunately the notation and terminology used have been highly inconsistent, even when the intent has been identical. In this chapter we have chosen to keep the notation as used in the original papers in the hope that this will make reference back to those papers more meaningful. Here we discuss how those notations are related.

Some models use the terms *instance* and *session* interchangeably, some use just one of these terms, and some separate the notation of instance and session (for example an instance may contain a session identifier as one of its state variables).

There are two main approaches to denoting sessions. The first is to use an identifier which names an instance at a party. This was the original BR approach with their $\Pi^s_{i,j}$ notation discussed in Sect. 2.2. The instance is then modelled to include various state variables which may include a session identifier. The second approach is to use explicit session identifiers with the expectation that these will (perhaps eventually) identify unique instances at a particular party. Some models allow session identifiers to evolve over time while for others they are not defined until they are fixed.

There are too many different notations and conventions in use to list them exhaustively. Instead, in Table 2.8 we present prominent examples of such notations. For some of these it is possible to separate the instance identifier (usually a party and an instance number) and the session identifier (typically a partial transcript). For others such a separation is harder.

Table 2.8: Comparing various instance and session styles

Model	Instance identifier	Session identifier	Notes
BR93/95	$\Pi^s_{i,j}$	–	
BPR00	Π^i_U	s	Session identifier s defined by the protocol. Partner identity is part of state. Matching sessions have the same s.
CK01	$(P_i, P_j, s, \mathsf{role})$	s	No explicit s defined. Matching sessions have same s.
HMQV	–	(ID_A, ID_B, Out, In)	Explicit matching rule is defined.
eCK	–	$(\mathsf{role}, ID, \tau)$	τ is the transcript. Explicit matching rule is defined.
eCK-PFS	(P, i)	–	Explicit matching rule defined.
MU08	–	$(ID_A, ID_B, *)$	$*$ is evolving transcript. One session can have multiple matching sessions.

2.5.4 Incorporating Public Key Infrastructure

In early models, specifically BR93 and CK01, the long-term keys of principals were generated with some known algorithm at the start of a run of the adversary. The adversary consequently could have no chance to influence the public keys in use and certain attacks which rely on adversarially chosen keys could not be captured. Later models therefore allowed the adversary to choose the long-term key for corrupted parties. This was first allowed in the BR95 model by allowing the adversary to input a new key to the corrupt query. Later models allow the adversary to update the long-term (public) keys of corrupted parties at any time, either implicitly [453, 470] or through an additional query [718, 188, 308] typically called establishparty.

More recently Boyd *et al.* [135] provided a more comprehensive treatment of certificates including the ability of adversaries to register invalid keys at the certification authority. They claimed for the first time to consider the following three features:

- registration of multiple public keys per user;
- flexible checking by certification authorities via a verification procedure;
- adversarial choice of public keys per session.

2.6 Shoup's Simulation Model

Shoup [674] developed a security model which remains unpublished, even though it has been used by several different authors to analyse various protocols. We call this Shoup's *simulation model*. There are many similarities with the Bellare–Rogaway model, in particular the adversary runs the network to set up all connections, and has various attacking options. We may regard Shoup's model as being more abstract than Bellare and Rogaway's, and indeed it is formally shown to be a generalisation. The major novelty is that Shoup defines two systems, an *ideal* system and a *real* system.

In the ideal system the adversary can run the protocol by initialising users, starting and aborting sessions, and interacting with applications. The adversary thus decides which sessions are connected but the session keys are chosen perfectly randomly and independently of other parameters. These can be obtained by the adversary by choosing to compromise sessions, but otherwise remain secret. A significant difference from the Bellare–Rogaway model is that Shoup's model includes the possibility of using the agreed key in an *application*. Use of the session key is ruled out in the Bellare–Rogaway model since it allows the adversary to distinguish between the session key and a random value.

In the real system the adversary still controls the network but is not given the keys and random values chosen by the principals. The adversary can initialise principals and can invoke instances of principals to respond to messages as in the Bellare–Rogaway model. In addition, a trusted third party, TTP, is defined, and principals can interact with TTP to obtain long-term keys. This allows explicit inclusion of certificate information for public keys in the model, another distinction from that of Bellare and Rogaway. Security is defined in a simple manner.

1. Each principal must terminate after a small (polynomial) number of message interactions.
2. If the adversary simply relays messages faithfully between principals then both accept and share the same session key.
3. For any efficient adversary in the real world there must exist an efficient adversary in the ideal world such that it is computationally infeasible to differentiate between the behaviour of the two adversaries and the information gained by them.

The motivation behind the notion of *simulatability* is that whatever the adversary could gain by interacting with the real system could have been gained in the ideal

system. But by construction the ideal system gains nothing for the adversary. An attractive feature of this security definition is that it is independent of the protocol goals. Shoup defined three shades of security by allowing the adversary different powers to corrupt principals.

Static corruptions cover the case in which the adversary does not use interactions with the principals to decide which principals to corrupt. In this case corruption is not explicitly modelled but any real system adversary is able to *register* principals for which the secrets and random values are known. Shoup proved that security with static corruptions in his simulation model is equivalent to security in the Bellare–Rogaway model. This seems paradoxical since Bellare and Rogaway explicitly allow the adversary to corrupt principals at any time. However, Shoup pointed out that there is an efficient reduction from the Bellare–Rogaway model to the same model in which only static corruptions are allowed.

Adaptive corruptions are those in which the adversary can choose which principals to corrupt at any time. Protocols that are secure against an adversary who can adaptively corrupt must provide forward secrecy. The reason is that if old session keys could be found from a corrupted long-term key then it must be possible to simulate this in the ideal world where no such compromise is available. Shoup showed that security against adaptive corruptions in the simulation model is equivalent to security with forward secrecy in the Bellare–Rogaway model.

Strong adaptive corruptions are adaptive corruptions in which the adversary obtains not only the long-term key of corrupted principals, but also any short-term secrets that have not been explicitly erased. Shoup explained how to maintain a secure session that is compatible with security of such a protocol.

2.7 Models for Enhanced Scenarios

Most models in the literature have focused on the case of two-party key establishment where the parties already have a long-term secret, either shared with a trusted third party or (more commonly in recent years) with a corresponding certified public key. There are various common scenarios which do not fit this case and models have been adapted or enhanced to take care of the differences.

- One example is password-based protocols (see Chap. 8) where the long-term secrets may be easily guessable by the adversary. We discussed in Sect. 2.2.4 how the winning condition of the adversary is typically relaxed in order to take this into account. Boyko *et al.* [145] distinguished between verifier- and non-verifier-based protocols and provided the first security definition for the former. Later, Abdalla *et al.* [16, 17] provided a stronger model, called *real-or-random* security, which allows the adversary unlimited access to the test query which *always* answers with the same choice of either the real session key or a random one. Abdalla *et al.* showed that this model is strictly stronger than the more usual

indistinguishability model. There are also simulation-based definitions for security of password-based protocols as well as ideal functionalities in the universal composability model. Pointcheval [616] provided a nice overview of recent developments.

- Another example is the identity-based scenario (see Chap. 7) where the long-term public key of any principal can be computed from the identity of the principal and public parameters. As in other identity-based settings, it is normal for identity-based key exchange models to provide the adversary with an extract oracle which reveals long-term private keys, subject to usual restrictions. The adversary may also be allowed to access the long-term master key in order to capture forward secrecy against the key generation centre. Formal models for identity-based key exchange were first discussed by Chen and Kudla [194]. Generalisations of identity-based key exchange to the attribute-based and predicate-based settings also required corresponding extensions to the models [104, 327, 694, 765].

In the rest of this section we will consider two scenarios which generalise the basic key exchange models. The first is the case of group key exchange where the number of principals involved is more than two. The second is the case of multi-factor key exchange where the number (and type) of long-term secrets per principal is more than one.

2.7.1 Models for Group Key Exchange

Group key exchange (see Chap. 9) has similar goals to two-party key exchange – to securely establish a session key and, optionally, to provide mutual authentication. As may be expected, to a large extent the models which have evolved for group key exchange have broad similarities to those for the two-party case. There are, however, a few issues that do not arise in the two-party case which have to be considered when making the generalisation.

- In the two-party case both principals are involved in either sending or receiving every message of the protocol. In the group case there can be, and often are, messages sent only between a subset of the principals. This makes partnering more tricky.
- Group key exchange protocols often include extensions to add or remove principals to an already accepted session.
- In the group case a subset of principals can collude in order to try to deceive other principals – these are usually called *insider attacks*.

Bresson *et al.* [148, 149, 154] were the first to generalise the BR models from the two-party to the multi-party case. In the first version of their model [154] the basic adversary queries are the same as we saw in the BR model, but an important distinction is the way that partnering is defined. The session identifier set (SIDS), $\mathsf{SIDS}(\Pi_i^s)$, at an instance Π_i^s, is a set whose elements consists of transcripts of messages between Π_i^s and every other potential instance Π_j^t. Then two instances Π_i^s and Π_j^t are *directly partnered* if they are both in the accepted state and

$\mathsf{SIDS}(\Pi_i^s) \cap \mathsf{SIDS}(\Pi_i^s) \neq \emptyset$. Finally, two instances Π_i^s and Π_j^t are *partnered* if there is a path of directly partnered instances between Π_i^s and Π_j^t. Bresson *et al.* [157] later noted that this definition of partnering fails to capture the intuitive understanding of mutual authentication. They gave examples where m uncorrupted principals accept each other but have different session keys. Later definitions [156, 330] typically demand that partners all share the same session identifier.

In their first paper, Bresson *et al.* [154] only considered static groups defined at the start of the protocol, but they extended this to the dynamic case in their second paper [148]. In the third paper [149] they extended the model once more, in particular including adversary queries to model the addition and removal of principals after sessions have accepted. In order to prove their protocol secure they also made use of a model of a smart card interface as previously used by Shoup and Rubin [676] (see Sect. 2.2.2.)

Insider attacks were first formally modelled by Katz and Shin [415]. Their definition of insider security formalises the intuition that an honest party U should not be deceived about the participation of another honest party U' even if there are dishonest parties in the set of partners expected by U and U'. Note that in a two-party protocol such a situation can never occur since there can be no additional corrupted party to disturb the protocol, so in this sense insider attacks are not relevant for two-party protocols. Katz and Shin ignored the case of KCI attacks in their formal model since they cannot be captured in the universally composable formalism which they used. Gorantla *et al.* [330] later provided a formal definition for insider attacks in a game-based definition.

Later work has expanded on the model to include additional adversary queries, in particular including state reveal and ephemeral key reveal queries similar to those in the CK and eCK models [774, 285].

2.7.2 Models for Multi-factor Key Exchange

Pointcheval and Zimmer [618] extended the basic AKE models to take into account the situation where a client may have three different authentication factors, each of which has different security properties. One is a full-strength cryptographic key, such as may be stored in a cryptographic token, the second is a password which may be memorised by a human client, and the third is a biometric such as a fingerprint. The model allows the adversary to obtain two of the three factors. Password security is defined with a real-or-random security criterion so that the adversary is still allowed to make unlimited (polynomially) guesses through send queries. However, there is a *liveness assumption* with regard to the biometric factor. Instead of making an unrealistic assumption that the biometric is secret, it is assumed that when a client's biometric has not been corrupted, the adversary can only use messages in the send query which have been output by an auxiliary compute query. The compute query takes as input an instance, a full-length secret and a password, while the biometric is chosen randomly from a distribution different from the correct one. The compute query models the adversary's ability to be active in the protocol using its own biometric which will usually be distinct from the correct one. The liveness assumption is

a reflection of the assumption that when a biometric is used for authentication there will be physical mechanisms in place to ensure that the owner of the biometric is present when it is used.

Stebila *et al.* [687] proposed a similar multi-factor model but, instead of modelling three factors of different type, they allowed any number of factors. These factors are all passwords, but each may be one of three types: a password shared with a server; a password for which a server stores only an image, or a one-time password. Fleischhacker *et al.* [279] proposed a generalised framework allowing a mixture of multiple types of factors and showed how to design secure protocols in their model in a generic manner.

2.8 Secure Channels

In the key exchange models we have looked at so far in this chapter, security has been defined to be essentially as strong as possible. The indistinguishability-based definitions used in our three main model classes (BR, CK and eCK) capture the notion that, in a computational sense, the adversary learns nothing useful about the session key. The intuition is that if the session key accepted by the parties is random from the adversary's viewpoint then it should be good for any application requiring a shared key.

It is worth giving greater consideration to the combination of authenticated key exchange with applications for at least two reasons.

1. When we use the session key in a particular application we should be aware that the key exchange protocol and the application in practice run in parallel so we should worry about analysing the security of the combination of key exchange and applications. Just because they are individually secure does not necessarily imply that they are secure when combined. We may need to restrict how the session key is used to ensure that security is maintained.
2. In some applications we may be unable to achieve the strong indistinguishability definition for key exchange. Indeed this turns out to be the case in several prominent real world secure channel protocols whose wide deployment makes them difficult to change. This does not necessarily imply that the key exchange in combination with the application is insecure, depending on what is required for security of the application. Therefore we may need to weaken the security model so that an achievable level of security can be defined.

One of the primary uses of the session key generated by an AKE protocol is for encryption and authentication of application data. The term *secure channel* is often used to describe the process of establishing a session key and then using it to secure application data in this way. In this section we will compare various definitions of secure channels. There has been significantly less work on secure channels than on AKE protocols, though a resurgence of interest in secure channels has come about due to increasing scrutiny of real-world secure channel protocols such as TLS.

2.8.1 CK01 Secure Channels

Section 2.3.2 described the Canetti–Krawczyk model for authenticated key exchange. In the same 2001 paper [178], Canetti and Krawczyk also defined a *secure network channels* protocol and showed how to construct such a channel from a CK01-secure AKE protocol, a secure symmetric encryption scheme, and a secure message authentication code.

The definition of secure network channels created by Canetti and Krawczyk, as well as the intermediate notions we will discuss below, are given in the same *simulation* paradigm as used by Bellare *et al.* [72] and discussed in Sect. 2.3.1. Recall that, in this simulation paradigm, we first define what we consider to be an 'ideally' secure protocol, and then compare this with the real protocol π. Roughly speaking, if π is secure then for any efficient adversary running π there is an efficient adversary against the ideal protocol such that the outputs of the two systems are indistinguishable.

The main definition of secure channels of Canetti and Krawczyk is as follows: a secure network channels protocol is a *network channels protocol* that provides *secure network authentication* and *secure network encryption*. We now define each of these concepts in turn.

Network Channels Protocol

A *network channels protocol* is defined as a combination of a key exchange protocol π and a pair of functions $(\mathrm{snd}, \mathrm{rcv})$, which will provide encryption and authentication of application data. The adversary can interact with a collection of parties implementing the network channels protocol in a way similar to the CK01 AKE model of Sect. 2.3.2. In particular, a session is identified by a tuple (P_i, P_j, s) for a principal P_i intending to communicate with principal P_j using session identifier s. The adversary can make the following queries of parties.

Establish session. The query establish-session allows the adversary to direct party P_i to run the key exchange protocol π and establish a session key k with party P_j and session identifier s.

Send message. The party is given a message m, to which it applies the keyed sending function $\mathrm{snd}_k(m)$ and returns the result m'.

Receive message. The party is given an input m', which in normal operation would be the output of a send query; the party applies the keyed receiving function $\mathrm{rcv}_k(m')$; if the result is not an error, then it records the output.

Expire session. The effect of the expire-session query is to delete the session key from the session specified as input to the query.

In the simulation paradigm, the security experiment maintains a transcript of all query events, such as 'P_i established session s with P_j', or 'P_i send message m to P_j in session s', and so on.

Secure Network Authentication Protocol

A network channels protocol is said to be a *secure network authentication protocol* if it emulates an 'ideal' network authentication protocol. The ideal network authentication protocol has the same adversary interface as described for a network channels protocol, but the operations performed for each query are ideal:

Establish session. The party does not run the key exchange protocol, instead it records in an ideal transcript simply that session s has been established with party P_j.

Send message. The party does not actually apply snd_k or return a value to the adversary. Instead the party simply records that message m is sent to the recipient P_j.

Receive message. The party does not actually receive any value from the adversary or apply rcv_k. Instead the party simply records that the message from P_i is received.

Expire session. The party has no session key to delete, and simply records the session as expired.

Clearly, an ideal network authentication protocol really does provide authenticated transmission of messages, since they are sent and received over an ideally authenticated channel, rather than passing back through the adversary's hands.

A secure network authentication protocol can be constructed by using a secure (existentially unforgeable under chosen message attack) message authentication code and appending the MAC tag to the message.

Secure Network Encryption Protocol

A network channels protocol is said to be a *secure network encryption protocol* if it provides confidentiality of communications, roughly in the sense of indistinguishability of messages under chosen plaintext attack. In particular, the standard network channels protocol experiment is extended with the following adversary query:

Test session. The adversary can, once, indicate a single test session (P_i, P_j, s), as well as two equal-length messages m_0 and m_1. A secret bit b is chosen (but not given to the adversary), and P_i is activated with $\mathsf{send}(P_i, P_j, s, m_b)$. The output is returned to the adversary.

The adversary outputs a bit b', its guess for b, and wins if it guesses correctly. The adversary is also allowed to corrupt parties and reveal session keys and states, provided it does not expose values that would allow it to trivially win the game. A network encryption protocol is said to be secure if the adversary's advantage in guessing b is negligible in the security parameter.

Secure Network Channels Protocol

A network channels protocol is said to be a secure channels protocol if it is both a secure network authentication protocol and a secure network encryption protocol.

Canetti and Krawczyk showed how to construct a secure channels protocol from a CK01-secure key exchange protocol π, a pseudorandom function family f, an IND-CPA secure symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$, and an unforgeable message authentication code MAC in the natural way, using an encrypt-then-MAC construction. In particular, the parties first execute the key exchange protocol π to derive a session key k. Then both parties compute the encryption key $k_0 = f(k,0)$ and the MAC key $k_1 = f(k,1)$. To send a message m, the sender constructs $c \| t$, where $c = \mathsf{Enc}_{k_0}(m)$ and $t = \mathsf{MAC}_{k_1}(c)$. The receiver verifies and then decrypts analogously.

2.8.2 CK02 Secure Channels

In 2002, Canetti and Krawczyk [181] updated their aforementioned 2001 paper [178] to create *universally composable* notions of key exchange and secure channels.

The UC framework is, like the BCK98 definitions, simulation-based, but is meant to be stronger, and it aims to ensure security when the protocol is composed with any other secure protocol.

- In the original BCK98 simulation framework, no adversary can distinguish between interacting with the real system and with an ideal system. In other words, for every adversary interacting with the real system, there exists a simulator interacting with the ideal system such that the two worlds have indistinguishable distributions.
- In the UC framework, in addition to the main adversary there is another adversarial entity, the environment, which prepares all inputs to the protocol(s). No environment should be able to distinguish between interacting with the adversary and the real protocol(s), or with the simulator and the ideal protocol.

Security in the UC framework is defined in terms of emulation of an ideal functionality. In the secure network channels ideal functionality, the parties are directed to establish a secure channel between them; then, when one party is directed to send a message to another party, the message is delivered over an ideal private connection, while the adversary is told the length of the message.

2.8.3 Authenticated and Confidential Channel Establishment (ACCE) Protocols

Several prominent real-world protocols, including the Transport Layer Security (TLS) protocol and the Secure Shell (SSH) protocol, aim to provide a secure channel. They do so by first establishing a session key using a key exchange protocol, and then using that key to perform authenticated encryption for application data. At a high level, this matches the approach of CK01. While the approach of Canetti and Krawczyk to defining and constructing secure channels is attractive – the definitions

are relatively simple, and the construction is elegant and modular – it is not always appropriate for analysing protocols that arise in practice.

Jager, Kohlar, Schäge, and Schwenk [392] proposed the *authenticated and confidential channel establishment (ACCE) protocols model* in 2012 for the purposes of modelling the security of signed-Diffie–Hellman ciphersuites in TLS. There were three main motivations for introducing a new model. First, the CK01 secure channels definition is in the simulation framework, rather than the more widely used game-based approach. Second, the security properties provided by TLS are more granular than properties captured by CK01 secure channels. And third, the construction of TLS does make use of a key exchange protocol and an authenticated encryption scheme. Instead there is an overlap between these two components, where key confirmation messages from the key exchange protocol are sent over the encrypted channel using the same key used for application data. This makes it impossible to prove the real session key in TLS is indistinguishable from a random key.

The ACCE model is a Bellare–Rogaway style model, with some changes. There are two security goals in the ACCE model: *entity authentication* and *channel confidentiality and integrity*. Entity authentication is defined in terms of matching conversations like in the BR93 model for authenticated key exchange, and the adversarial interaction in the entity authentication is similar as well.

The most significant change from AKE security models is of course that the ACCE model includes secure transmission of application messages as an explicit goal. Thus, the execution is divided into two phases:

- In the *pre-accept phase*, parties typically execute an authenticated key exchange protocol, performing mutual authentication and establishing a session key. (In TLS, this corresponds to the handshake protocol.) Upon successful authentication, a party enter the *accept state*. During the pre-accept phase, a session key is established.
- In the *post-accept phase*, parties can transmit application data over the encrypted and authenticated channel. (In TLS, this corresponds to the record layer protocol.)

The adversary can interact with each party's execution π_i^s using the following queries.

Send pre-accept phase protocol message. This send query directs the party to process a protocol message in the pre-accept phase. It has no effect once the party has entered the accept state.

Session key reveal and **Long-term key reveal**. These queries allow the adversary to obtain the session key of any accepted session or obtain the long-term secret key of a party.

The above queries are similar to those typically found in the key exchange security models seen earlier in this chapter. However, the ACCE model does not include a test query for session key indistinguishability. Instead, the security experiment explicitly models the security of the channel. The main idea is that, in the post-accept phase of each session, the adversary plays a *stateful authenticated encryption* game,

meaning the adversary attempts either to distinguish which of two messages was encrypted by the sender (an IND-CPA-like game) or to cause a receiver to accept a ciphertext that was not sent by the corresponding sender.

Specifically, in each session π_i^s, each party has a secret random bit b_i^s, and the adversary's goal is to guess the bit in any uncompromised session. (Here 'uncompromised' means, as usual, that the adversary has not revealed the session key of the session (or its partner) or compromised the long-term key of the partner prior to acceptance.) The session's secret bit b_i^s is used for two purposes simultaneously. Firstly, b_i^s is used to choose which of two adversary-supplied messages is encrypted (as in an IND-CPA security experiment for encryption). Secondly, b_i^s is implicitly leaked to the adversary if the adversary successfully injects a forged ciphertext.

An adversary in the ACCE model is deemed successful if one of two events happens: either the adversary causes some uncompromised session to *accept maliciously* (i.e. without a matching session), or the adversary guesses the hidden bit b_i^s of any uncompromised session π_i^s with probability significantly better than $1/2$.

Variants

Several variants of ACCE have been developed since it was first proposed. The original ACCE definition of Jager *et al.* aimed to capture stateful *length-hiding* authenticated encryption *with auxiliary data* as the notion most appropriate to TLS. Additionally, ACCE has been extended to cover:

- other types of authentication, specifically server-only authentication [442, 456], authentication using pre-shared symmetric keys [489], and passwords [517];
- other security properties of real-world protocols, such as renegotiation [304] and multi-ciphersuite security [89]; see Chap. 6 for more information on these concepts.

2.9 Conclusion

In this chapter we have attempted to identify the major developments in computational models for key exchange without getting drawn too deeply into the technical details. A general trend has been that models have become more complex since the original computational model of Bellare and Rogaway was proposed in 1993. While the vast array of available models provides a rich arsenal from which the protocol analyst can choose a suitable weapon, it is usually difficult to compare results proven in different models. As we saw in Sect. 2.5, it is often the case that between two models neither one is stronger, and currently there is no common agreement on the "right" model for key exchange.

From the observed trends it is hard not to predict that new, probably even more complex, models will be proposed. One consequence of this is that proofs by humans becomes ever more difficult and error-prone. In the past few years there has

been a new emphasis on tools for helping to deal with proof complexity in computational security models [58, 117] and such tools have been successfully applied to key exchange models [57]. Progress towards unifying existing models would be very beneficial. At the same time, new directions in key exchange models, such as the ideas of George and Rackoff [301], may be overdue.

3

Protocols Using Shared Key Cryptography

3.1 Introduction

The majority of protocols for key establishment and entity authentication that have been proposed in the literature concentrate on the case where there are exactly two users who wish to communicate or establish a session key. This is commonly referred to as the two-party case. In this chapter we discuss two-party key establishment and authentication protocols based on symmetric algorithms. The next chapter discusses two-party protocols using public key algorithms, while the multi-party case is covered in Chap. 6.

We can classify two-party key establishment protocols using the following two criteria discussed in Chap. 1.

1. The cryptographic keys available *a priori*.
2. The method of session key generation.

If only shared keys are available to establish a new session key, there are essentially two cases to consider with respect to criterion 1.

(a) The two principals already share a secret key.
(b) Each principal shares a key with a trusted server.

Criterion 2 is concerned with the method of session key generation, for which there are three different possibilities in general: key transport, key agreement and hybrid. If a protocol has only two principals and is server-less, as in case (a) above, one cannot distinguish between key transport and hybrid key generation. The criteria mentioned above lead to the classification of $1 \times 2 + 1 \times 3 = 5$ different classes of protocols. The recognition of the criteria allows two-party shared key protocols found in the literature to be classified into one of the above five classes. However, in this chapter we mainly emphasise the division between server-less and server-based protocols.

In the remainder of this section, we explain the notation used to describe protocols in this chapter. Section 3.2 discusses protocols aimed at providing entity authentication without key establishment. Section 3.3 discusses protocols aimed at providing key establishment without a server, including key transport protocols and key

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_3

agreement protocols. Section 3.4 discusses protocols aimed at providing key establishment using a server, including key transport protocols, key agreement protocols and hybrid protocols. For each class of protocols, we provide published protocols from the literature.

Notation

A widely used notation for denoting a message part as a ciphertext is $\{M\}_K$ where M is the input data to a symmetric encryption algorithm that is parametrised by the secret key K. There are other familiar notations for indicating the use of encryption when specifying protocols but the above notation remains a popular one. Frequently, protocol designers have used the above-mentioned notation to imply that encryption provides both confidentiality and integrity properties. Recall from Chap. 1 that there are many variants of encryption providing security against different threats. Ignoring such differences leads to problems for implementers and also prevents proper security analysis.

If the protocol designer requires encryption for achieving both confidentiality and integrity, in other words if the protocol requires authenticated encryption, then we will use the above notation for encryption when presenting the protocol. Some designers use one notation for cryptographic transformations that provide message integrity and another notation for cryptographic transformations that provide confidentiality. In such cases, we use the notation $[\![\cdot]\!]_K$ to denote a ciphertext obtained from an encryption algorithm that provides the confidentiality property alone and the notation $[\cdot]_K$ to denote a ciphertext obtained from a one-way cryptographic transformation such as a MAC which provides the integrity property alone.

The notation used in this chapter is summarised in Table 3.1.

Table 3.1: Notation used throughout Chap. 3

A and B	The two users who wish to share a new session key
S	A trusted server
ID_A, ID_B, ID_S	The identities of A, B and S
$\{M\}_K$	Authenticated encryption of message M with key K
$[\![M]\!]_K$	Encryption of message M with key K to provide confidentiality
$[M]_K$	One-way transformation of message M with key K to provide integrity

3.2 Entity Authentication Protocols

Protocols that aim at providing entity authentication without key establishment are relatively scarce in the literature. Perhaps this is because the variety of approaches is

quite limited, or maybe because the usefulness of such protocols is questionable as discussed in Chap. 2. In this section we examine the prominent examples and discuss whether they achieve the definitions of entity authentication introduced in Chap. 2, or the simpler liveness property.

3.2.1 Bird–Gopal–Herzberg–Janson–Kutten–Molva–Yung Protocols

A paper by IBM researchers Bird *et al.* [103] in 1993 was one of the first to demonstrate a wide class of attacks on several authentication protocols, including a draft protocol proposed by ISO. Based on the attacks on these protocols, they developed a set of security criteria to avoid such attacks and proposed protocols that meet their criteria. They started with a basic protocol which they did not regard as secure, and improved it in a series of steps through consideration of various attacks and other design requirements. Eventually, a good protocol was achieved.

Protocol 3.1 is the basic protocol of Bird *et al.* Here u and v are two functions such that K_{AB} is needed to calculate them, and they do not give away K_{AB}.

1. $A \to B : N_A$
2. $B \to A : N_B, u(K_{AB}, N_A, \ldots)$
3. $A \to B : v(K_{AB}, N_B, \ldots)$

Protocol 3.1: Bird *et al.* canonical protocol 1

Initially we will assume that the functions u and v are the same. Under this assumption the protocol does not achieve the matching conversation goal, which Bird *et al.* regarded as important for the security of any authentication protocol. In Attack 3.1, A is used as an oracle against B. Suppose I is an adversary who wishes to attack the protocol.

- I starts a protocol run with B while masquerading as A.
- In parallel, I starts a protocol run with A while masquerading as B.

1. $I_A \to B : N_I$
2. $B \to I_A : N_B, u(K_{AB}, N_I, \ldots)$
1′. $I_B \to A : N_B$
2′. $A \to I_B : N_A, u(K_{AB}, N_B, \ldots)$
3. $I_A \to B : u(K_{AB}, N_B, \ldots)$

Attack 3.1: An oracle attack on Protocol 3.1

In Attack 3.1, B accepts even though the conversations do not match. In light of this attack, Bird *et al.* concluded that the functions u and v must be different from one another so that A's reply in the parallel session cannot be used by the adversary to complete the first run. While the condition that u and v be different is adequate to prevent the attack, it is not necessary to achieve the protocol goals. As long as the identities of the sender and the intended partner are included inside the functions u and v, the authentication goal can be achieved even if the functions u and v are the same. Of course, a similar attack on such a protocol is still possible, but it would not violate the protocol goal.

3.2.2 Bellare–Rogaway MAP1 Protocol

The MAP1 mutual authentication protocol was proposed in a landmark paper of Bellare and Rogaway [75]. They provided a formal definition of matching conversations and showed that MAP1 is provably secure. The messages are shown in Protocol 3.2.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, [ID_B, ID_A, N_A, N_B]_{K_{AB}}$
3. $A \rightarrow B : [ID_A, N_B]_{K_{AB}}$

Protocol 3.2: Bellare–Rogaway MAP1 protocol

In the Bellare–Rogaway model of security (see Sect. 2.2), an adversary may only interact with sessions of the same protocol. An attack of Alves-Foss [30] shows why this assumption is important in practice. He designed Protocol 3.3, known as EVE1, which can also be shown to be provably secure.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, [ID_A, ID_B, N_A, N_B]_{K_{AB}}$
3. $A \rightarrow B : [ID_A, N_B]_{K_{AB}}$

Protocol 3.3: Protocol for attacking MAP1 protocol

The only difference between MAP1 and EVE1 is that the identities of A and B are swapped in message 2. A chosen protocol attack on the MAP1 protocol is now possible. Suppose I is an adversary who wishes to attack the protocol. In Attack 3.2 A is used as an oracle against herself.

- I masquerades as B in a run of the MAP1 protocol started by A.
- In parallel, I starts a run of the EVE1 protocol with A while masquerading as B.

1. $A \rightarrow I_B : N_A$
1′. $I_B \rightarrow A : N_A$
2′. $A \rightarrow I_B : N'_A, [ID_B, ID_A, N_A, N'_A]_{K_{AB}}$
2. $I_B \rightarrow A : N'_A, [ID_B, ID_A, N_A, N'_A]_{K_{AB}}$
3. $A \rightarrow I_B : [ID_A, N'_A]_{K_{AB}}$

Attack 3.2: Chosen protocol attack on MAP1

Is Attack 3.2 on the MAP1 protocol valid? A reasonable conclusion may be that the attack is invalid since it violates an assumption of the model used in proving the protocol secure. On the other hand, the attack is a reminder that provable security does not guarantee security against chosen protocol attacks.

3.2.3 ISO/IEC 9798-2 Protocols

The international standard ISO/IEC 9798 Part 2 [380] specifies six protocols using symmetric encryption algorithms. Four of these protocols are intended to provide entity authentication alone, while two are intended to provide key establishment as well as entity authentication. These last two are essentially identical to two protocols in the ISO/IEC 11770-2 standard, which are described in Sect. 3.4.4. Two of the four protocols aimed solely at entity authentication are concerned with unilateral authentication, while the other two are concerned with mutual authentication. Below we examine these protocols with some optional text fields omitted.

The first protocol, shown as Protocol 3.4, consists of a single message from a claimant A to a verifier B. It provides unilateral entity authentication of A to B. The timestamp T_A allows B to deduce that A is live, while inclusion of the identity B ensures that A has knowledge of B as her peer entity.

$A \rightarrow B : \{T_A, ID_B\}_{K_{AB}}$

Protocol 3.4: ISO/IEC 9798-2 one-pass unilateral authentication protocol

The second protocol (Protocol 3.5) is similar to the first, but uses a nonce instead of a timestamp. It provides the same properties as Protocol 3.4.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_B, ID_B\}_{K_{AB}}$

Protocol 3.5: ISO/IEC 9798-2 two-pass unilateral authentication protocol

The third protocol (Protocol 3.6) is constructed from two instances of Protocol 3.4. It provides mutual entity authentication between A and B.

1. $A \rightarrow B : \{T_A, ID_B\}_{K_{AB}}$
2. $B \rightarrow A : \{T_B, ID_A\}_{K_{AB}}$

Protocol 3.6: ISO/IEC 9798-2 two-pass mutual authentication protocol

Attack 3.3 on Protocol 3.6 was found by Basin, Cremers and Meier [61] using the tool Scyther. The result is that A rejects the first run and instead accepts the other run in which adversary I masquerades as B. Both parties accept and could correctly infer that the other has indicated a recent willingness to communicate. However, Attack 3.3 shows that A and B do not agree on their roles: both accept as responders (while A rejects as initiator in the first run). This shows that the protocol does not guarantee the property known as *injective agreement*.

1. $A \rightarrow B : \{T_A, ID_B\}_{K_{AB}}$
2. $B \rightarrow I_A : \{T_B, ID_A\}_{K_{AB}}$
$1'$. $I_B \rightarrow A : \{T_B, ID_A\}_{K_{AB}}$
$2'$. $A \rightarrow I_B : \{T'_A, ID_B\}_{K_{AB}}$

Attack 3.3: Attack on Protocol 3.6

In order to avoid Attack 3.3, Basin *et al.* [61] proposed that messages in this protocol (and indeed for all protocols in the standard) should include elements to prevent messages being interchanged with messages in other protocols or with different messages within the same protocol. This can be done by including a protocol and message identifier in each message. This proposal was made mandatory in a 2013 corrigendum to the 9798-2 standard.[1]

The fourth protocol (Protocol 3.7), like Protocol 3.6, is aimed at providing mutual authentication but uses nonces instead of timestamps. Notice that Protocol 3.7 is not simply a combination of two instances of the nonce-based unilateral authentication protocol (Protocol 3.5) in which the number of messages has been reduced from four to three. Instead, the second message includes both nonces which binds them together. This design may be chosen because the standard implicitly regards the matching conversation goal as important for security.

In all of the above four protocols, the inclusion of the identity of B in the encrypted message from A is optional. Furthermore, in Protocol 3.6, the inclusion of

[1] Technical Corrigendum 3 to ISO/IEC 9798-2:2008, February 2013.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_A, N_B, ID_B\}_{K_{AB}}$
3. $B \rightarrow A : \{N_B, N_A\}_{K_{AB}}$

Protocol 3.7: ISO/IEC 9798-2 three-pass mutual authentication protocol

the identity of A in the encrypted message from B is also optional. The standard comments that these fields are included to prevent reflection attacks. Their inclusion is made optional so that the protocols can be 'optimised' for networking environments where such attacks are precluded by other means.

3.2.4 ISO/IEC 9798-4 Protocols

The international standard ISO/IEC 9798 Part 4 [378] specifies four protocols using a cryptographic check function or, in other words, a message authentication code. All of these protocols are intended to provide entity authentication alone, and they correspond very closely to the first four protocols in the ISO/IEC 9798 Part 2 standard examined in Sect. 3.2.3. Thus two of the four protocols are concerned with unilateral authentication, while the other two are concerned with mutual authentication. As before we examine these protocols with some optional text fields omitted.

The first protocol, shown as Protocol 3.8, consists of a single message from a claimant A to a verifier B. It provides unilateral entity authentication of A to B. The timestamp T_A allows B to deduce that A is live, while inclusion of the identity B ensures that A has knowledge of B as her peer entity.

$A \rightarrow B : T_A, \mathsf{MAC}_{K_{AB}}(T_A, ID_B)$

Protocol 3.8: ISO/IEC 9798-4 one-pass unilateral authentication protocol

The second protocol (Protocol 3.9) is similar to the first, but uses a nonce instead of a timestamp. It provides the same properties as Protocol 3.8.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \mathsf{MAC}_{K_{AB}}(N_B, ID_B)$

Protocol 3.9: ISO/IEC 9798-4 two-pass unilateral authentication protocol

The third protocol (Protocol 3.10) is constructed from two instances of Protocol 3.8. It provides mutual entity authentication between A and B. An attack, essentially the same as Attack 3.3 on Protocol 3.10, was found by Basin *et al.* [61] which shows that A and B do not necessarily agree on their roles.

1. $A \rightarrow B : T_A, \mathsf{MAC}_{K_{AB}}(T_A, ID_B)$
2. $B \rightarrow A : T_B, \mathsf{MAC}_{K_{AB}}(T_B, ID_A)$

Protocol 3.10: ISO/IEC 9798-4 two-pass mutual authentication protocol

As before, to avoid the attack messages in the protocol should include elements to prevent messages being interchanged with messages in other protocols or with different messages within the same protocol. This proposal was made mandatory in a 2012 corrigendum to the 9798-4 standard.[2]

The fourth protocol (Protocol 3.11), like Protocol 3.10, is aimed at providing mutual authentication but uses nonces instead of timestamps.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, \mathsf{MAC}_{K_{AB}}(N_A, N_B, ID_B)$
3. $B \rightarrow A : \mathsf{MAC}_{K_{AB}}(N_B, N_A)$

Protocol 3.11: ISO/IEC 9798-4 three-pass mutual authentication protocol

Similar to the protocol in ISO/IEC 9798-2 examined in Sect. 3.2.3, in all of the ISO/IEC 9798-4 protocols, the inclusion of the identity of B in the encrypted message from A is optional. Furthermore, in Protocol 3.10, the inclusion of the identity of A in the encrypted message from B is also optional.

3.2.5 Woo–Lam Authentication Protocol

All the entity authentication protocols we have looked at so far assume that a shared key already exists between the two principals involved. Woo and Lam [736] devised several protocols for authentication. One of these was a unilateral authentication protocol using a trusted server as a *key translation centre* with the job of converting messages encrypted with one key that it knows into messages encrypted with a different key. Protocol 3.12 shows the message flows.

The idea is that B chooses his nonce N_B and challenges A to encrypt it with K_{AS}. On receipt of the purported encryption, B asks S to translate it into an encryption with K_{BS}, which B can decrypt and check. There are several attacks known on Protocol 3.12, the first of which was found by Abadi (as attributed by Woo and Lam [737]).

As shown in Attack 3.4, the adversary I starts two runs with B, in one of which I claims to be A. The two protocol runs continue in parallel but I simply sends a random value R when asked to respond to the challenge intended for A. Furthermore, I uses the challenge intended for A in the encryption for the legitimate run. The server S can only successfully translate the properly encrypted ciphertext but the returned

[2] Technical Corrigendum 2 to ISO/IEC 9798-4:1999, July 2012.

1. $A \rightarrow B : ID_A$
2. $B \rightarrow A : N_B$
3. $A \rightarrow B : \{N_B\}_{K_{AS}}$
4. $B \rightarrow S : \{ID_A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Protocol 3.12: Woo–Lam unilateral authentication protocol

value is correct only for the run where I masquerades as A. The result is that B accepts the run in which I is masquerading as A and rejects the other run.

1. $I_A \rightarrow B : ID_A$
1′. $I \rightarrow B : ID_I$
2. $B \rightarrow I_A : N_B$
2′. $B \rightarrow I : N'_B$
3. $I_A \rightarrow B : R$
3′. $I \rightarrow B : \{N_B\}_{K_{IS}}$
4. $B \rightarrow S : \{ID_A, R\}_{K_{BS}}$
4′. $B \rightarrow S : \{ID_I, \{N_B\}_{K_{IS}}\}_{K_{BS}}$
5. $S \rightarrow B : \{N_B\}_{K_{BS}}$

Attack 3.4: Abadi's attack on Protocol 3.12

Woo and Lam [737] considered a number of variants of Protocol 3.12 in an effort to identify the precise source of the problem. Clark and Jacob [217] showed that most of these were still vulnerable to typing attacks. However, such attacks are prevented if principals can detect replay of messages they have created, which is an assumption of Woo and Lam.

3.2.6 Comparison of Entity Authentication Protocols

Table 3.2 summarises the major properties of the seven entity authentication protocols described earlier. For the goals, an entry (*) indicates that the goal is claimed for the protocol but fails due to attack. In the second column, we indicate whether the definition of entity authentication given in Definition 13 is achieved. From the table, we see that all the protocols meet this definition.

In many protocols, knowledge of the peer entity is conveyed implicitly in the authentication message. For example, in the Bellare–Rogaway MAP1 protocol even though the identity of B is not included in the authentication field of message 3, it is possible to infer it through the use of K_{AB}. In the final column, we indicate whether specific attacks have been proposed in the literature. As discussed in Sect. 3.2.1, the

Table 3.2: Summary of major properties of specific entity authentication protocols

Properties → ↓ *Protocol*	Liveness	Entity authentication	Attack
Bird *et al.* canonical 1 (3.1)	$A+B$	$A+B$	Yes
Bellare–Rogaway MAP1 (3.2)	$A+B$	$A+B$	Yes
9798-2 one-pass unilateral (3.4)	B	B	No
9798-2 two-pass unilateral (3.5)	B	B	No
9798-2 two-pass mutual (3.6)	$A+B$	$A+B$	No
9798-2 three-pass mutual (3.7)	$A+B$	$A+B$	No
Woo–Lam (3.12)	B (*)	B (*)	Yes

attack on the Bird *et al.* canonical protocol fails to violate the authentication goal. The chosen protocol attack on the Bellare–Rogaway MAP1 protocol does violate the authentication goal, but that attack was aimed at showing a limitation of security proofs in practice rather than showing a weakness of the protocol. The attacks on the ISO/IEC 9798-2 protocols discussed in Sect. 3.2.3 are avoided by using the latest version of the standard including corrigenda.

3.3 Server-Less Key Establishment

This section discusses protocols that allow keys to be established directly between two users without the use of a server. The protocols considered require that the two users already share a long-term secret key and may require either that one user generates the established key (key transport) or that both users contribute part of the established key (key agreement).

Table 3.3 gives additional notation used in this section. In the remainder of this section, we examine server-less key transport protocols followed by server-less key agreement protocols.

Table 3.3: Additional notation used for server-less protocols

K_{AB}	The long-term key initially shared by A and B
K'_{AB}	The value of the new session key

3.3.1 Andrew Secure RPC Protocol

Although dating from 1989, the Andrew secure RPC protocol [654], shown in Protocol 3.13, is still a widely used example in the literature. The protocol has two rather

independent components. In the first three messages, A and B perform a handshake using a key they already share, K_{AB}. In the final message, B sends a new session key K'_{AB} to A. Nonce N_A is chosen by A and nonces N_B, N'_B are chosen by B.

1. $A \to B: \{N_A\}_{K_{AB}}$
2. $B \to A: \{N_A + 1, N_B\}_{K_{AB}}$
3. $A \to B: \{N_B + 1\}_{K_{AB}}$
4. $B \to A: \{K'_{AB}, N'_B\}_{K_{AB}}$

Protocol 3.13: Andrew secure RPC protocol

Burrows *et al.* [171] pointed out a major problem with Protocol 3.13: A has no assurance that K'_{AB} is fresh. An intruder could substitute a previously recorded message 4 (from B to A) and force A to accept an old, possibly compromised, session key. Another problem was pointed out by Clark and Jacob [216]. They proposed Attack 3.5, a typing attack in which an intruder records message 2 and substitutes it in place of message 4.

1. $A \to B: \{N_A\}_{K_{AB}}$
2. $B \to A: \{N_A + 1, N_B\}_{K_{AB}}$
3. $A \to B: \{N_B + 1\}_{K_{AB}}$
4. $I_B \to A: \{N_A + 1, N_B\}_{K_{AB}}$

Attack 3.5: Clark–Jacob attack on Andrew protocol

The result of the attack is that A accepts the value $N_A + 1$ as a session key with B. Clark and Jacob pointed out that the potential damage of Attack 3.5 depends on what property the nonce N_A is assumed to have. If N_A is a predictable nonce such as a counter value, then an attacker could force A into accepting a bogus quantity as a session key, whose value could be known to the attacker. If N_A were random, however, then the potential damage of the attack is not so immediate since there is no release of the session key.

It is interesting to consider a revised version of the protocol suggested by Burrows *et al.* shown as Protocol 3.14. Their idea was to change the treatment of the nonces used in the protocol. The nonce N_A need not be secret; when sent by A in plaintext it still forms a typical usage of the challenge–response mechanism. The nonce N_B could be omitted altogether. Instead of sending N_B, B could send a key K'_{AB} along with A's nonce in message 2. Further differences can be found in the last two messages of Protocol 3.14. In the second last message, the encryption with K'_{AB} is intended to assure B that A knows the key. In the last message, B sends N'_B in

plaintext since its purpose is not connected with protocol, but with the subsequent communications session.

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : \{N_A, K'_{AB}\}_{K_{AB}}$
3. $A \rightarrow B : \{N_A\}_{K'_{AB}}$
4. $B \rightarrow A : N'_B$

Protocol 3.14: Revised Andrew protocol of Burrows *et al.*

Lowe [502] published Attack 3.6 on Protocol 3.14. An intruder I engages in two protocol runs with A while masquerading as B. In one of these runs I is the initiator of the protocol, while in the other A is induced to act as initiator.

1. $A \rightarrow I_B : ID_A, N_A$
1′. $I_B \rightarrow A : ID_B, N_A$
2′. $A \rightarrow I_B : \{N_A, K'_{AB}\}_{K_{AB}}$
2. $I_B \rightarrow A : \{N_A, K'_{AB}\}_{K_{AB}}$
3. $A \rightarrow I_B : \{N_A\}_{K'_{AB}}$
3′. $I_B \rightarrow A : \{N_A\}_{K'_{AB}}$
4. $I_B \rightarrow A : N_I$
4′. $A \rightarrow I_B : N'_A$

Attack 3.6: Lowe's attack on revised Andrew protocol

The result of the attack is that A has completed two successful runs of the protocol, apparently with B, although B has not engaged in the protocol. More importantly, the attack defeats goals concerning entity authentication rather than key establishment. The attack is valid if either entity authentication or key confirmation was a protocol goal. Lowe proposed to fix the protocol by adding B's identity to message 2. A similar attack was found by Liu *et al.* [499] on an alternative revised version of the protocol suggested by Burrows *et al.* in which the final encrypted message received by A includes A's nonce.

3.3.2 Janson–Tsudik 2PKDP Protocol

Janson and Tsudik [394] proposed Protocol 3.15 which extends a two-party authentication protocol of Bird *et al.*[103] to provide key establishment. One distinctive aspect of 2PKDP is that it employs two separate cryptographic algorithms: one algorithm that provides confidentiality and another that provides authentication. The

algorithm used for confidentiality purposes is bitwise exclusive-or, while that for authentication is a MAC algorithm. The protocol design allows an encryption-based (CBC-MAC) or hash-based MAC algorithm to be used. In the first message, A sends her nonce N_A to B. In the second message, B generates a new session key K'_{AB} and computes two values, $AUTH$ and $MASK$, using K_{AB}.

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : AUTH, MASK \oplus K'_{AB}$
3. $A \rightarrow B : [N_A, K'_{AB}, A]_{K_{AB}}$

Protocol 3.15: Janson–Tsudik 2PKDP protocol

The quantities $AUTH$ and $MASK$ are defined as follows:

$$AUTH = [N_A, K'_{AB}, B]_{K_{AB}}$$
$$MASK = [\![AUTH]\!]_{K_{AB}}.$$

The second part of message 2 can be viewed as analogous to an encryption of K'_{AB} using a one-time pad. Upon receiving message 2, A computes a MAC under K_{AB} of the received $AUTH$ value to allow decryption of the session key from the second part of message 2, then verifies that the first part of message 2 agrees with the MAC under K_{AB} of the nonce sent earlier, the received session key, and B's identity. Verification of the first part of message 2 implies key freshness as well as key integrity. Upon receiving message 3, B verifies that the received value is correct, which implies key confirmation of A to B.

3.3.3 Boyd Two-Pass Protocol

Protocol 3.16 by Boyd [130] allows both A and B to contribute part of the established key. The messages sent are simply the random numbers chosen by A and B. The new

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B$

Protocol 3.16: Boyd two-pass protocol

key is $K'_{AB} = f(N_A, N_B, K_{AB})$, where f is a combining function such that it must be infeasible to find $f(.,.,K_{AB})$ without knowledge of K_{AB}, even after repeated use. This property is necessary to ensure key authentication (that is, the secrecy of K'_{AB}). Another property required of f is that it is one-way in the first two inputs. This property is necessary to ensure key freshness. For a concrete example, any practical secure MAC algorithm may be chosen for f.

3.3.4 ISO/IEC 11770-2 Server-Less Protocols

The international standard ISO/IEC 11770 Part 2 [381] specifies 13 protocols using symmetric encryption algorithms. Six of these protocols are server-less, while the other seven rely on a trusted server. The server-based protocols are described in Sect. 3.4.4. Some of the protocols make use of a *key derivation function* $f(\cdot)$ in forming the new session key from two or more inputs. Two examples of f are given in the appendix of the standard: one is bitwise XOR of the inputs and the other is application of a hash function to the concatenation of the inputs. Cremers and Horvat [229] performed an analysis of the protocols in ISO/IEC 11770-2 using the Scyther tool. For the six protocols discussed in this section they did not find any violations of the security properties claimed in the standard.

The first two of the six server-less protocols each use only one message pass and provide only implicit key authentication. Mechanism 1, shown as Protocol 3.17, consists only of the encrypted timestamp of A. The new session key is derived as $K'_{AB} = f(K_{AB}, T_A)$ where f is the key derivation function. The single message in mechanism 2, shown as Protocol 3.18, consists only of the new encrypted session key. This means that B gains no assurance about its freshness.

$A \rightarrow B : \{T_A\}_{K_{AB}}$

Protocol 3.17: ISO/IEC 11770-2 Key Establishment Mechanism 1

$A \rightarrow B : \{K'_{AB}\}_{K_{AB}}$

Protocol 3.18: ISO/IEC 11770-2 Key Establishment Mechanism 2

The other four server-less protocols are derived from each of the four two-party entity authentication protocols that were described in Sect. 3.2.3 by adding a key (or more generally *keying material*) to each encrypted message. The next four protocols, Protocols 3.19 to 3.22, may be compared with the four entity authentication protocols, Protocols 3.4 to 3.7. The entity authentication properties of each corresponding pair are the same.

Mechanism 3, shown as Protocol 3.19, consists of a single message from a claimant A to a verifier B. The new session key, K'_{AB}, is chosen by A and encrypted for B. Both A and B achieve the good key property, but only B achieves key confirmation.

$A \rightarrow B : \{T_A, ID_B, K'_{AB}\}_{K_{AB}}$

Protocol 3.19: ISO/IEC 11770-2 Key Establishment Mechanism 3

Mechanism 4, shown as Protocol 3.20, uses a nonce instead of a timestamp. The properties achieved are the same as for Mechanism 3.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_B, ID_B, K'_{AB}\}_{K_{AB}}$

Protocol 3.20: ISO/IEC 11770-2 Key Establishment Mechanism 4

Mechanism 5, shown as Protocol 3.21, is constructed from two instances of Protocol 3.19. Now both A and B choose keying material, F_{AB} and F_{BA} respectively. (Optionally either one of these may be omitted.) The session key is derived as $K'_{AB} = f(F_{AB}, F_{BA})$ where f is the key derivation function. Both A and B obtain the good key property of Definition 15, but if F_{BA} is included in the session key derivation only A will achieve key confirmation.

1. $A \rightarrow B : \{T_A, ID_B, F_{AB}\}_{K_{AB}}$
2. $B \rightarrow A : \{T_B, ID_A, F_{BA}\}_{K_{AB}}$

Protocol 3.21: ISO/IEC 11770-2 Key Establishment Mechanism 5

Mechanism 6, shown as Protocol 3.22, is similar to Protocol 3.21. Keying material is provided from both parties and the session key is calculated in the same way. A major difference is that it uses nonces instead of timestamps. The key establishment properties achieved are the same as for Mechanism 5. However, like Protocol 3.7, inclusion of both nonces in messages 2 and 3 binds the protocol messages together. In contrast, an adversary could interleave two runs of Protocol 3.21 so that A and B do not see matching conversations.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : \{N_A, N_B, ID_B, F_{AB}\}_{K_{AB}}$
3. $B \rightarrow A : \{N_B, N_A, F_{BA}\}_{K_{AB}}$

Protocol 3.22: ISO/IEC 11770-2 Key Establishment Mechanism 6

As with the authentication protocols in Sect. 3.2.3, in all of the above four protocols, the inclusion of the identity of B in the encrypted message from A is optional. Furthermore, in Protocol 3.21, the inclusion of the identity of A in the encrypted message from B is also optional.

3.3.5 Comparison of Server-Less Protocols

Table 3.4 summarises the major properties of the 10 server-less protocols described earlier. The last five columns summarise the goals that these protocols meet. For these goals, an entry (*) indicates that the goal is claimed for the protocol but fails due to an attack. From the table, we see that all the protocols except the Andrew secure RPC protocol and ISO/IEC 11770-2 Mechanism 2 meet the good key goal. Further, the revised version of the Andrew protocol suggested by Burrows *et al.* achieves the good key goal but fails to meet the key confirmation goal.

Table 3.4: Summary of major properties of specific server-less protocols

Properties → ↓ *Protocol*	No. of passes	Key control	Key freshness	Key auth.	Key conf.	Attack
Andrew (3.13)	4	*B*	No (*)	Yes	No	Yes
BAN–Andrew (3.14)	4	*B*	Yes	Yes	No (*)	Yes
Janson–Tsudik (3.15)	3	*B*	Yes	Yes	Yes	No
Boyd (3.16)	2	*A*/*B*	Yes	Yes	No	No
11770-2 Mech. 1 (3.17)	1	*A*	Yes	Yes	*B*	No
11770-2 Mech. 2 (3.18)	1	*A*	No	Yes	No	No
11770-2 Mech. 3 (3.19)	1	*A*	Yes	Yes	*B*	No
11770-2 Mech. 4 (3.20)	2	*B*	Yes	Yes	*B*	No
11770-2 Mech. 5 (3.21)	2	*A*/*B*	Yes	Yes	*A*	No
11770-2 Mech. 6 (3.22)	3	*A*/*B*	Yes	Yes	*A*	No

Table 3.4 indicates that key confirmation may be obtained by one principal in most of the ISO mechanisms. This is because when a key is received from the partner, the recipient knows that the sender is in possession of the key. It is interesting to note that the ISO/IEC 11770-2 standard [381] indicates that none of these protocols provides key confirmation. This may be because mutual key confirmation is expected. The standard indicates that key confirmation may be achieved by sending a time-varying parameter encrypted with the session key. Note that Mechanism 1 allows *B* to know that *A* sent the timestamp (assuming that reflection attacks are prevented) and so *B* has assurance that *A* has the ability to calculate the session key.

3.4 Server-Based Key Establishment

There exist numerous examples of server-based protocols in the literature. Most of these are key transport or key agreement protocols in which the server, or one or both

users, has the responsibility for key generation. Hybrid protocols in which all three of them share the responsibility for key generation are less common and not as well known as the other two classes of protocols.

An important consideration in the design of server-based key transport protocols is who generates the session key. Many protocol designers implicitly assume that the users are not capable of generating good-quality session keys, leaving this task for the server. However, this dependence is not always necessary, as may be seen in published protocols where users, not the server, choose a session key.

Table 3.5 gives additional notation used in this section.

Table 3.5: Additional notation used for server-based protocols

A and B	Two users wishing to establish a session key
S	The server
K_{AS}, K_{BS}	Long-term keys initially shared by A and S, and by B and S
K_{AB}	Session key to be shared by A and B

3.4.1 Needham–Schroeder Shared Key Protocol

The famous protocol proposed by Needham and Schroeder [581] in 1978 is shown as Protocol 3.23. As discussed in Chap. 1, this protocol achieves the good key property with respect to A but not B. The second message encrypted with A's shared key K_{AS} includes both A's nonce and B's identity, assuring A of session key freshness and key authentication, respectively.

1. $A \rightarrow S: \ ID_A, ID_B, N_A$
2. $S \rightarrow A: \ \{N_A, ID_B, K_{AB}, \{K_{AB}, ID_A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B: \ \{K_{AB}, ID_A\}_{K_{BS}}$
4. $B \rightarrow A: \ \{N_B\}_{K_{AB}}$
5. $A \rightarrow B: \ \{N_B - 1\}_{K_{AB}}$

Protocol 3.23: Needham–Schroeder shared key protocol

For B the situation is slightly different: he decrypts the encrypted message relayed by A to learn the session key value and then carries out a nonce handshake with A to be sure that the message is not a replay. However, the handshake can be easily subverted since an adversary can be expected to know the value of an old session key. This weakness of the protocol was originally pointed out by Denning and

Sacco [240]. In their attack an intruder uses a compromised session key to masquerade as A to B. To overcome the attack, Denning and Sacco suggested Protocol 3.24 as a solution, using timestamps to allow verification of key freshness.

1. $A \to S: \ ID_A, ID_B$
2. $S \to A: \ \{ID_B, K_{AB}, T_S, \{ID_A, K_{AB}, T_S\}_{K_{BS}}\}_{K_{AS}}$
3. $A \to B: \ \{ID_A, K_{AB}, T_S\}_{K_{BS}}$

Protocol 3.24: Denning–Sacco protocol

An attack of Chevalier and Vigneron [202] shows that it is essential that *all* fields within encrypted messages are checked. In Attack 3.7, an adversary I starts a run of the protocol as B, and intercepts the reply sent by S. Now, suppose that the implementation is such that B does not distinguish between the timestamp T_S and the concatenated field $T_S, \{ID_B, K_{AB}, T_S\}_{K_{AS}}$. Then I can simply reuse the message sent by S to masquerade as A in a new run with responder B. The result is that B does not achieve liveness of A. Note the attack does not affect key establishment properties.

1. $I_B \to S: \ ID_B, ID_A$
2. $S \to I_B: \ \{ID_A, K_{AB}, T_S, \{ID_B, K_{AB}, T_S\}_{K_{AS}}\}_{K_{BS}}$

$3'$. $I_A \to B: \ \{ID_A, K_{AB}, T_S, \{ID_B, K_{AB}, T_S\}_{K_{AS}}\}_{K_{BS}}$

Attack 3.7: Chevalier–Vigneron attack on Denning–Sacco protocol

Bauer *et al.* [65] discussed the vulnerability of the Needham–Schroeder protocol to the compromise of A's long-term key: an intruder who learns the long-term key of A can impersonate A (as in the Denning–Sacco attack) even after the compromise is detected and the long-term key replaced. They suggested a solution without using timestamps shown as Protocol 3.25. The protocol they proposed is essentially symmetric with respect to A and B: each of them sends a nonce to S in plaintext, and S returns the nonces in separate messages for A and B.

1. $A \to B: \ ID_A, N_A$
2. $B \to S: \ ID_A, N_A, ID_B, N_B$
3. $S \to B: \ \{K_{AB}, ID_A, N_B\}_{K_{BS}}, \{K_{AB}, ID_B, N_A\}_{K_{AS}}$
4. $B \to A: \ \{K_{AB}, ID_B, N_A\}_{K_{AS}}$

Protocol 3.25: Bauer–Berson–Feiertag protocol

Buchholtz [166] proposed an attack on Protocol 3.25, shown as Attack 3.8. The outcome of the attack is that B has completed a protocol run using nonce N_B with initiator A. Similarly, A has completed a protocol run using nonce N'_A with initiator B. However, there is a single run of server S with the nonces N'_A and N_B. Although the attack violates protocol goals regarding agreement on roles and exchanged values, it does not affect the key establishment properties.

1. $A \to B: ID_A, N_A$
2. $B \to I_S: ID_A, N_A, ID_B, N_B$
$1''$. $B \to A: ID_B, N'_B$
$2''$. $A \to I_S: ID_B, N'_B, ID_A, N'_A$
$2'$. $I_B \to S: ID_A, N'_A, ID_B, N_B$
3. $S \to I_B: \{K_{AB}, ID_A, N_B\}_{K_{BS}}, \{K_{AB}, ID_B, N'_A\}_{K_{AS}}$
$3'$. $I_S \to B: \{K_{AB}, ID_A, N_B\}_{K_{BS}}, anything$
$3''$. $I_S \to A: \{K_{AB}, ID_B, N'_A\}_{K_{AS}}, anything$

Attack 3.8: Buchholtz's attack on Bauer–Berson–Feiertag protocol

3.4.2 Otway–Rees Protocol

The Otway–Rees protocol [597], like Protocol 3.25, provides symmetric assurances of freshness. The message flows are shown as Protocol 3.26, where M is a second nonce generated by A. This protocol is susceptible to a typing attack, as described in Sect. 1.4.7. The attack described there is due to Boyd [128], subsequently rediscovered by Clark and Jacob [216].

1. $A \to B: M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}$
2. $B \to S: M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}, \{N_B, M, ID_A, ID_B\}_{K_{BS}}$
3. $S \to B: M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \to A: M, \{N_A, K_{AB}\}_{K_{AS}}$

Protocol 3.26: Otway–Rees protocol

Consider exactly what actions are required of S upon receiving message 2 in the protocol. There are essentially two possibilities:

A1. S checks that the values obtained by decrypting the fields M, ID_A and ID_B in the two encrypted parts match.

A2. S checks that the plaintext versions of (M, ID_A, ID_B) match the values obtained by decrypting the fields M, ID_A and ID_B in the two encrypted parts.

An attack of Boyd and Mao [138] shows that it is essential that the plaintext versions of M, A and B are checked. Without this, the session key can turn out to be shared between A and an intruder, allowing the intruder to masquerade as B as shown in Attack 3.9. Suppose A starts a protocol run with B. An intruder C who wishes to impersonate B chooses a nonce N_C and substitutes the message shown in Attack 3.9 in place of the original one.

$2'.\ C_B \rightarrow S : M, ID_A, ID_C, \{N_A, M, ID_A, ID_B\}_{K_{AS}}, \{N_C, M, ID_A, ID_B\}_{K_{CS}}$

Attack 3.9: Attack on Otway–Rees protocol without plaintext checking

If we assume that S does only the checking specified in A1, the message will be found correct by S and so S will encrypt K_{AB} with the key K_{CS} in message $3'$. It is clear that this attack violates the goal of implicit key authentication: A believes the key is shared with B, whereas in fact it is shared with C. Note that the attack is easily prevented if S does the checking specified in A2. Whether this is a valid attack on the protocol depends on the action taken by S, something (unfortunately) not clearly specified in the protocol description.

In Protocol 3.26 the encrypted message received by a user from the server does not include a concrete field for the identity of the other user to whom S intends to make the key known. What indication, then, does a user have about who else the session key is shared with? The answer to this question can be found by examining a simplified version of the protocol due to Burrows *et al.* shown as Protocol 3.27. In the simplified version of the protocol, the nonce N_B is sent unencrypted in message 2.

1. $A \rightarrow B : M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}$
2. $B \rightarrow S : M, ID_A, ID_B, \{N_A, M, ID_A, ID_B\}_{K_{AS}}, N_B, \{M, ID_A, ID_B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

Protocol 3.27: Otway–Rees protocol modified by Burrows *et al.*

Protocol 3.27 was shown to be flawed by Boyd and Mao. Consider an attacker C who has obtained the encrypted message $\{M, ID_C, ID_B\}_{K_{BS}}$ by engaging in a previous legitimate run of the protocol with B. To attack the protocol, C starts a run with B and masquerades as A by capturing message 2 and modifying it, replacing cleartext identifier A by C and $\{M, ID_A, ID_B\}_{K_{BS}}$ by $\{M, ID_C, ID_B\}_{K_{BS}}$. The attacking run proceeds as shown in Attack 3.10.

The result of the attack is that B believes that the session key is shared with A, whereas in fact it is shared with C. Notice that the above attack is not applicable to

1. $C_A \rightarrow B : M, ID_A, ID_B, \{N_C, M, ID_C, ID_B\}_{K_{CS}}$
2. $B \rightarrow C_S : M, ID_A, ID_B, \{N_C, M, ID_C, ID_B\}_{K_{CS}}, N_B, \{M, ID_A, ID_B\}_{K_{BS}}$
2′. $C_B \rightarrow S : M, ID_C, ID_B, \{N_C, M, ID_C, ID_B\}_{K_{CS}}, N_B, \{M, ID_C, ID_B\}_{K_{BS}}$
3. $S \rightarrow B : \quad M, \{N_C, K_{CB}\}_{K_{CS}}, \{N_B, K_{CB}\}_{K_{BS}}$
4. $B \rightarrow C_A : M, \{N_C, K_{CB}\}_{K_{CS}}$

Attack 3.10: Attack on Otway–Rees protocol modified by Burrows *et al.*

the original Otway–Rees protocol where the nonce N_B is cryptographically bound to the identity A by encryption in message 2. As a result of this binding, B can rely on the nonce N_B in message 3 to infer that the key is shared with A. The attack on the simplified protocol is possible because it removes the binding. The treatment of nonces in the Otway–Rees protocol was discussed by van Oorschot [593] and subsequently by Abadi and Needham [6]. The latter authors suggested an alternative protocol where the nonce N_B (as well as the nonce N_A) is sent unencrypted in message 2 and yet the above attack is prevented. The main idea is that in Protocol 3.28 the encrypted message received by each user includes the identity of the other user to whom S makes the session key known.

1. $A \rightarrow B : ID_A, ID_B, N_A$
2. $B \rightarrow S : ID_A, ID_B, N_A, N_B$
3. $S \rightarrow B : \{N_A, ID_A, ID_B, K_{AB}\}_{K_{AS}}, \{N_B, ID_A, ID_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : \{N_A, ID_A, ID_B, K_{AB}\}_{K_{AS}}$

Protocol 3.28: Otway–Rees protocol modified by Abadi and Needham

This idea was first discussed in a preliminary draft of Boyd and Mao's paper [137] and subsequently by van Oorschot [593]. With this modification, it is easy to check that the previous attack no longer works. Note that Protocol 3.28 is similar in structure and goal to the protocol of Bauer *et al.* (Protocol 3.25). Like the original Otway–Rees protocol it provides key authentication and key freshness assurances, although it differs with respect to goals for entity authentication. In Protocol 3.26, A has assurance that B is alive: when A receives message 4, she knows that B must have sent message 2 recently. In Protocol 3.28, A does not achieve liveness of B.

3.4.3 Kerberos Protocol

The Kerberos software system was developed at MIT to protect the network services provided as part of project Athena. It is one of the *de facto* standards for authentication on computer networks. Kerberos uses as its building block a key establishment protocol based on the Needham–Schroeder protocol but with timestamps instead of

challenge–response, following Denning and Sacco's suggestion. A good overview of the current version of the Kerberos protocol, known as Version 5, can be found in a paper by Neuman and Ts'o [582]. The Version 5 protocol has evolved from the Version 4 protocol; it addresses several shortcomings of the Version 4 protocol including some potential security weaknesses.

Differences between Kerberos Versions 4 and 5 are described by Kohl *et al.* [441]. One weakness of the Version 4 protocol is that the encryption method used does not provide adequate integrity protection for encrypted messages, even though the protocol was specifically designed with this requirement in mind. Version 4 makes use of a non-standard mode of DES known as *plaintext cipher-block chaining* (PCBC) mode with the property that errors in the decrypted ciphertext propagate to all successive blocks of plaintext. However, as pointed out by Kohl [440], PCBC encryption is susceptible to a block-swapping attack which allows a partially garbled message to be accepted by the receiver. Kerberos Version 5 uses standard CBC encryption and embeds a checksum in the message before encryption to provide sufficient integrity protection.

The basic Kerberos protocol involves three parties: a client which desires to use some service, an application server which provides a service, and an authentication server (AS) which is contacted by the client before attempting to access the application server. We will use the notations A, B and S below to denote the Kerberos terms client, application server and AS, respectively. The client and the server do not initially share a key between themselves, but they do share a key with S. For the sake of clarity, Protocol 3.29 shows only those message fields that are critical to security; details of other message fields can be found in RFC 1510 [439].

1. $A \rightarrow S: \ ID_A, ID_B, N_A$
2. $S \rightarrow A: \ \{K_{AB}, ID_B, L, N_A, \ldots\}_{K_{AS}}, \{K_{AB}, ID_A, L, \ldots\}_{K_{BS}}$
3. $A \rightarrow B: \ \{ID_A, T_A\}_{K_{AB}}, \{K_{AB}, ID_A, L, \ldots\}_{K_{BS}}$

Protocol 3.29: Basic Kerberos protocol

In the Kerberos papers, the protocol message fields that are encrypted with K_{BS} and K_{AB} are referred to as the *ticket* and the *authenticator*, respectively. Among other data, the ticket contains the session key generated by S that will be used by the client and server, the client's identity, and an expiration time L after which the session key is no longer valid. The authenticator contains the client's identity and a timestamp from the client's clock. If the timestamp is successfully verified by the application server, then the server obtains assurance that the client possesses the session key contained in the ticket.

The basic Kerberos protocol allows an optional fourth message shown in Protocol 3.30, in which the server returns the client's timestamp along with other optional information, all encrypted using the session key.

4. $B \rightarrow A : \{T_A, \ldots\}_{K_{AB}}$

Protocol 3.30: Optional Kerberos message to complete mutual authentication

The Kerberos protocol has key establishment as well as entity authentication as its goal. From A's viewpoint the protocol provides the good key property since A can be confident that the key is fresh and known only to herself and B. From B's viewpoint the protocol provides only key authentication since the ticket does not contain any information from which B can be confident that the key is fresh. However, the freshness of the key is judged by a different measure. To ensure key freshness from B's viewpoint, the protocol relies on the expiration time contained in the ticket, rather than sending with the key a quantity known to be new. As long as the ticket has not expired, B can still be confident that the session key is safe, even if it was used in a previous session with A. The use of a ticket without an absolute freshness indicator has a useful aspect. It makes it possible for a client to cache the ticket from the server so that a session key may be *re-established* directly with B without the intervention of S. (Protocols that use tickets to re-establish session keys are often known as repeated authentication protocols in the literature.)

As for the authentication properties achieved by the protocol, the combination of authenticator and ticket in the third message provides strong entity authentication of A to B. The fourth message, if present, provides entity authentication of B to A.

3.4.4 ISO/IEC 11770-2 Server-Based Protocols

The server-less protocols in the international standard ISO/IEC 11770 Part 2 [381] were described in Sect. 3.3.4. Here we discuss the seven server-based protocols in that standard. In four of these the server chooses the session key and acts as a key distribution centre. In the other three the session key is chosen by either A or B and the server acts as a *key translation centre* to make that key available to the other party.

The first version of the ISO/IEC 11770-2 standard was published in 1998. From 2004 a number of attacks were found on several of the server-based protocols. A second edition of the standard was published in 2008. There were further attacks found on some of the protocols in the second edition, although currently all known attacks can be avoided by taking appropriate precautions as we explain below. We first mention the two protocols for which no attack was found.

Key Establishment Mechanism 7. The server simply encrypts and sends K_{AB} to both parties including the identity of the peer entity. This is similar to the Bauer–Berson–Feiertag protocol (Protocol 3.25) but the lack of nonces means that neither A nor B gains key freshness.

Key Establishment Mechanism 10. As shown in Protocol 3.31, principal A first sends an encrypted request message to S which checks its authenticity (including checking the freshness of the timestamp or sequence number). This provides the rather unusual feature that only authentic parties are able to request new keys. S

replies by sending the key to both principals after encrypting it together with the identity of the peer entity and a timestamp or counter.

1. $A \rightarrow S: \{T_A, ID_B\}_{K_{AS}}$
2. $S \rightarrow A: \{T_S, K_{AB}, ID_B\}_{K_{AS}}$
3. $S \rightarrow B: \{T'_S, K_{AB}, ID_A\}_{K_{BS}}$

Protocol 3.31: ISO/IEC 11770-2 Key Establishment Mechanism 10

In Key Establishment Mechanism 8, shown in Protocol 3.32, principal A sends a nonce to S and the key is returned encrypted for A together with A's nonce and the identity of B. In addition the key is encrypted for B together with a timestamp or counter and the identity of principal A.

1. $A \rightarrow S: N_A, ID_B$
2. $S \rightarrow A: \{N_A, K_{AB}, ID_B\}_{K_{AS}}, \{T_S, K_{AB}, ID_A\}_{K_{BS}}$
3. $A \rightarrow B: \{T_S, K_{AB}, ID_A\}_{K_{BS}}$

Protocol 3.32: ISO/IEC 11770-2 Key Establishment Mechanism 8

Chen and Mitchell [197] found a typing attack on Protocol 3.32 using a parsing ambiguity. Suppose that a malicious principal's identity C is equal to the concatenation of bit 0 with A's identity, that is, $C = (0, A)$. The attack proceeds as shown in Attack 3.11. The result of the attack is that B believes the value $(K_{CB}, 0)$ is a key to

1. $C \rightarrow S: \quad N_C, ID_B$
2. $S \rightarrow C: \quad \{N_C, K_{CB}, ID_B\}_{K_{CS}}, \{T_S, K_{CB}, ID_C\}_{K_{BS}}$

$3'$. $C_A \rightarrow B: \{T_S, (K_{CB}, 0), ID_A\}_{K_{BS}}$

Attack 3.11: Chen–Mitchell attack on ISO/IEC 11770-2 Key Establishment Mechanism 8

be shared with A, although it is known to C. One way to prevent Attack 3.11 is to ensure that parsing of message components is unambiguous and this was required in a corrigendum to the standard in 2009.

Key Establishment Mechanism 8 was one of several protocols from ISO/IEC 11770-2 attacked by Cremers and Horvat [229] in their analysis using the Scyther tool. Their attack assumes an adversarial principal taking roles both as a server S, and

as a user A or B. Similar attacks also apply against Key Establishment Mechanisms 9, 12 and 13. The attacks are not prevented by precautions against type checking but a defence is to prevent principals from using the same key when acting in different roles: a principal which can both be a server and have a role as initiator should have two independent keys to use in each role. This precaution was already mandated for related protocols in the ISO/IEC 9798-2 protocols (see Technical Corrigendum 3 to ISO/IEC 9798-2, February 2013).

In Key Establishment Mechanism 9 both principals send their nonces to S and they are returned with the encrypted key and the identity of the peer entity. This protocol is identical to Protocol 3.25 except that it also provides key confirmation through an additional exchange using the session key. An essentially identical protocol is also included in ISO/IEC 9798 Part 2 [380] where it is called *five-pass authentication*.

The first of the protocols that uses S as a key translation centre, called Key Establishment Mechanism 11, does not provide key freshness. Nevertheless, the standard claims that it provides key authentication which Cremers and Horvat [229] showed is not correct if the principals can be tricked into accepting a principal identity as a key value. Again, this kind of attack can be prevented by labelling protocol fields with their type.

In comparison with Key Establishment Mechanism 11, Mechanism 12 is similar but now adds a nonce for A to check that the key was received by S and a timestamp for B to check freshness. The session key, K_{AB}, is chosen by A. Protocol 3.33 shows the version of the protocol from the first (1998) edition of the 11770-2 standard.

1. $A \rightarrow S:\ \{N_A, ID_B, K_{AB}\}_{K_{AS}}$
2. $S \rightarrow A:\ \{N_A, ID_B\}_{K_{AS}}, \{T_S, K_{AB}, ID_A\}_{K_{BS}}$
3. $A \rightarrow B:\ \{T_S, K_{AB}, ID_A\}_{K_{BS}}$

Protocol 3.33: ISO/IEC 11770-2 (1998) Key Establishment Mechanism 12

An optional handshake for entity authentication and key confirmation is also specified. It is interesting to compare this protocol with the wide-mouthed-frog protocol (Protocol 3.36) below. Although there are distinct similarities, the asymmetry in Protocol 3.33 prevents the reflection attack described on Protocol 3.36.

Unfortunately, Protocol 3.33 was shown to be flawed by Cheng and Comley [200] who pointed out two attacks. In Attack 3.12, the adversary I masquerades as A by replaying the first message from a previous run of the protocol, containing an old key K'_{AB} used by A and B. I intercepts the reply sent by S to A and forwards the encrypted part intended for B unchanged, thereby forcing B to use the old key K'_{AB}. Attack 3.12 works because the nonce N_A cannot be checked for freshness by S. In fact the 1998 standard allowed any *time-varying parameter* to be used in place of N_A, and one way to avoid the attack is to replace N_A by a timestamp so that the replay can be detected by S.

1. $I_A \rightarrow S: \{N_A, ID_B, K'_{AB}\}_{K_{AS}}$
2. $S \rightarrow I_A: \{N_A, ID_B\}_{K_{AS}}, \{T_S, K'_{AB}, ID_A\}_{K_{BS}}$
3. $I_A \rightarrow B: \{T_S, K'_{AB}, ID_A\}_{K_{BS}}$

Attack 3.12: Replay attack on Protocol 3.33

Attack 3.13, also found by Cheng and Comley [200], is an example of a typing attack which allows a malicious principal C to masquerade as principal B to A. To perpetrate the attack, C sends an encrypted request to S as if it intends to send principal A a session key equal to B's identity. C replays the second encrypted part of the message sent out by S back to S to start another run of the protocol while masquerading as B. C then completes the rest of the protocol as if it were B. The result of the attack is that A accepts the identity of C as a session key with B.

1. $C \rightarrow S: \{N_C, ID_B, ID_A\}_{K_{CS}}$
2. $S \rightarrow C: \{N_A, ID_B\}_{K_{CS}}, \{T_S, ID_A, ID_C\}_{K_{BS}}$
3. *Omitted*.

$1'$. $C_B \rightarrow S: \{T_S, ID_A, ID_C\}_{K_{BS}}$
$2'$. $S \rightarrow C_B: \{T_S, ID_A\}_{K_{BS}}, \{T'_S, ID_C, ID_B\}_{K_{AS}}$
$3'$. $C_B \rightarrow A: \{T'_S, ID_C, ID_B\}_{K_{AS}}$

Attack 3.13: Typing attack on Protocol 3.33

Cheng and Comley proposed a modified protocol, shown as Protocol 3.34, which avoids these attacks. Unfortunately, Protocol 3.34 was itself found to be vulnerable

1. $A \rightarrow S: \{N_A, ID_B, K_{AB}\}_{K_{AS}}$
2. $S \rightarrow A: \{N_A, ID_B, \{T_S, K_{AB}, ID_A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \rightarrow B: \{T_S, K_{AB}, ID_A\}_{K_{BS}}$

Protocol 3.34: Key Establishment Mechanism 12 modified by Cheng and Comley

to a typing attack by Mathuria and Sriram [525], as shown in Attack 3.14. This attack assumes that A cannot differentiate a random session key from the encrypted value $\{T_S, K_{AB}, A\}_{K_{BS}}$.

The revised version of ISO/IEC 11770-2 from 2008 (including Technical Corrigendum 1) makes the following changes to Key Establishment Mechanism 12 in comparison with Protocol 3.33.

1. $A \rightarrow I_S : \{N_A, ID_B, K_{AB}\}_{K_{AS}}$
2. $I_S \rightarrow A : \{N_A, ID_B, K_{AB}\}_{K_{AS}}$
3. $A \rightarrow I_B : K_{AB}$

Attack 3.14: Attack on Cheng and Comley's Protocol 3.34

1. The time-variant parameter, shown as N_A in Protocol 3.33, must be either a time-stamp or a counter. This prevents Attack 3.12.
2. Each message includes a message identifying number. This prevents messages being replayed in the 'wrong position' such as in Attack 3.13.
3. Contatenation must be implemented in such a way as to ensure that there is a unique parsing of messages. This prevents parsing ambiguity attacks [197].

Even with all these precautions an attack was still found by Cremers and Horvat [229] as mentioned above. In this *role mixup attack* [61] the attacking principal plays the roles of the server and a normal user. This can be prevented by ensuring that different keys are used in different roles.

The final protocol in ISO/IEC 11770-2, Key Establishment Mechanism 13, is shown as Protocol 3.35. This time both parties send their nonces to S, while K_{AB} is chosen by B. Mathuria and Sriram [525] found a typing attack on Protocol 3.35 too, but Technical Corrigendum 1 to ISO/IEC 11770-2, September 2009, demands that there is no ambiguity in parsing concatenated messages, thus ruling out such attacks.

1. $A \rightarrow B : N_A$
2. $B \rightarrow S : \{N_B, N_A, ID_A, K_{AB}\}_{K_{BS}}$
3. $S \rightarrow B : \{N_B, ID_A\}_{K_{BS}}, \{N_A, K_{AB}, ID_B\}_{K_{AS}}$
4. $B \rightarrow A : \{N_A, K_{AB}, ID_B\}_{K_{AS}}$

Protocol 3.35: ISO/IEC 11770-2 Key Establishment Mechanism 13

In this section we have seen that the server-based protocols in the ISO/IEC 11770-2 standard have been the subject of several attacks. Nevertheless, these attacks all have fixes and the protocols can still be considered secure if implemented carefully. As a result of their analysis of the related protocols in ISO/IEC 9798-2, Basin *et al.* [61] propose two principles for security protocol designs, which they suggested can complement earlier principles [6].

Position tagging. Messages should include information about the protocol they are for and their position in that protocol.

Inclusion of principals and their roles. Messages should include the identity of all relevant principals and their roles.

3.4.5 Wide-Mouthed-Frog Protocol

Numerous server-based key transport protocols assume that users trust only a server to choose the key for a session. The wide-mouthed-frog protocol, due to Burrows *et al.* [171], is intended for environments where one user trusts the other to choose the key. The server simply makes the key chosen by one user available to the other. The message flows are shown in Protocol 3.36, where T_A and T_S denote timestamps from the local clocks of A and S, respectively.

1. $A \rightarrow S : A, \{T_A, ID_B, K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : \{T_S, ID_A, K_{AB}\}_{K_{BS}}$

Protocol 3.36: Wide-mouthed-frog protocol

The intention is that the timestamps in messages 1 and 2 provide freshness assurances to S and B, respectively. However, this may not provide adequate protection as shown by Attack 3.15. Suppose I is an intruder who has recorded one run of the protocol. I replays the message sent out by S in the first run back to S to start a second run of the protocol while masquerading as B. This would cause S to send $\{T'_S, ID_B, K_{AB}\}_{K_{AS}}$, where T'_S is a new timestamp. Again I replays this message to S to start a third run of the protocol, this time masquerading as A. The intruder continues to execute new runs of the protocol, long after the session key is discarded. Therefore I can force B to re-accept the key again simply by allowing him to receive a sufficiently up-to-date message from S containing the key.

$1'.\ I_B \rightarrow S : ID_B, \{T_S, ID_A, K_{AB}\}_{K_{BS}}$
$2'.\ S \rightarrow I_A : \{T'_S, ID_B, K_{AB}\}_{K_{AS}}$
$1''.\ I_A \rightarrow S : A, \{T'_S, ID_B, K_{AB}\}_{K_{AS}}$
$2''.\ S \rightarrow B : \{T''_S, ID_A, K_{AB}\}_{K_{BS}}$

Attack 3.15: Attack on wide-mouthed-frog protocol

Attack 3.15 was discussed by Anderson and Needham [34] and by Clark and Jacob [216]. However, it should be mentioned that in the BAN logic paper [171], where Protocol 3.36 first appeared, it is assumed that each principal will recognise and reject their own messages. This prevents these attacks.

3.4.6 Yahalom Protocol

The Yahalom protocol first appeared in the BAN logic paper [171]. It is frequently used as a benchmark protocol by researchers using formal methods for protocol verification (for example, see Paulson [607]). One reason for this may be that it has a

rather unusual structure. More importantly, it is subject to some subtle attacks, which make it a challenging subject for testing a new technique.

1. $A \rightarrow B: ID_A, N_A$
2. $B \rightarrow S: ID_B, \{ID_A, N_A, N_B\}_{K_{BS}}$
3. $S \rightarrow A: \{ID_B, K_{AB}, N_A, N_B\}_{K_{AS}}, \{ID_A, K_{AB}\}_{K_{BS}}$
4. $A \rightarrow B: \{ID_A, K_{AB}\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Protocol 3.37: Yahalom protocol

Protocol 3.37 provides key authentication to both A and B. Key freshness is more problematic because the positions of A and B differ. Note that A gains key freshness from the first part of message 3. A knows that this part is not a replay from an old protocol run since she knows the message was sent recently. The same cannot be said of the message received by B. In message 4, B has no direct indication of the freshness of the key from the server but infers that the key K_{AB} was used recently by A. Since the server makes N_B available only to the party requested by B, B can be assured that the encryption with K_{AB} must have been formed by A recently. If A acts properly in relaying the message that was encrypted for B by S as part of the current run, then B can be assured of session key freshness. If A misbehaves by relaying a similar encrypted message she has from an old run, then B cannot determine if the key is fresh. It seems clear that both users gain liveness of the other. A knows that B is active as verification of the first part of message 3 implies B sent message 2 recently. Because of the encryption of N_B with K_{AB} in message 4, B is assured that A really knows K_{AB}. Thus it appears that B gains key confirmation and liveness from message 4.

Burrows *et al.* suggested a modified form of the Yahalom protocol, shown in Protocol 3.38. The nonce N_B is sent unencrypted by B in message 2 and is returned by S in the message part encrypted with K_{BS}. The modified form constitutes a typical usage of the challenge–response mechanism.

1. $A \rightarrow B: ID_A, N_A$
2. $B \rightarrow S: ID_B, N_B, \{ID_A, N_A\}_{K_{BS}}$
3. $S \rightarrow A: N_B, \{ID_B, K_{AB}, N_A\}_{K_{AS}}, \{ID_A, K_{AB}, N_B\}_{K_{BS}}$
4. $A \rightarrow B: \{ID_A, K_{AB}, N_B\}_{K_{BS}}, \{N_B\}_{K_{AB}}$

Protocol 3.38: Yahalom protocol modified by Burrows *et al.*

Paulson [607] suggested a protocol closely related to Protocol 3.38. The only difference is that the message encrypted with K_{BS} includes B's identity.

Protocol 3.38 was shown to be flawed by Syverson [703] who pointed out two attacks. Suppose I is an intruder who wishes to attack the protocol. In Attack 3.16 the intruder I starts a protocol run with B while masquerading as A. After receiving the second message, I starts another run with B by sending an arbitrary value N_I concatenated with N_B from the first run. The intruder I is then able to play the role of S in message 4 of the first run by replaying the encrypted component sent out by B in the second run back to B.

---

1. $I_A \rightarrow B:\ ID_A, N_I$
2. $B \rightarrow I_S:\ ID_B, N_B, \{ID_A, N_I\}_{K_{BS}}$

$1'$. $I_A \rightarrow B:\ ID_A, (N_I, N_B)$

$2'$. $B \rightarrow I_S:\ ID_B, N'_B, \{ID_A, N_I, N_B\}_{K_{BS}}$

3. $Omitted$.
4. $I_S \rightarrow B:\ \{ID_A, N_I, N_B\}_{K_{BS}}, \{N_B\}_{N_I}$

Attack 3.16: Syverson's attack on modified Yahalom protocol

Attack 3.16 is an example of a typing attack. The result of the attack is that B will interpret the value N_I as a session key with A. The attack relies on the assumption that nonces can be of arbitrary length (this is termed substituting 'doubled' nonces for nonces by Syverson).

Attack 3.17 shows Syverson's second attack on Protocol 3.38. It begins with the intruder I intercepting an initial message from A to B. On receiving this message, I initiates a new protocol run with A using N_A as challenge, while masquerading as B. The second protocol run proceeds as follows. I modifies the message from A to S, replacing the new nonce N'_A with the old nonce N_A. I also intercepts the message from S to B and simply replays the encrypted components of this message back as encrypted components of message 3 in the first run but with the order of components switched.

---

1. $A \rightarrow I_B:\ ID_A, N_A$

$1'$. $I_B \rightarrow A:\ ID_B, N_A$

$2'$. $A \rightarrow I_S:\ ID_A, N'_A, \{ID_B, N_A\}_{K_{AS}}$

$2''$. $I_A \rightarrow S:\ ID_A, N_A, \{ID_B, N_A\}_{K_{AS}}$

$3'$. $S \rightarrow I_A:\ N_A, \{ID_A, K_{AB}, N_A\}_{K_{BS}}, \{ID_B, K_{AB}, N_A\}_{K_{AS}}$

2. $Omitted$.
3. $I_S \rightarrow A:\ N_I, \{ID_B, K_{AB}, N_A\}_{K_{AS}}, \{ID_A, K_{AB}, N_A\}_{K_{BS}}$

Attack 3.17: Syverson's alternative attack on modified Yahalom protocol

Attack 3.17 shows that A would be wrong to conclude, after a successful run, that B is active. Note that these two attacks are different not only in structure but also in aim. The first attack violates key establishment properties whereas the second attack violates entity authentication properties.

3.4.7 Janson–Tsudik 3PKDP Protocol

The 3PKDP protocol proposed by Janson and Tsudik [394] is a server-based protocol that uses 2PKDP (discussed in Sect. 3.3.2) as a building block. This protocol has two executions of 2PKDP: firstly between A and S (messages 1 to 3), and then between B and S (messages 4 to 6). The final three messages are intended for entity authentication. In Protocol 3.39 the quantities $AUTH$ and $MASK$ are defined as follows.

$$\begin{aligned} AUTH_A &= [N_A, K_{AB}, ID_B]_{K_{AS}} \\ MASK_A &= [\![AUTH_A]\!]_{K_{AS}} \\ AUTH_B &= [N_B, K_{AB}, A]_{K_{BS}} \\ MASK_B &= [\![AUTH_B]\!]_{K_{BS}}. \end{aligned}$$

1. $A \rightarrow S:\ ID_A, ID_B, N_A$
2. $S \rightarrow A:\ AUTH_A, MASK_A \oplus K_{AB}$
3. $A \rightarrow S:\ [N_A, K_{AB}, ID_A]_{K_{AS}}$
4. $B \rightarrow S:\ ID_B, ID_A, N_B$
5. $S \rightarrow B:\ AUTH_B, MASK_B \oplus K_{AB}$
6. $B \rightarrow S:\ [N_B, K_{AB}, ID_B]_{K_{BS}}$
7. $A \rightarrow B:\ ID_A, N'_A$
8. $B \rightarrow A:\ [N'_A, N'_B, ID_B]_{K_{AB}}, N'_B$
9. $A \rightarrow B:\ [N'_A, N'_B, ID_A]_{K_{AB}}$

Protocol 3.39: Janson–Tsudik 3PKDP protocol

Protocol 3.39 achieves goals concerning both key establishment and entity authentication. It achieves the good key goal and the mutual authentication goal. Considering that the entity authentication goal implies the far-end operative property, it is easily seen that the protocol also provides the enhanced goals of mutual belief in the key and key confirmation. One oddity of this protocol is that messages 3 and 6 do not seem to have any useful purpose. All S can hope to learn from these messages is that A and B are really out there, a property that is not crucial to the service provided by the protocol. Protocol 3.40 is a modified version proposed by Janson and Tsudik which omits the above-mentioned messages and routes all communication with S via B. It achieves the same goals as 3PKDP using five messages rather than nine messages.

1. $A \rightarrow B: ID_A, N_A, N'_A$
2. $B \rightarrow S: ID_A, ID_B, N_A, N_B$
3. $S \rightarrow B: AUTH_A, MASK_A \oplus K_{AB}, AUTH_B, MASK_B \oplus K_{AB}$
4. $B \rightarrow A: AUTH_A, MASK_A \oplus K_{AB}, [N'_A, N'_B, ID_B]_{K_{AB}}, N'_B$
5. $A \rightarrow B: [N'_A, N'_B, ID_A]_{K_{AB}}$

Protocol 3.40: Janson–Tsudik optimised 3PKDP protocol

3.4.8 Bellare–Rogaway 3PKD Protocol

Protocol 3.41 was proposed by Bellare and Rogaway [78]. It has key establishment as its goal and is provably secure in the Bellare–Rogaway model. The 3PKD protocol uses two distinct cryptographic transformations: a symmetric encryption algorithm and a MAC. As one possibility, the encryption function can be constructed from a keyed pseudorandom function f_K. Specifically, the encryption of message m using f under a shared key K is computed as the quantity $(r, m \oplus f_K(r))$, where r is a random number.

1. $A \rightarrow B: N_A$
2. $B \rightarrow S: N_A, N_B$
3. $S \rightarrow A: [\![K_{AB}]\!]_{K_{AS}}, [ID_A, ID_B, N_A, [\![K_{AB}]\!]_{K_{AS}}]_{K_{AS}}$
4. $S \rightarrow B: [\![K_{AB}]\!]_{K_{BS}}, [ID_A, ID_B, N_B, [\![K_{AB}]\!]_{K_{BS}}]_{K_{BS}}$

Protocol 3.41: Bellare–Rogaway 3PKD protocol

Protocol 3.41 provides both A and B with assurances of key authentication and key freshness. It is not designed to provide entity authentication or key confirmation. The provably secure style definition has the property that if the session key itself is used to cryptographically protect messages within the protocol, the resulting protocol cannot be considered secure. Thus standard techniques for key confirmation, such as encrypting something using the session key, are not compatible with the security proof of the 3PKD protocol.

3.4.9 Woo–Lam Key Transport Protocol

Woo and Lam [737] proposed a protocol intended to achieve mutual entity authentication as well as key establishment. As shown in Protocol 3.42, principals A and B exchange nonces before contacting the server. This allows them to include both nonces in the encrypted messages sent to S.

Clark and Jacob [217] found an attack on Protocol 3.42 in which a malicious principal B can force A to accept two copies of the same session key as new keys.

1. $A \rightarrow B: N_A$
2. $B \rightarrow A: N_B$
3. $A \rightarrow B: \{ID_A, ID_B, N_A, N_B\}_{K_{AS}}$
4. $B \rightarrow S: \{ID_A, ID_B, N_A, N_B\}_{K_{AS}}, \{ID_A, ID_B, N_A, N_B\}_{K_{BS}}$
5. $S \rightarrow B: \{ID_B, N_A, N_B, K_{AB}\}_{K_{AS}}, \{ID_A, N_A, N_B, K_{AB}\}_{K_{BS}}$
6. $B \rightarrow A: \{ID_B, N_A, N_B, K_{AB}\}_{K_{AS}}, \{N_A, N_B\}_{K_{AB}}$
7. $A \rightarrow B: \{N_B\}_{K_{AB}}$

Protocol 3.42: Woo–Lam key transport protocol

Lowe [502] also found a typing attack which allows the adversary to masquerade as A and obtain the session key accepted by B. Even without these attacks, it is difficult to recommend Protocol 3.42 as a practical protocol in view of the large number of message flows and encryptions required in comparison with the alternatives.

3.4.10 Gong Key Agreement Protocols

Gong [316] designed several protocols to illustrate lower bounds he derived on the numbers of messages and rounds in server-based protocols for key establishment. Because he was not concerned with minimising message lengths, he took a conservative approach to message formats. In the protocols in this section, messages encrypted with key K are of the format

$$\{Sender, Recipient, Client1, Key, Client2, Freshness\ ID\}_K$$

where *Client1* and *Client2* will share *Key*, and *Freshness ID* may be a nonce or a timestamp. Since some of these fields may be identical, there is some redundancy in many of the messages.

Protocol 3.43 is a key agreement protocol since the session key is derived from the information contributed by both A and B. The server makes each party's contribution available to the other but does not itself contribute any information to the session key. The encrypted portions of the protocol messages include timestamps to provide freshness guarantees.

1. $A \rightarrow S: ID_A, ID_B, \{ID_A, ID_S, ID_A, K_1, ID_B, T_A\}_{K_{AS}}$
2. $S \rightarrow B: \{ID_S, ID_B, ID_A, K_1, ID_B, T_S\}_{K_{BS}}$
3. $B \rightarrow S: \{ID_B, ID_S, ID_B, K_2, ID_A, T_B\}_{K_{BS}}$
4. $S \rightarrow A: \{ID_S, ID_A, ID_B, K_2, ID_A, T_S\}_{K_{AS}}$

Protocol 3.43: Gong's timestamp-based protocol

The values K_1 and K_2 are random numbers serving as contributions of A and B respectively to the session key. Both A and B compute the session key as $f(K_1, K_2)$

where f is a one-way function. This protocol provides to both A and B implicit key authentication and key freshness. The timestamps may be replaced by random nonces to provide freshness guarantees. The resulting protocol has one more message as shown in Protocol 3.44.

1. $A \rightarrow B: ID_A, ID_B, N_A$
2. $B \rightarrow S: ID_A, ID_B, N_B, \{ID_B, ID_S, ID_B, K_2, ID_A, N_A\}_{K_{BS}}$
3. $S \rightarrow A: \{ID_S, ID_A, ID_B, K_2, ID_A, N_A\}_{K_{AS}}, N_B$
4. $A \rightarrow S: \{ID_A, ID_S, ID_A, K_1, ID_B, N_B\}_{K_{AS}}$
5. $S \rightarrow B: \{ID_S, ID_B, ID_A, K_1, ID_B, N_B\}_{K_{BS}}$

Protocol 3.44: Gong's nonce-based protocol

Protocols 3.43 and 3.44 may be extended to provide key confirmation; the cost is an extra message. In the first protocol, B could make use of K_{AB} as an encryption key in message 3 to assure A that he really knows K_{AB}. A new fifth message could provide B with similar assurance. In the second protocol, A could make use of K_{AB} as an encryption key in message 4 to assure B that she really knows K_{AB}. A new sixth message could provide A with similar assurance.

In Protocols 3.43 and 3.44 both A and B check the freshness of an element received with the keying material. While this is the usual method for ensuring session key freshness in key transport protocols, it is generally redundant in key agreement protocols. Each user can ensure that the key is fresh by simply ensuring that its keying material is fresh; there is no need for a user to be able to verify that the input received from the other user is fresh. Thus, the timestamps in Protocol 3.43 and the nonces in Protocol 3.44 could be eliminated. Protocol 3.45, due to Gong, makes use of this optimisation. It provides key authentication, key freshness and key confirmation in only five messages. As before, both A and B compute the session key as $f(K_1, K_2)$.

1. $A \rightarrow S: ID_A, ID_B, \{ID_A, ID_S, ID_A, K_1, ID_B\}_{K_{AS}}, N_A$
2. $S \rightarrow B: ID_A, ID_B, \{ID_S, ID_B, ID_A, K_1, ID_B\}_{K_{BS}}, N_A$
3. $B \rightarrow S: \{ID_B, ID_S, ID_B, K_2, ID_A\}_{K_{BS}}, \{ID_B, ID_A, N_A\}_{K_{AB}}, N_B$
4. $S \rightarrow A: \{ID_S, ID_A, ID_B, K_2, ID_A\}_{K_{AS}}, \{ID_B, ID_A, N_A\}_{K_{AB}}, N_B$
5. $A \rightarrow B: \{ID_A, ID_B, N_B\}_{K_{AB}}$

Protocol 3.45: Gong's alternative protocol

3.4.11 Boyd Key Agreement Protocol

Protocol 3.46, proposed by Boyd [131], provides key authentication, key freshness and key confirmation in only four messages. It is a server-based protocol in which both users as well as the server contribute to the key value. The values N_A and N_B are generated by A and B respectively as input to the MAC function determining the session key. Additionally, S generates a value K_S which serves as the MAC key. Both A and B compute the session key as $K_{AB} = \mathsf{MAC}_{K_S}(N_A, N_B)$.

1. $A \rightarrow S: \ ID_A, ID_B, N_A$
2. $S \rightarrow B: \ \{ID_A, ID_B, K_S\}_{K_{AS}}, \{ID_A, ID_B, K_S\}_{K_{BS}}, N_A$
3. $B \rightarrow A: \ \{ID_A, ID_B, K_S\}_{K_{AS}}, [N_A]_{K_{AB}}, N_B$
4. $A \rightarrow B: \ [N_B]_{K_{AB}}$

Protocol 3.46: Boyd key agreement protocol

3.4.12 Gong Hybrid Protocol

Protocol 3.47, due to Gong [313], is an example of a hybrid protocol in which only A and S have an input to the key derivation function. It employs two one-way functions: a function f used for key derivation and a function g used for authentication. These functions need not be distinct. The output of the key derivation function f is divided into three components:

$$f(N_S, N_A, ID_B, K_{BS}) = (K_{AB}, H_A, H_B).$$

The first component is the session key value itself. The second component is sent from A to B to assure the latter that A has the key. The third component provides A with reciprocal assurance. In this protocol B derives freshness by checking an element received with the key, while A derives freshness by generating a fresh input to the session key generation process.

1. $A \rightarrow B: \ ID_A, ID_B, N_A$
2. $B \rightarrow S: \ ID_A, ID_B, N_A, N_B$
3. $S \rightarrow B: \ N_S, f(N_S, N_B, ID_A, K_{BS}) \oplus (K_{AB}, H_A, H_B), g(K_{AB}, H_A, H_B, K_{BS})$
4. $B \rightarrow A: \ N_S, H_B$
5. $A \rightarrow B: \ H_A$

Protocol 3.47: Gong's hybrid protocol

Boyd and Mathuria [140] demonstrated an unusual feature of Protocol 3.47. Suppose A has executed a normal run of the protocol with B, with the derived session key being K'_{AB} and the other two quantities being H'_A and H'_B. Furthermore, suppose that A has also recorded the reply from the server to B and thus is in possession of the value $g(K'_{AB}, H'_A, H'_B, K_{BS})$. Now, A is able to complete the protocol with B as shown in Attack 3.18.

---

3. $S \rightarrow A_B$: $N_S, f(N_S, N_B, ID_A, K_{BS}) \oplus (K_{AB}, H_A, H_B)$,
   $g(K_{AB}, H_A, H_B, K_{BS})$
3′. $A_S \rightarrow B$: $N_S, f(N_S, N_B, ID_A, K_{BS}) \oplus (K'_{AB}, H'_A, H'_B)$,
   $g(K'_{AB}, H'_A, H'_B, K_{BS})$
4. $B \rightarrow A$: N_S, H'_B
5. $A \rightarrow B$: H'_A

Attack 3.18: Insider attack on Protocol 3.47

The insight gained from this attack is that a malicious principal A can force B into accepting an old session key as new. It highlights an assumption that was probably not obvious when the protocol was designed.

Saha and RoyChowdury [644] proposed Protocol 3.48 as an improvement to Protocol 3.47. The general design is similar but it adds B's nonce to the input of the function used for authentication of the key. This allows the weakness of Gong's protocol to be avoided. The session key K_{AB} is chosen by S and the reply sent from S to A is symmetrical to S's reply to B. Another protocol with the same design goals, but allowing A to choose the key, was proposed by the same authors. Both protocols enjoy a formal proof of security in the Bellare–Rogaway model.

---

1. $A \rightarrow B$: ID_A, ID_B, N_A
2. $B \rightarrow S$: ID_A, ID_B, N_A, N_B
3. $S \rightarrow B$: $N_S, f(N_S, N_B, ID_A, K_{BS}) \oplus (K_{AB}, H_A, H_B), g(K_{AB}, H_A, H_B, N_B, K_{BS})$,
   $f(N_S, N_A, ID_B, K_{AS}) \oplus (K_{AB}, H_A, H_B), g(K_{AB}, H_A, H_B, N_A, K_{AS})$
4. $B \rightarrow A$: $N_S, H_B, f(N_S, N_A, ID_B, K_{AS}) \oplus (K_{AB}, H_A, H_B), g(K_{AB}, H_A, H_B, N_A, K_{AS})$
5. $A \rightarrow B$: H_A

Protocol 3.48: Saha–RoyChowdhury protocol

3.4.13 Comparison of Server-Based Protocols

Table 3.6 summarises the major properties of the 22 server-based protocols described earlier. For the goals an entry (*) indicates that the goal is claimed but fails due to at-

tack. From the table, we see that all of the protocols except the Needham–Schroeder protocol and the wide-mouthed-frog protocol achieve the key freshness goal. Further, all of the protocols except the BAN Otway–Rees protocol and the BAN Yahalom protocol achieve the key authentication goal.

Table 3.6: Summary of major properties of specific server-based protocols

Properties → ↓ *Protocol*	No. of passes	Key control	Fresh key	Key auth.	Key conf.	Attack
Needham–Schroeder (3.23)	5	*S*	*A* (*)	*A+B*	*A*	Yes
Denning–Sacco (3.24)	3	*S*	*A+B*	*A+B*	No	No
Bauer–Berson–Feiertag (3.25)	4	*S*	*A+B*	*A+B*	No	No
Otway–Rees (3.26)	4	*S*	*A+B*	*A+B*	No	No
Otway–Rees modified (3.27)	4	*S*	*A+B*	*A* (*)	No	Yes
Otway–Rees modified (3.28)	4	*S*	*A+B*	*A+B*	No	No
Kerberos (3.29)	3	*S*	*A+B*	*A+B*	*B*	No
11770-2 Mechanism 10 (3.31)	3	*S*	*A+B*	*A+B*	No	No
11770-2 Mechanism 12 (3.33)	3	*A*	*A+B*	*A+B*	No	No
11770-2 Mechanism 13 (3.35)	4	*B*	*A+B*	*A+B*	No	No
Wide-mouthed-frog (3.36)	2	*A*	No (*)	Yes	No	Yes
Yahalom (3.37)	4	*S*	*A+B*	*A+B*	*B*	No
BAN Yahalom (3.38)	4	*S*	*A+B*	*A* (*)	No (*)	Yes
3PKDP (3.39)	9	*S*	*A+B*	*A+B*	*A+B*	No
Optimised 3PKDP (3.40)	5	*S*	*A+B*	*A+B*	*A+B*	No
Bellare–Rogaway (3.41)	4	*S*	*A+B*	*A+B*	No	No
Woo–Lam (3.42)	7	*S*	No (*)	*B* (*)	*A+B*	Yes
Gong timestamp (3.43)	4	*A/B*	*A+B*	*A+B*	No	No
Gong nonce-based (3.44)	5	*A/B*	*A+B*	*A+B*	No	No
Gong alternative (3.45)	5	*A/B*	*A+B*	*A+B*	*A+B*	No
Boyd four-pass (3.46)	4	*S/A/B*	*A+B*	*A+B*	*A+B*	No
Gong hybrid (3.47)	5	*S/A*	*A+B*	*A+B*	*A+B*	No

3.5 Key Establishment Using Multiple Servers

Each of the server-based protocols we have examined so far in this chapter has involved three principals: one server and two users. One natural way to generalise this situation is to allow more than two users. Key establishment with multiple users is the subject of Chap. 9. A different generalisation is to use more than one server. There are at least two potential benefits of such an architecture.

- If one or more servers become unavailable, it may still be possible for the users to establish a session key.
- If one or more servers are untrustworthy, users may still be able to establish a good key.

There have been a few concrete proposals for protocols using multiple servers. We examine two of these in this section.

3.5.1 Gong's Multiple Server Protocol

Gong [315] proposed a number of variant protocols all with the same basic structure. A feature of all these protocols is that the users, A and B, choose the keying material while the n servers, $S_1, S_2, \ldots, S_n$, act as key translation centres to allow keying material from one user to be made available to the other. Initially A shares a long-term key $K_{A,i}$ with each server S_i, and similarly B shares $K_{B,i}$ with S_i.

In order to ensure that the correct key can be recovered even if some servers become unavailable, A and B both split up their secrets using a threshold scheme (see Sect. 1.3.6). Specifically, A chooses a secret x and splits it into shares x_1, x_2, $\ldots$, x_n so that x can be recovered from any t shares. Similarly B chooses a secret y and divides it into shares y_1, y_2, $\ldots$, y_n. Protocol 3.49 shows a simplified version of Gong's main protocol. Messages 2 and 3 are repeated for each of the n servers so there are $2n+3$ messages sent in total. On receipt of the translated shares from each server, A is able to recover the secret y of B, and similarly B recovers x. The session key is defined as $K_{AB} = h(x,y)$.

1. $A \rightarrow B:\ ID_A, ID_B, N_A, \{ID_A, ID_B, x_i, cc(x)\}_{K_{A,i}}$
2. $B \rightarrow S_i:\ ID_A, ID_B, N_A, N_B, \{ID_A, ID_B, x_i, cc(x)\}_{K_{A,i}}, \{ID_B, ID_A, y_i, cc(y)\}_{K_{B,i}}$
3. $S_i \rightarrow B:\ \{ID_B, N_A, y_i, cc_i(y)\}_{K_{A,i}}, \{ID_A, N_B, x_i, cc_i(x)\}_{K_{B,i}}$
4. $B \rightarrow A:\ \{ID_B, N_A, y_1, cc_1(y)\}_{K_{A,1}}, \ldots, \{ID_B, N_A, y_n, cc_n(y)\}_{K_{A,n}}, \{N_A\}_{K_{AB}}, N_B$
5. $A \rightarrow B:\ \{N_B\}_{K_{AB}}$

Protocol 3.49: Gong's simplified multi-server protocol

In order to prevent malicious rogue servers from disrupting the protocol, A and B form a *cross-checksum* for all the shares. The cross-checksum for x is $cc(x) =$

$(h(x_1), h(x_2), \ldots, h(x_n))$ where h is a one-way function. The cross-checksum $cc_i(x)$ received from server S_i may or may not be equal to the correct value $cc(x)$. When B receives a checksum $cc_i(x)$ from server S_i he calculates $h(x_j)$ for every x_j received from any other server S_j, and compares the result with that in $cc_i(x)$. If they are the same then S_j is allocated one credit point. When all the checks are complete B retains those shares from the servers with the most credit points. The users require t shares in order to recover the secret. The process will ensure that the correct secret is recovered as long as half of the responding servers are honest and t of them are available.

In Protocol 3.49 an adversary could replay the messages of A or B (or even both) since the server cannot detect replayed requests. However, this does not seem to be a major problem since both parties have assurance that K_{AB} is fresh, from the freshness of their own input. Gong's main protocol [315] differs from Protocol 3.49 by including an initial exchange between A and each server, in which each server delivers a nonce that is returned with the encrypted messages intended for translation. A consequence of this is that servers can detect replayed requests and ignore them, but the cost of this is that the number of messages is increased to $4n+3$.

3.5.2 Chen–Gollmann–Mitchell Protocol

Chen *et al.* [193] designed two protocols using multiple servers. A significant difference from Protocol 3.49 is that in both of their protocols the servers, instead of the users, choose the keying material. Furthermore, instead of sharing one secret, each server chooses an independent secret value. Both users employ a cross-checksum to decide which servers have given valid inputs and all of these are used in the definition of the session key. The parallel protocol of Chen *et al.* is shown as Protocol 3.50. Messages 2 and 3 are repeated n times between B and each server S_i so there are $2n+4$ messages in total.

1. $A \rightarrow B:\ ID_A, ID_B, N_A$
2. $B \rightarrow S_i:\ ID_A, ID_B, N_A, N_B$
3. $S_i \rightarrow B:\ \{ID_B, N_A, K_i\}_{K_{A,i}}, \{ID_A, N_B, K_i\}_{K_{B,i}}$
4. $B \rightarrow A:\ \{ID_B, N_A, K_1\}_{K_{A,1}}, \ldots, \{ID_B, N_A, K_n\}_{K_{A,n}}, cc_B(1), \ldots, cc_B(n)$
5. $A \rightarrow B:\ cc_A(1), \ldots, cc_A(n), \{ID_B, N_B, N'_A\}$
6. $B \rightarrow A:\ \{ID_A, N'_A, N_B\}$

Protocol 3.50: Chen–Gollmann–Mitchell multi-server protocol

The cross-checksum used in Protocol 3.50 is quite different from that used in Protocol 3.49. If B has apparently received all n keys, then

$$cc_B(i) = \{h(K_1), h(K_2), \ldots, h(K_n)\}_{K_i},$$

for all i with $1 \leq i \leq n$, where h is a one-way function. However, if B has not received any message from server S_j then $cc_B(j)$ is simply an error message, and also $h(K_j)$ is replaced by an error message in the calculation of the other $cc_B(i)$ values. On receipt of the checksums $cc_B(1), \ldots, cc_B(n)$ from B, A first decrypts the values and compares them. Some of these may be different if A and B have received some different K_i values, but as long the majority of the servers are honest and operational, the majority of the decrypted values will be the same. The K_i secrets are retained for this majority of i values and the others discarded. The cross-checksums $cc_A(i)$ are then defined in a symmetrical way, except that error messages are inserted either if A did not receive a K_i value, or if B has indicated that he did not receive it.

Once the good keys have been identified, the session key K_{AB} is defined to be the hash of all the good K_i values concatenated. Chen *et al.* showed that as long as at least half of the servers are honest and operational, then an honest A and B pair will accept the same K_i values and hence the same K_{AB}.

Chen *et al.* also proposed a variant of Protocol 3.50, which they called a 'cascade protocol'. The difference is that instead of B making a request to each server by repeating message 2, the request is passed on from server S_1 to server S_2 and so on. The response from each server is also sent on to the next server. Finally server S_n returns all the server responses to B, and the last three messages are the same as in Protocol 3.50. The advantage of the cascade protocol is that the number of messages sent is reduced to $n+5$ since all but one of the responses in message 3 of Protocol 3.50 are no longer required. However, the protocol will only work if all the servers are operational; if only one server fails to cooperate, either maliciously or due to an error, then the protocol will fail.

3.6 Conclusion

To a large extent the problem of key establishment between two parties using symmetric cryptography seems to have been solved. There have been no significant new protocols in recent years. Efficient solutions have evolved which have resisted attacks or even have security proofs. Tables 3.2, 3.4 and 3.6 show that there exist a variety of protocols which should be suitable for most applications requiring these sorts of protocols.

4

Authentication and Key Transport Using Public Key Cryptography

4.1 Introduction

It is generally regarded that there are two main potential advantages of public key techniques over symmetric cryptography. The first is that public key systems allow the straightforward definition of digital signatures, thereby enabling the service of non-repudiation which is so useful in commercial applications. The second is the simplification of key management, because there is no requirement for the online third party that is part of typical protocols based on symmetric cryptography. The first of these advantages is not really our concern in this book since non-repudiation is of limited value in authentication and key establishment. However, the second advantage has led to a great variety of new key establishment protocols since the invention of public key cryptography. In the modern distributed communications environments exemplified by the Internet, public-key-based protocols have become far more important than protocols based on symmetric cryptography.

There are two costs that must be paid in exchange for these benefits. The first one is the high computational cost that comes with all known public key cryptosystems. Despite the advances made in public key cryptosystems and the advantages of elliptic curve cryptology [105], public key algorithms require two or three orders of magnitude more computation than symmetric algorithms. Although computing power on a typical desktop or mobile device is such that a handful of public key operations will not cause a delay of more than a fraction of a second, the overhead for servers of multiple clients and for low-power computing devices is still significant. At the same time increased computing power means that longer key sizes are required so that the cost of public key operations increases. It is therefore essential that designers of public-key-based protocols should minimise the number of public key operations wherever possible. Another issue to be considered is whether the protocol requires more private key operations (signature generations and decryptions) or more public key operations (signature verifications and encryptions). RSA and related algorithms are much more efficient for public key operations than for private key operations, while for most algorithms based on discrete logarithms the opposite is true. Furthermore, although algorithms based on discrete logarithms (including elliptic curve

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_4

algorithms) are more efficient than RSA overall, a protocol that requires mainly public key operations may be much more efficiently implemented using RSA with a small public exponent. Generally it is not a simple task to compare the efficiency of different protocols when implemented using different algorithms.

The second cost of public key cryptography is that public keys still need to be managed. The best solution for a public key infrastructure is still the subject of considerable research. Although public keys need not (indeed usually should not) be kept confidential their integrity must be maintained, normally through use of *certificates* signed by reputable third parties. The question of how to deal with compromised private keys is a tricky one, but the most established solution is to use certificate revocation lists to check whether a public key is still valid, much as a blacklist is checked before accepting a credit card. In the descriptions of protocols given in this chapter we shall ignore the certificates, or any other means, used to ensure that a public key is valid, and simply assume that public keys are available to the parties that need them and are guaranteed to be correct. What *correct* means here is that the claimed owner of the public key is the only entity who is able to use the corresponding private key. Designers and users of public key protocols need to be aware that this is an important simplification that must be addressed in any implementation.

In this chapter we shall not cover the important class of public-key-based key agreement protocols which are the subject of Chap. 5. In the remainder of this section we explain our notation in this chapter and discuss general design principles for public key protocols. Section 4.2 examines public key protocols for entity authentication and Sect. 4.3 covers public key protocols providing key transport.

4.1.1 Notation

The notation used in this chapter is summarised in Table 4.1. In all our protocol descriptions generic algorithms for public key encryption and digital signatures are shown, although in some cases we will note that certain protocols were designed with specific algorithms in mind. We assume that all encryption algorithms provide semantic security and sometimes comment if non-malleability is also required.

Table 4.1: Notation used throughout Chap. 4

$\mathsf{Enc}_X(M)$	Encryption of message M using the public key of principal X
$\mathsf{Sig}_X(M)$	Signature with appendix of message M by principal X
N_X	Random nonce value chosen by principal X
T_X	Timestamp chosen by principal X
$\{M\}_K$	Symmetric encryption of message M with key K

4.1.2 Design Principles for Public Key Protocols

Anderson and Needham [35] proposed a set of what they called *robustness principles* for public-key-based protocols. These can be considered as more specific instances of the general principles for protocol design proposed earlier by Abadi and Needham and which are discussed in Sect. B.4. They form a checklist that can be used by protocol designers to avoid the most common errors. A summary of these principles is shown in Table 4.2. Anderson and Needham gave several examples of potential attacks that can result from ignoring these principles.

Table 4.2: Anderson and Needham's robustness principles for public key protocols

1. Sign before encrypting. If a signature is affixed to encrypted data then one cannot assume that the signer has any knowledge of the data.
2. Be careful how entities are distinguished. If possible avoid using the same key for two different purposes (such as signing and decryption) and be sure to distinguish different runs of the same protocol from each other.
3. Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent.
4. Account for all the bits – how many provide equivocation, redundancy, computational complexity, and so on.
5. Do not assume the secrecy of anybody else's 'secrets' (except possibly those of a certification authority).
6. Do not assume that a message you receive has a particular form (such as g^r for known r) unless you can check this.
7. Be explicit about the security parameters of cryptographic primitives.

Later Syverson [704] questioned the applicability of some of the principles proposed by Anderson and Needham by showing examples when they are not appropriate. Nevertheless, Syverson concluded that the principles are still useful but should be used intelligently and critically by the protocol designer. In this chapter we will see that several protocols ignore the first Anderson–Needham principle, including protocols that are proven secure. Indeed, with suitable precautions, it is known that signing either before or after encrypting can provide suitable security [32]. This is discussed further in Sects. 4.3.1 and 4.3.2.

4.2 Entity Authentication Protocols

Before looking at protocols for key establishment we examine some protocols that achieve only authentication. In this section we examine some prominent examples, in particular those in the international standard ISO/IEC 9798 Part 3 [377]. We discuss

whether they achieve the definition of entity authentication introduced in Definition 13 or the simpler liveness property.

4.2.1 Protocols in ISO/IEC 9798-3

Five protocols in ISO/IEC 9798-3 are designed for authentication between a pair of principals. (Two further server-based protocols are also included whose purpose is to verify the public keys of principals – we do not cover these here.) Two of these protocols are for unilateral authentication of one party to another, and three are for mutual authentication of both parties to each other. Two of the protocols (Protocols 4.2 and 4.5) are also included in the former US standard FIPS 196 [573] which includes some further guidance on the use of optional fields.

Each protocol includes options for various 'text' fields to be included in each message for application-dependent data. According to the standard, a text field may be included in a signature for various reasons including:

- to authenticate any information;
- to add extra redundancy to the signature;
- to provide additional time variant parameters such as timestamps;
- to provide validity information for the protocol in use.

Unsigned text fields may be used for the claimed identity of the message sender which is not otherwise explicitly stated in the protocols. Since they are not part of the basic protocols we shall ignore the optional text fields in the following descriptions.

The first protocol in the standard, shown as Protocol 4.1, consists of a single message from a claimant A to a verifier B. The timestamp T_A is used to provide freshness, or alternatively it may be replaced by a counter. The protocol assures B that A is alive, as well as providing assurance that A is aware of B as her peer entity.

1. $A \rightarrow B : T_A, ID_B, \mathsf{Sig}_A(T_A, ID_B)$

Protocol 4.1: ISO/IEC 9798-3 one-pass unilateral authentication

The second protocol (Protocol 4.2) uses a nonce instead of a timestamp, which is the most obvious difference from Protocol 4.1. The other difference is the inclusion of the random value N_A chosen by A. This field has nothing to do with authentication but is included to ensure that A is not signing a message that has been chosen by B; this could cause problems for A if the signature scheme and signing key used in the protocol are also used in other applications. This protocol provides entity authentication of A to B.

It is interesting that the standard allows the field containing the identity B in message 2 to be omitted, stating that its inclusion 'depends on the environment in which this authentication mechanism is used'. With a weaker definition of entity authentication, having no requirement for knowledge of the peer entity, the omission of this

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, N_B, ID_B, \mathsf{Sig}_A(N_A, N_B, ID_B)$

Protocol 4.2: ISO/IEC 9798-3 two-pass unilateral authentication

field is acceptable. However, if knowledge of the peer entity is required then this field must be included in the signature of message 2. The standard specifically forbids exclusion of the corresponding field in the one-pass Protocol 4.1. We can speculate that the reasoning behind this is that when a timestamp is used B has nothing to connect him to the protocol instance, whereas when B's challenge is returned he can reason that he is 'connected' with A even if A has not indicated this. We refer the reader to Sect. 1.5.3 for more discussion on the possible shades of entity authentication.

The third protocol (Protocol 4.3) is simply the combination of two instances of Protocol 4.1 for one-pass unilateral authentication and as in that protocol the timestamps T_A and T_B may be replaced by counters. This protocol provides mutual entity authentication. Because the messages sent are independent of each other, this protocol can be executed in one round.

1. $A \rightarrow B : T_A, ID_B, \mathsf{Sig}_A(T_A, ID_B)$
2. $B \rightarrow A : T_B, ID_A, \mathsf{Sig}_B(T_B, ID_A)$

Protocol 4.3: ISO/IEC 9798-3 two-pass mutual authentication

Chen and Mitchell [197] showed that a typing attack applies if the optional text fields are included in each message, shown as Protocol 4.4.

1. $A \rightarrow B : T_A, ID_B, \text{text}_1, \mathsf{Sig}_A(T_A, ID_B, \text{text}_1)$
2. $B \rightarrow A : T_B, ID_B, \text{text}_2, \mathsf{Sig}_B(T_B, ID_A, \text{text}_2)$

Protocol 4.4: ISO/IEC 9798-3 two-pass mutual authentication with text fields included

Suppose that a malicious principal's identity C is equal to the concatenation of B's identity with some bit-string x, that is, $ID_C = (ID_B, x)$. The attack proceeds as shown in Attack 4.1.

In addition to the Chen and Mitchell attack, Basin *et al.* [61] found two other attacks: a *role mixup attack* in which the principals play different roles from the ones they intend to play; and a reflection attack in which two instances of the same party are intended to communicate but only one actually participates. Basin *et al.* [61] designed new protocol versions using their two principles of (i) tagging messages

1. $C \rightarrow A: \ T_C, A, \text{text}_1, \text{Sig}_C(T_C, ID_A, \text{text}_1)$
2. $A \rightarrow C: \ T_A, ID_C, \text{text}_2, \text{Sig}_A(T_A, ID_C, \text{text}_2)$
$1'$. $C_A \rightarrow B: T_A, ID_A, \text{text}_3, \text{Sig}_A(T_A, ID_B, \text{text}_3)$
$2'$. $B \rightarrow C_A: T_B, ID_A, \text{text}_4, \text{Sig}_B(T_B, ID_A, \text{text}_4)$

Attack 4.1: Chen–Mitchell attack on Protocol 4.4

to include their protocol and position, and (ii) including in messages the identity of the relevant principals and their roles. (See Sect. 3.4.4 for further discussion on these principles.) Such measures were made mandatory in a 2012 corrigendum to the 9798-3 standard.[1]

The fourth protocol (Protocol 4.5) is an extension of Protocol 4.2, allowing both A and B to use respective nonces, N_A and N_B. The standard again allows the field B in message 2 and the field A in message 3 to be omitted; but the field must be included at least in message 2 if it is desired for B to gain assurance that A is aware of B as her peer entity. The standard specifically forbids exclusion of the corresponding fields in Protocol 4.3 for two-pass mutual authentication.

1. $B \rightarrow A: N_B$
2. $A \rightarrow B: N_A, N_B, ID_B, \text{Sig}_A(N_A, N_B, ID_B)$
3. $B \rightarrow A: N_B, N_A, ID_A, \text{Sig}_B(N_B, N_A, ID_A)$

Protocol 4.5: ISO/IEC 9798-3 three-pass mutual authentication

Blake-Wilson and Menezes [108] proved the security of Protocol 4.5 using the Bellare–Rogaway model described in Chap. 2. This means that the protocol satisfies the matching conversations property. By providing signatures of the other party's identity in messages 2 and 3, the protocol also provides knowledge of the peer entity.

Protocol 4.6 shows an earlier version of Protocol 4.5 that was proposed during the standardisation process. The only difference from Protocol 4.5 is that B chooses and signs a nonce N'_B in the final message, which is different from the nonce N_B used in the first two messages. A probable reason for this choice is that it ensures that B does not have to sign a message that is predictable by A.

Protocol 4.6 is subject to Attack 4.2, which has become known as the 'Canadian attack' because it was publicised by the Canadian team taking part in the standards process. In the attack the adversary C sets up two protocol runs, masquerading as B to A and as A to B. The response from A in the second run can be used by C to complete the first run with B.

The result of the attack is that A completes the protocol apparently with B, whereas in fact the protocol was run with C. In terms of matching conversations

[1] Technical Corrigendum 2 to ISO/IEC 9798-3:2008, March 2012.

1. $B \rightarrow A : N_B$
2. $A \rightarrow B : N_A, N_B, ID_B, \mathsf{Sig}_A(N_A, N_B, ID_B)$
3. $B \rightarrow A : N'_B, N_A, ID_A, \mathsf{Sig}_B(N'_B, N_A, ID_A)$

Protocol 4.6: Early version of ISO/IEC 9798-3 three-pass mutual authentication

1. $C_B \rightarrow A : N_C$
2. $A \rightarrow C_B : N_A, N_C, ID_B, \mathsf{Sig}_A(N_A, N_C, ID_B)$

$1'$. $C_A \rightarrow B : N_A$

$2'$. $B \rightarrow C_A : N_B, N_A, ID_A, \mathsf{Sig}_B(N_B, N_A, ID_A)$

3. $C_B \rightarrow A : N_B, N_A, ID_A, \mathsf{Sig}_B(N_B, N_A, ID_A)$

Attack 4.2: Canadian attack on Protocol 4.6

this is a valid attack, since A and B disagree on the second message. However, A is correct to conclude that B is alive and has indicated awareness of A as his peer entity. Therefore the extensional protocol goals do not seem to be violated. This attack is closely related to Attack 1.4 discussed in Chap. 1. Mitchell and Thomas [558] discuss Attack 4.2. They also considered methods to prevent signatures obtained during similar protocols being abused by an adversary. These methods include use of a protocol identifier in every signed message.

Protocol 4.7 is the final two-party protocol in the standard; it allows authentication to be conducted in parallel between A and B. Thus messages 1 and $1'$ can be sent together, as can 2 and $2'$. As with Protocol 4.5 the standard allows the identity fields B and A, in messages 2 and $2'$ respectively, to be omitted. Once again, omission of these fields means that knowledge of the peer entity cannot be provided, this time in either direction.

1. $A \rightarrow B : N_A$

$1'$. $B \rightarrow A : N_B$

2. $A \rightarrow B : N_A, N_B, ID_B, \mathsf{Sig}_A(N_A, N_B, ID_B)$

$2'$. $B \rightarrow A : N_B, N_A, ID_A, \mathsf{Sig}_B(N_B, N_A, ID_A)$

Protocol 4.7: ISO/IEC 9798-3 two-pass parallel authentication

We note that Protocols 4.5 and 4.7 both link together previous protocol messages in their signed messages; this means that they provide the matching conversations property described in Chap. 1. In this sense it can be argued that they provide more than Protocol 4.3 even though all three protocols provide mutual entity authentication.

4.2.2 Protocols in ISO/IEC 9798-5

The international standard ISO/IEC 9798-5 [382] is devoted to entity authentication mechanisms using zero knowledge techniques. Six classes of protocols are specified, depending mainly on the algebraic setting. The standard specifies protocols based on:

- discrete logarithms in the integers modulo a prime;
- discrete logarithms in the integers modulo a composite;
- discrete logarithms in elliptic curve groups;
- the identity-based setting;
- public key encryption;
- integer factorisation.

The protocols apply zero-knowledge proofs from the literature due to Fiat–Shamir [275], Guillou–Quisquater [335], Schnorr [657], Brandt *et al.* [147] and Girault *et al.* [307]. With one exception, only unilateral authentication is specified for each protocol type. The exception, the integer factorisation setting, has protocols for both unilateral and mutual authentication. These protocols are strongly dependent on specific cryptographic mechanisms and are designed to prove knowledge of private keys corresponding to known public keys.

4.2.3 SPLICE/AS

The protocol known as SPLICE/AS was proposed in 1990 by Yamaguchi *et al.* [749] to provide mutual authentication between a client A and a server B. A number of different papers have progressively found attacks and proposed improvements. The full protocol includes retrieval of certified public keys from an authentication server. Hwang and Chen [370] showed that the original certificate format was flawed, allowing an adversary to alter the apparent public key of either client or server. The authentication part is shown in Protocol 4.8, where L is a lifetime of the message, purported to be used to prevent replay.

1. $A \rightarrow B: ID_A, ID_B, T_A, L, \mathsf{Enc}_B(N_A), \mathsf{Sig}_A(ID_A, T_A, L, \mathsf{Enc}_B(N_A))$
2. $B \rightarrow A: ID_B, ID_A, \mathsf{Enc}_A(ID_B, N_A + 1)$

Protocol 4.8: SPLICE/AS protocol

The protocol is intended to provide mutual entity authentication. The server B checks the signature and timestamp on receipt of the first message in order to authenticate A. When A receives the returned message she checks her nonce in order to authenticate B. However, Protocol 4.8 was attacked by Clark and Jacob [215] who noted that an adversary C is able to intercept the message from A to B and replace the signature of A with C's own signature. Consequently A believes the protocol has been run with B while B believes it has been run with C.

1. $A \rightarrow C_B : ID_A, ID_B, T_A, L, \mathsf{Enc}_B(N_A), \mathsf{Sig}_A(ID_A, T_A, L, \mathsf{Enc}_B(N_A))$
1′. $C \rightarrow B : ID_C, ID_B, T_A, L, \mathsf{Enc}_B(N_A), \mathsf{Sig}_C(ID_C, T_A, L, \mathsf{Enc}_B(N_A))$
2′. $B \rightarrow C : ID_B, ID_C, \mathsf{Enc}_C(ID_B, N_A + 1)$
2. $C_B \rightarrow A : ID_B, ID_A, \mathsf{Enc}_A(ID_B, N_A + 1)$

Attack 4.3: Attack of Clark and Jacob on SPLICE/AS protocol

Attack 4.3 shows that neither party can be aware of their peer entity and that matching conversations cannot be guaranteed. However, the responder B can be sure that the signer of the first message is live, while A also gets this assurance, since only B is able to decrypt N_A. Clark and Jacob proposed Protocol 4.9 as an alternative to prevent their attack.

1. $A \rightarrow B : ID_A, ID_B, T_A, L, \mathsf{Enc}_B(ID_A, N_A), \mathsf{Sig}_A(ID_A, T_A, L, \mathsf{Enc}_B(ID_A, N_A))$
2. $B \rightarrow A : ID_B, ID_A, \mathsf{Enc}_A(ID_B, N_A + 1)$

Protocol 4.9: Clark–Jacob variant of SPLICE/AS

This protocol is again intended to provide mutual entity authentication. A look at the message sent from A to B shows that A has given no indication that she wishes to communicate with B so that this protocol does not provide knowledge of the peer entity. The only difference from Protocol 4.8 is that the identity of A is included in the encrypted field of message 1 which, it is assumed, cannot be altered by C. This assumption is only reasonable if the public key encryption algorithm used is non-malleable. Gray [333] pointed out that Attack 4.3 still works if the encrypted identity of A can be changed to another identity. He therefore proposed the simplified Protocol 4.10 which does provide knowledge of the peer entity. This can be seen as a hybrid of Protocols 4.3 and 4.5 since B relies on a timestamp to ensure liveness while A uses a nonce.

1. $A \rightarrow B : ID_A, ID_B, T_A, L, N_A, \mathsf{Sig}_A(ID_B, T_A, L, N_A)$
2. $B \rightarrow A : ID_B, ID_A, N_A, \mathsf{Sig}_B(ID_A, N_A)$

Protocol 4.10: Gray variant of SPLICE/AS

4.2.4 Comparison of Entity Authentication Protocols

Table 4.3 compares some of the main features of the various entity authentication protocols explored in this section. While there were attacks on the older protocols,

the analysis of the standardised protocols in the ISO/IEC 9798-3 standard by Basin *et al.* [61] has led to a better understanding. The versions referred to in Table 4.3 are those with the corrections specified by Basin *et al.* which have been formally analysed.

Table 4.3: Summary of properties of public key entity authentication protocols

Properties →	Liveness	Entity	Security	Attack
↓ *Protocol*		authentication	proof	
9798-3 one-pass unilateral (4.1)	*B*	*B*	Yes	No
9798-3 two-pass unilateral (4.2)	*B*	*B*	Yes	No
9798-3 two-pass mutual (4.3)	*A*+*B*	*A*+*B*	Yes	No
9798-3 three-pass mutual (4.5)	*A*+*B*	*A*+*B*	Yes	No
9798-3 two-pass parallel (4.7)	*A*+*B*	*A*+*B*	Yes	No
SPLICE/AS (4.8)	*A*+*B*	No	No	Yes
Clark–Jacob SPLICE (4.9)	*A*+*B*	No	No	Yes
Gray SPLICE (4.10)	*A*+*B*	*A*+*B*	No	No

4.3 Key Transport Protocols

As discussed in Chap. 1, key transport refers to protocols in which one principal chooses a session key and securely transports it to the other principal, or principals. Sometimes it can be difficult to classify protocols as key transport or key agreement. Some protocol specifications allow each of two principals to choose and transport their own key but leave open whether these two will be combined to form an agreed key, or used separately. We have included in this chapter those protocols that can potentially be used for key transport, but may be implemented to provide key agreement. An important practical example of key transport is found in various versions of the TLS handshake protocol; this is discussed in detail in Chap. 6.

4.3.1 Protocols in ISO/IEC 11770-3

In this section we shall examine protocols specified in the international standard ISO/IEC 11770 Part 3 [383]. The standard specifies six key transport protocols in a generic fashion and with some optional items. We shall see that many of these standardised protocols are related to other previously published protocols, particularly those in ISO/IEC 9798-3.

Protocols in the standard are presented using generic encryption and signature functions, but specific examples are included in an annex (which is not a formal part

of the standard). As with the protocols in ISO/IEC 9798-3 discussed in Sect. 4.2.1, there are various optional text fields included in all the standardised protocols which are mostly ignored in our descriptions here. The standard does not distinguish between signatures with message recovery and signatures with appendix by assuming that if a signature with appendix is used then the message signed is sent together with the signature. We show all protocols using signatures with appendix.

Protocol 4.11 shows Mechanism 1, the simplest in the standard. The session key K_{AB} is chosen by A and sent to B encrypted with B's public key. Also encrypted are the identity of A and the timestamp T_A (or alternatively a counter). In this protocol, as with all the key transport protocols in this standard, it is essential that the public key encryption used provides non-malleability as well as semantic security. If this were not the case then the adversary may be able to change the value of the fields A and T_A included with the encrypted session key. Notice that these fields are not generally required to be confidential, so it may be inferred that the designers intended to use the non-malleability to bind them to the session key. However, the properties of the encryption algorithm used are not explicitly stated in the standard.

1. $A \rightarrow B : \mathsf{Enc}_B(ID_A, K_{AB}, T_A)$

Protocol 4.11: ISO/IEC 11770-3 Key Transport Mechanism 1

From A's viewpoint Protocol 4.11 provides a good key since A can choose the key to be fresh and the encryption provides confidence that it is known only to herself and to B. However, A achieves no assurance with regard to key confirmation, or even that B is operative. Since there is no authentication at all of the origin of the key this protocol gives no assurance to B as to who this key is shared with. As long as B can rely on the freshness of T_A, he knows that the message received is fresh; but this does not seem useful since without authentication of the sending party B cannot deduce that K_{AB} itself is fresh. Indeed the standard states that the inclusion of T_A is optional.

Protocol 4.12 shows Mechanism 2, which extends Mechanism 1 by adding a signature of the whole message by A. The timestamp T_A may again be replaced by a counter. As before, the protocol provides a good key for A but provides no key confirmation from B. However, in contrast to Protocol 4.11, the signature of A allows B to achieve key confirmation. As long as B trusts A to generate the key faithfully, B also achieves the good key property. Protocol 4.12 is similar to Protocol 4.1 for entity authentication given in Sect. 4.2. Indeed, through use of the optional text fields in that protocol, Protocol 4.12 conforms to the 9798-3 standard as well as to 11770-3.

1. $A \rightarrow B : ID_B, T_A, \mathsf{Enc}_B(ID_A, K_{AB}), \mathsf{Sig}_A(ID_B, T_A, \mathsf{Enc}_B(ID_A, K_{AB}))$

Protocol 4.12: ISO/IEC 11770-3 Key Transport Mechanism 2

It is interesting to note that this protocol violates the first principle of Anderson and Needham (see Table 4.2). The motivation behind this principle is that the signature does not provide assurance that the signer knows the plaintext in the encrypted message. For example, in Protocol 4.12 an adversary C could remove the signature of A and replace it with C's own signature. This illustrates that it is essential that B trusts the signer of the message only to sign keys that it has generated and encrypted. However, as long as the encryption used is non-malleable an adversary cannot change the identity of A in the encrypted part of the message and so this does not result in a valid attack.

Anderson and Needham gave details of an attack on both RSA and discrete logarithm encryption algorithms which allows a malicious principal B to gain a signature on an encrypted message of his choice by registering a new public encryption key matched to the signed encrypted message. However, these attacks do not apply if non-malleable versions of encryption are used; also Syverson [704] discussed a number of practical difficulties with the attack. Nevertheless it is prudent for users of this protocol to consider such possibilities.

Mechanism 3, shown in Protocol 4.13, swaps around the order in which the signature and encryption are applied in Mechanism 2. The intention of each of the fields is the same so that the protocol achieves the same goals as Mechanism 2 as long as B trusts A to generate a good key. The inclusion of the timestamp T_A (or a counter) is again optional in the standard but without it B cannot gain key freshness or key confirmation.

1. $A \rightarrow B : \mathsf{Enc}_B(ID_B, K_{AB}, T_A, \mathsf{Sig}_A(ID_B, K_{AB}, T_A))$

Protocol 4.13: ISO/IEC 11770-3 Key Transport Mechanism 3

Protocol 4.14 is a closely related protocol proposed by Denning and Sacco [240]. Two preliminary messages, which allow principal A to obtain public key certificates for both A and B, are omitted here. The only difference from Protocol 4.13 is the omission of the identity of B.

1. $A \rightarrow B : \mathsf{Enc}_B(K_{AB}, T_A, \mathsf{Sig}_A(K_{AB}, T_A))$

Protocol 4.14: Denning–Sacco public key protocol

This omission allows an attack, discussed by Abadi and Needham [6], in which a malicious B can engage in a protocol run with A as initiator, and then send message 1 to C re-encrypted with C's public key. As a result C believes the key to be shared with A but it is also known to B. Abadi and Needham suggested including the identities of both A and B in the signature of message 1 to prevent this attack, but including only B's identity, as in Protocol 4.13, seems sufficient.

Mechanism 4, shown in Protocol 4.15, is a two-pass protocol very similar to Mechanism 2 (with roles reversed), the main difference being that A now uses a nonce N_A to achieve key freshness and entity authentication of B. As long as B is trusted to generate the key, A achieves the good key and key confirmation properties. B achieves the good key property but no authentication of A.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : ID_A, N_A, N_B, \mathsf{Enc}_A(ID_B, K_{AB}), \mathsf{Sig}_B(ID_A, N_A, N_B, \mathsf{Enc}_A(ID_B, K_{AB}))$

Protocol 4.15: ISO/IEC 11770-3 Key Transport Mechanism 4

The random number N_B is optional in the standard, and is apparently used only to maintain consistency with Protocol 4.2 which is the corresponding entity authentication protocol in the 9798-3 standard. Protocol 4.15, with N_B omitted, was proven secure by Shoup in his simulation model [674] under the assumption of only static corruptions by the adversary (recall that this is equivalent to a proof of security in the Bellare–Rogaway model as discussed in Chap. 2).

Protocol 4.16 shows Mechanism 5, which is a mutual version of Mechanism 4 for which two session keys K_{AB} and K_{BA} are chosen by A and B respectively. The standard suggests that the two session keys may be combined using a one-way hash function, in which case this protocol is strictly a key agreement protocol rather than a key transport protocol. It also suggests that either $\mathsf{Enc}_B(ID_A, K_{AB})$ could be omitted from message 3 so that K_{BA} becomes the session key, or $\mathsf{Enc}_A(ID_B, K_{BA})$ could be omitted from message 2 so that K_{AB} is the session key. If K_{AB} is used to define the session key, then only B obtains key confirmation. Mechanism 5 conforms to the 9798-3 standard by adding optional text fields to Protocol 4.5, the corresponding entity authentication protocol.

1. $A \rightarrow B : N_A$
2. $B \rightarrow A : N_B, N_A, ID_A, \mathsf{Enc}_A(ID_B, K_{BA}), \mathsf{Sig}_B(N_B, N_A, ID_A, \mathsf{Enc}_A(ID_B, K_{BA}))$
3. $A \rightarrow B : N_A, N_B, ID_B, \mathsf{Enc}_B(ID_A, K_{AB}), \mathsf{Sig}_A(N_A, N_B, ID_B, \mathsf{Enc}_B(ID_A, K_{AB}))$

Protocol 4.16: ISO/IEC 11770-3 Key Transport Mechanism 5

The final key transport protocol in the ISO/IEC 11770-3 standard is Mechanism 6, shown in Protocol 4.17. In contrast to all the other protocols in the standard it uses only encryption and no signatures. Perhaps it is most clear here that the encryption algorithm requires non-malleability since otherwise all the fields used for authentication could be potentially altered by the adversary.

The standard states that K_{AB} and K_{BA} may be combined using a one-way function to form a single session key. It is also stated that K_{AB} may be used by A to encipher

1. $A \rightarrow B : \mathsf{Enc}_B(ID_A, K_{AB}, N_A)$
2. $B \rightarrow A : \mathsf{Enc}_A(ID_B, K_{BA}, N_A, N_B)$
3. $A \rightarrow B : N_B$

Protocol 4.17: ISO/IEC 11770-3 Key Transport Mechanism 6

messages for B and to authenticate messages from B, and K_{BA} may be used in an analogous way by B. The protocol achieves mutual entity authentication and mutual key confirmation.

An earlier draft version of the ISO/IEC 11770-3 standard included Protocol 4.18 instead of Protocol 4.17; the former has become known as the Helsinki protocol due to the location of a particular meeting of the relevant standards committee. The only difference from the final standardised Protocol 4.17 is that in message 2 the identity field of B is missing.

1. $A \rightarrow B : \mathsf{Enc}_B(ID_A, K_{AB}, N_A)$
2. $B \rightarrow A : \mathsf{Enc}_A(K_{BA}, N_A, N_B)$
3. $A \rightarrow B : N_B$

Protocol 4.18: Helsinki protocol

Attack 4.4 on the Helsinki protocol was published by Horng and Hsu [364] in 1998. There is a strong similarity between this attack and Lowe's earlier attack on the Needham–Schroeder public key protocol discussed in Sect. 4.3.3 below. The adversary, C, induces A to commence the protocol with C, and then starts a protocol run with B while masquerading as A.

1. $A \rightarrow C : \mathsf{Enc}_C(ID_A, K_{AB}, N_A)$
1′. $C_A \rightarrow B : \mathsf{Enc}_B(ID_A, K'_{AB}, N_A)$
2′. $B \rightarrow C_A : \mathsf{Enc}_A(K_{BA}, N_A, N_B)$
2. $C \rightarrow A : \mathsf{Enc}_A(K_{BA}, N_A, N_B)$
3. $A \rightarrow C : N_B$
3′. $C_A \rightarrow B : N_B$

Attack 4.4: Attack on Helsinki protocol

Both A and B have the view of a successful protocol run. However, if the session key is $f(K_{AB}, K_{BA})$ for some one-way function f, then A 'believes' she shares this key with C, while B 'believes' he shares $f(K'_{AB}, K_{BA})$ with A. Notice that the goal of

implicit key authentication has not been violated in this attack, because C does not know K_{BA} and therefore cannot compute either of the session keys accepted by A and B. However, entity authentication is not achieved in that B has incorrect knowledge of his peer entity. Mitchell and Yeun [560] proposed to fix the protocol by adding B's identity to message 2 which, as we have seen, was the solution adopted in the final ISO/IEC 11770 standard.

Earlier versions of the ISO/IEC 11770-3 standard claimed that key confirmation is achieved for both A and B in Protocol 4.17. Although this intuitively seems to be true, Cremers and Horvat [229] showed that a complex attack is possible which violates key confirmation in the case where an optional text field (not shown in Protocol 4.17) is used. The additional text field in the first message allows a principal in the role of B to interpret an instance of message 2 as an instance of message 1, and that principal will hence generate a new message 2. Although this attack requires several, possibly unrealistic, assumptions it shows that the claimed key confirmation property does not always hold. This claim of key confirmation is removed in the 2015 version of the ISO/IEC 11770-3 standard.

4.3.2 Blake-Wilson and Menezes Key Transport Protocol

Blake-Wilson and Menezes [108] have proven the security of Protocol 4.19, which is a simplified version of Protocol 4.16 (indeed it is an optional variant conforming to the standard). Here the session key K_{AB} is chosen by B. The protocol was strongly related to Protocol 4.5 for entity authentication, and like that was proven secure in the Bellare–Rogaway model. The proof incorporates the assumption that the encryption algorithm provides non-malleability by assuming that the adversary is able to conduct a chosen ciphertext attack.

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : ID_B, ID_A, N_B, N_A, \mathsf{Enc}_A(ID_B, K_{AB}), \mathsf{Sig}_B(ID_B, ID_A, N_B, N_A, \mathsf{Enc}_A(ID_B, K_{AB}))$
3. $A \rightarrow B : ID_A, ID_B, N_B, \mathsf{Sig}_A(ID_A, ID_B, N_B)$

Protocol 4.19: Blake-Wilson–Menezes key transport protocol

We note that again this protocol ignores the first principle of Anderson and Needham (see Table 4.2) not to sign encrypted data. In this case the order of encryption and signature is not chosen by chance; it *has* to be this way around in order for the proof to work. The reason for this is that it is necessary in the proof to be able to simulate how the principals in the protocol behave when the plaintext is not known. If the plaintext is included in a signature then it is not possible to determine if a received message has been properly signed without knowing if the plaintext in the signature is the same as in the ciphertext. Anderson and Needham [35] commented that protocol logics, such as the BAN logic, on the contrary cannot be used when

encrypted messages are signed. This illustrates an interesting dichotomy between different methods of protocol validation.

4.3.3 Needham–Schroeder Public Key Protocol

The Needham–Schroeder public key protocol [581] was one of the earliest published key establishment protocols along with its well-known companion using symmetric encryption (see Sect. 3.4.1). Protocol 4.20 shows the messages exchanged. There is a strong similarity with the Helsinki protocol (Protocol 4.18). The protocol was designed to provide mutual entity authentication but with the option of using the exchanged nonces, N_A and N_B, as shared secrets for key establishment.

1. $A \rightarrow B: \mathsf{Enc}_B(N_A, ID_A)$
2. $B \rightarrow A: \mathsf{Enc}_A(N_A, N_B)$
3. $A \rightarrow B: \mathsf{Enc}_B(N_B)$

Protocol 4.20: Needham–Schroeder public key protocol

Although this protocol was designed as long ago as 1978, it aroused quite some interest much later. Lowe in 1996 [501] discovered Attack 4.5 which shows that B cannot be sure that the final message came from A. Notice that A has never explicitly declared her intention to converse with B so this protocol cannot provide any assurance to B that A has knowledge of B as the peer entity.

1. $A \rightarrow C: \mathsf{Enc}_C(N_A, ID_A)$
1′. $C_A \rightarrow B: \mathsf{Enc}_B(N_A, ID_A)$
2′. $B \rightarrow C_A: \mathsf{Enc}_A(N_A, N_B)$
2. $C \rightarrow A: \mathsf{Enc}_A(N_A, N_B)$
3. $A \rightarrow C: \mathsf{Enc}_C(N_B)$
3′. $C_A \rightarrow B: \mathsf{Enc}_B(N_B)$

Attack 4.5: Lowe's attack on Needham–Schroeder public key protocol

Attack 4.5 is similar to Attack 4.4 on the Helsinki protocol. In order to fix the protocol against his attack, Lowe proposed the variant Protocol 4.21 which simply includes the identifier of B in the second message.

Lowe was able to prove secure a finite model of Protocol 4.21 using the model checker FDR (see Sect. 1.6.1) and extended the proof to the infinite version using several pages of mathematical reasoning. It is again important to notice that this revised protocol is only secure as long as the encryption algorithm used provides non-malleability. Otherwise it cannot be guaranteed that an adversary will not be

1. $A \rightarrow B : \mathsf{Enc}_B(N_A, ID_A)$
2. $B \rightarrow A : \mathsf{Enc}_A(N_A, N_B, ID_B)$
3. $A \rightarrow B : \mathsf{Enc}_B(N_B)$

Protocol 4.21: Lowe's variant of Needham–Schroeder public key protocol

able to alter the value of the identifier B in message 2 even without knowing the values of N_A and N_B. Protocol 4.21 has a lot in common with Protocol 4.17 and the properties it achieves are the same. Instead of encrypting an explicit session key, the nonces of A and B can act as a shared secret; this is the reason why the third message needs to be encrypted.

Attack of Bana, Adão and Sakurada

Bana, Adão and Sakurada [52] proposed a typing attack on Protocol 4.21. In the attack shown as Attack 4.6, an adversary I masquerades as B and intercepts the third message from A. I then replays this message to B to initiate a new run with B. This attack requires the assumption that the encryption of a single nonce sent in the third message of one run will be interpreted as the encryption of a nonce and a principal identity when replayed as the first message of another run.

3. $A \rightarrow I_B : \mathsf{Enc}_B(N_B)$
$1'$. $I \rightarrow B : \mathsf{Enc}_B(N_I, ID_I)$
$2'$. $B \rightarrow I : \mathsf{Enc}_I(N_I, N'_B, ID_B)$
3. $I_A \rightarrow B : \mathsf{Enc}_B(N'_B)$

Attack 4.6: Bana–Adão–Sakurada attack on Needham–Schroeder–Lowe protocol

Key Compromise Impersonation Attack of Basin, Cremers and Horvat

Basin, Cremers and Horvat [59] later showed that a key compromise impersonation attack is possible on Protocol 4.21. Attack 4.7 shows an attacking run, where an intruder I induces A to start a protocol run with B. The intruder then intercepts message 2 from B and decrypts this message, using knowledge of the private key of A, to obtain N_A. Finally, I masquerades as B to A, by sending an encrypted message which consists of a concatenation of N_A and a nonce value chosen by I, which will be accepted by A. This means that A will accept N_I as shared with B, whereas it is actually shared with I. The result of the attack is that it is no longer safe to use the nonces of A and B as a shared secret. Note that the attack requires more assumptions than usual, namely that the party to be impersonated is online.

1. $A \rightarrow B: \ \mathsf{Enc}_B(N_A, ID_A)$
2. $B \rightarrow I_A: \mathsf{Enc}_A(N_A, N_B, ID_B)$
2′. $I_B \rightarrow A: \mathsf{Enc}_A(N_A, N_I, ID_B)$
3. $A \rightarrow I_B: \mathsf{Enc}_B(N_I)$

Attack 4.7: Key compromise impersonation attack on Needham–Schroeder–Lowe protocol

To overcome the attack, Basin *et al.* suggested Protocol 4.22 as a solution, using hashing to make the nonces of A and B secret from an adversary who has obtained either A or B's private key but not both.

1. $A \rightarrow B: \mathsf{Enc}_B(N_A, ID_A)$
2. $B \rightarrow A: \mathsf{Enc}_A(h(N_A, N_B), N_B, ID_B)$
3. $A \rightarrow B: \mathsf{Enc}_B(h(N_B))$

Protocol 4.22: Needham–Schroeder–Lowe protocol modified by Basin *et al.*

4.3.4 Needham–Schroeder Protocol Using Key Server

The original Needham–Schroeder public key protocol allows A and B to obtain each other's public keys from a trusted server. This requires additional message flows that are omitted from the simplified version attacked by Lowe. The full version of the protocol with a key server present (sometimes called NSPK-KS) is shown as Protocol 4.23.

1. $A \rightarrow S: \ ID_B$
2. $S \rightarrow A: \ Cert(B)$
3. $A \rightarrow B: \mathsf{Enc}_B(N_A, ID_A)$
4. $B \rightarrow S: \ ID_A$
5. $S \rightarrow B: \ Cert(A)$
6. $B \rightarrow A: \mathsf{Enc}_A(N_A, N_B)$
7. $A \rightarrow B: \mathsf{Enc}_B(N_B)$

Protocol 4.23: Needham–Schroeder public key protocol using key server

Meadows [534] found an attack on Protocol 4.23 using the NRL Protocol Analyzer. The attack depends on the assumption that a random nonce can be used as a

principal name. The attacker I masquerades as A to initiate a protocol run with B. I intercepts B's reply and sends it to A as the first message of a new protocol run. Upon receipt of this message, A believes it is a run initiated by a principal having identity N_B. Now A will send N_B in cleartext to S. Using knowledge of N_B, I can masquerade as A to B in the first run, as shown in Attack 4.8.

3. $I_A \rightarrow B: \mathsf{Enc}_B(N_I, ID_A)$
4. $B \rightarrow S: ID_A$
5. $S \rightarrow B: Cert(A)$
6. $B \rightarrow I_A: \mathsf{Enc}_A(N_I, N_B)$

$1'$. $I \rightarrow A: \mathsf{Enc}_A(N_I, N_B)$

$2'$. $A \rightarrow I_S: N_B$

7. $I_A \rightarrow B: \mathsf{Enc}_B(N_B)$

Attack 4.8: Meadows' attack on NSPK-KS

4.3.5 Protocols in the X.509 Standard

The X.500 series of recommendations was standardised by the ITU (formerly CCITT) in parallel with ISO to provide directory services for communications. One purpose of the directory is to store certificates for public keys and Part 8 of the standard [387] uses such public keys as the basis for the Authentication Framework. This framework specification includes a number of protocols for authentication and key establishment which can be used for access control to the directory or for other purposes. The protocols are classified as either *simple* or *strong*. Simple authentication uses passwords sent either in cleartext or as input to a one-way function; we shall consider only the strong authentication protocols that use public key cryptographic methods.

There are three protocols specified, with one, two and three message flows respectively. Each protocol extends the previous one by adding an extra message. The goal of each protocol is transport of a session key from A to B and, for the two- and three-flow protocols, transport of a session key from B to A.

In the simplest protocol there is only one message which is sent from A to B. The certificates (or more generally a chain of certificates) are omitted in the following descriptions, as well as optional signed data. The standard allows for simplified versions in which the session key is omitted, intended to provide entity authentication only. Protocol 4.24 shows the one-pass version with the encrypted data forming the session key K_{AB}; this is suggested as only one possibility by the standard.

There is a strong similarity between Protocol 4.24 and Protocol 4.12. The main difference is that here the identity of A is missing from the encrypted data, which makes any potential attacks on Protocol 4.12 easier to mount. In particular, an adversary may remove the signature on the message and replace it with a new signature on

1. $A \rightarrow B: T_A, N_A, ID_B, \mathsf{Enc}_B(K_{AB}), \mathsf{Sig}_A(T_A, N_A, ID_B, \mathsf{Enc}_B(K_{AB}))$

Protocol 4.24: X.509 one-pass authentication

the identical message which leads to B believing that the key was sent by the adversary. This problem has been pointed out by Burrows *et al.* [171] while I'Anson and Mitchell [371] also discussed the consequences of such an attack when the encrypted portion of the message acts as a confidential request for information. Both sets of authors suggested that to fix this problem the signature should include the unencrypted key (hashed to protect its confidentiality) with the encrypted key sent separately. An alternative is to use Protocol 4.12 instead.

Basin *et al.* [59] have pointed out that a compromise of B's long-term private key allows a key impersonation attack against B, a key compromise impersonation attack. As an improvement, they proposed a variant of the one-pass protocol, adding a second pass as shown in Protocol 4.25.

1. $B \rightarrow A: \mathsf{Enc}_A(N_B)$
2. $A \rightarrow B: T_A, N_A, ID_B, \mathsf{Enc}_B(\{K_{AB}\}_{N_B}), \mathsf{Sig}_A(T_A, N_A, ID_B, \mathsf{Enc}_B(\{K_{AB}\}_{N_B}))$

Protocol 4.25: Basin–Cremers–Horvat variant of X.509 one-pass authentication

Protocol 4.25 employs a symmetric key chosen by B uniquely for this session, N_B. Notice that an adversary who knows B's long-term private key can remove the asymmetric encryption to obtain the ciphertext $\{K_{AB}\}_{N_B}$. However, this will not help the adversary in obtaining the session key K_{AB} as long as N_B remains confidential.

Protocol 4.26 is the two-pass X.509 protocol; the same first message is sent from A to B as in Protocol 4.24 and a reply is sent from B to A, which is symmetrical except that both nonces are included in the signed part of this message.

1. $A \rightarrow B: T_A, N_A, ID_B, \mathsf{Enc}_B(K_{AB}), \mathsf{Sig}_A(T_A, N_A, ID_B, \mathsf{Enc}_B(K_{AB}))$
2. $B \rightarrow A: T_B, N_B, ID_A, N_A, \mathsf{Enc}_A(K_{BA}), \mathsf{Sig}_B(T_B, N_B, ID_A, N_A, \mathsf{Enc}_A(K_{BA}))$

Protocol 4.26: X.509 two-pass authentication

Similar remarks to those concerning Protocol 4.24 apply. The protocol may be fixed in different ways such as by adding the sender's name to the encrypted blocks to form a mutual version of Protocol 4.12. The encrypted data sent from B to A is shown as a session key K_{BA}; although the standard suggests this data may be used as a session key there is no recommendation on whether this should be used separately

or combined with K_{AB}. Protocol 4.27 is the final X.509 protocol and includes a third message intended to provide acknowledgement of message 2 by A.

1. $A \rightarrow B : T_A, N_A, ID_B, \mathsf{Enc}_B(K_{AB}), \mathsf{Sig}_A(T_A, N_A, ID_B, \mathsf{Enc}_B(K_{AB}))$
2. $B \rightarrow A : T_B, N_B, ID_A, N_A, \mathsf{Enc}_A(K_{BA}), \mathsf{Sig}_B(T_B, N_B, ID_A, N_A, \mathsf{Enc}_A(K_{BA}))$
3. $A \rightarrow B : N_B, ID_B, \mathsf{Sig}_A(N_B, ID_B)$

Protocol 4.27: X.509 three-pass authentication

There is no need in either this protocol or the two-pass protocol for both T_B and N_A to be included since either of them is enough for A to acquire freshness of K_{BA}. Indeed the standard states that T_B may be set to zero which, as pointed out by Burrows *et al.* [171], makes T_B completely redundant. The standard also states that T_A need not be checked in message 1 either.

In the first (1988) version of the X.509 standard, the field B was absent from the third message of Protocol 4.27. A consequence of this was that if T_A was not used for freshness then the protocol could be attacked, since B is not able to check that message 3 is part of the same protocol run. Specifically C can replay an old first message from A to B and, in order to complete the protocol, need only obtain A's signature on the challenge received in message 2. C can now obtain such a signature by engaging in a protocol run with A as the initiator. This attack was detailed by both Burrows *et al.* [171] and I'Anson and Mitchell [371].

4.3.6 Public Key Kerberos

Protocol 4.28 involves two parties: a client A and an authentication server S. The first message includes an authenticator $\mathsf{Sig}_A(T_A, N_A)$ containing a timestamp and a nonce N_A signed by A, the name of the ticket granting server B for whom A wants a session key, and another nonce N'_A. If the timestamp is sufficiently recent, S generates a fresh symmetric key k and replies with a message containing credentials for A. The first part of this message contains S's signature over k and the nonce N_A sent in the first message. Because the signature is encrypted using A's public key, only A can learn k. Using k, A learns the session key K_{AB} from the last part of the second message. The field TGT is of the form $\{K_{AB}, ID_A, T_S\}_{K_{BS}}$, where K_{BS} is a long-term key shared by B and S. Note that A cannot read the data that is encrypted with K_{BS}.

1. $A \rightarrow S : Cert(A), \mathsf{Sig}_A(T_A, N_A), ID_A, ID_B, N'_A$
2. $S \rightarrow A : \mathsf{Enc}_A(Cert(S), \mathsf{Sig}_S(k, N_A)), ID_A, \mathrm{TGT}, \{K_{AB}, N'_A, T_S, ID_B\}_k$

Protocol 4.28: Ticket granting protocol of public key Kerberos

Protocol 4.28 was attacked by Cervesato *et al.* [185] who noted that an adversary I is able to intercept the message from A to S and replace the signature of A with I's own signature. Consequently, A accepts a session key for use with B, even though it is known to I. Attack 4.9 is similar to Attack 4.3 on the SPLICE/AS protocol.

1. $A \rightarrow I_S : Cert(A), \mathsf{Sig}_A(T_A, N_A), ID_A, ID_B, N'_A$
1′. $I \rightarrow S : Cert(I), \mathsf{Sig}_I(T_A, N_A), ID_I, ID_B, N'_A$
2′. $S \rightarrow I : \mathsf{Enc}_I(Cert(S), \mathsf{Sig}_S(k, N_A)), ID_I, \mathrm{TGT}, \{K_{IB}, N'_A, T_S, ID_B\}_k$
2. $I_S \rightarrow A : \mathsf{Enc}_A(Cert(S), \mathsf{Sig}_S(k, N_A)), ID_A, \mathrm{TGT}, \{K_{IB}, N'_A, T_S, ID_B\}_k$

Attack 4.9: Attack of Cervesato *et al.* on public-key Kerberos

In order to fix the protocol against this attack, Cervesato *et al.* proposed a variant protocol which simply includes the identifier of A in the signature of S.

4.3.7 Beller–Chang–Yacobi Protocols

Beller, Chang and Yacobi [80, 81, 82], and Beller and Yacobi [83] proposed hybrid protocols using a combination of asymmetric and symmetric cryptographic algorithms. These protocols were designed to satisfy the requirements of the mobile communications environment. They were intended to provide security between a mobile station and a base station of the fixed network, rather than to provide end-to-end security between mobile users.

There are at least two requirements in addition to those usually needed for authentication and key establishment protocols.

- The computational load on the mobile station must be minimised, even at the expense of increased load on the base station.
- The identity of the mobile station must remain hidden from the adversary.

The protocols of Beller *et al.* were critically examined by Carlsen [183], who identified some possible attacks and suggested protocol modifications to avoid them. He also pointed out an inherent shortcoming of their protocols. Although these protocols hide the identity of an initiating mobile station, the dual requirement of hiding the identity of the responding station remained unsolved.

The protocols of Beller *et al.* rely on a public key cryptosystem for which encryption is particularly efficient, at least in comparison to other public key cryptosystems. The specific public key cryptosystem employed is due to Rabin [622], in which encryption and decryption are tantamount, respectively, to modulo squaring and extracting a modulo square root (MSR). Instead of showing the mathematical details of the MSR algorithms, we shall continue to use our more general notation in describing the protocols of Beller *et al.* (hereafter referred to as the MSR protocols). The MSR protocols consist of three variants with different complexity and security features.

MSR Protocol

In the following, the notation SC_B is a structure known as the *secret certificate* of the mobile station, B, which is issued by a trusted central authority. This certificate can be checked by anyone using the public key of the central authority in order to verify the mobile station's identity. Unlike a usual public key certificate, this certificate must be kept secret from all other mobile users and eavesdroppers, because it is all that is required to masquerade as B. Protocol 4.29 shows the basic MSR protocol [82].

---

1. $A \rightarrow B : ID_A, K_A$
2. $B \rightarrow A : \mathsf{Enc}_A(K_{AB}), \{ID_B, SC_B\}_{K_{AB}}$

Protocol 4.29: Basic MSR protocol of Beller, Chang and Yacobi

Upon receiving the base station A's public key K_A, the mobile station uses it to encrypt the session key K_{AB}, and sends the encrypted message to A. The mobile station also sends its identity and secret certificate encrypted under K_{AB} to authenticate K_{AB} to the base station. The symmetric encryption with K_{AB} in message 2 is of negligible computational effort compared to the public key encryption in the same message; therefore the computational effort at the mobile station is effectively limited to that of modulo squaring of the session key. Carlsen [183] identified two security weaknesses in Protocol 4.29. The first of these weaknesses appears to have been recognised as early as 1993 by Beller *et al.* [82] themselves.

- The public key of A is uncertified, thereby allowing anyone to masquerade as A.
- It is not possible for A to differentiate between a new run of the protocol and one where messages from an old run are replayed by a malicious adversary. At the least this may allow the adversary to transfer connection charges to B. In addition replay of an old compromised session key then allows the adversary to masquerade as B.

Improved MSR (IMSR) Protocol

The improved MSR protocol of Beller *et al.* [82], IMSR, overcomes a major weakness of MSR by using a public key certificate of the base station. This results in a twofold increase in the computational complexity as compared to Protocol 4.29 since the mobile station now calculates an additional modulo square to verify the base station's certificate on receiving message 1.

Apart from this feature it is identical to the basic MSR protocol, and therefore does not address the problem of replay. Carlsen [183] recognised this and suggested an 'improved IMSR' protocol which includes a challenge–response mechanism to allow B to detect a session key replay as shown in Protocol 4.30. (He also adds an expiration time to the public key certificate of A, to allow for checks on the certificate's validity, while at the same time deleting A's identity from the certificate for

purposes of anonymity. The effect of this latter change is that base station 'impersonation attacks' become possible, as pointed out by Mu and Varadharajan [569]. As usual, this public key certificate is omitted from our description.)

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : \mathsf{Enc}_A(K_{AB}), \{N_A, ID_B, SC_B\}_{K_{AB}}$

Protocol 4.30: Improved IMSR protocol of Carlsen

Upon receiving the final message, A decrypts it using the session key K_{AB}, and checks that the value N_A is the same as the nonce sent in message 1. Curiously, although Carlsen clearly identified the problem of replay, his suggested improvement does not really overcome it. In the above protocol, if K_{AB} is compromised an adversary can obtain SC_B, and thus freely masquerade as B.

Beller *et al.* [81] mentioned the security threat posed to the IMSR protocol by an adversary who obtains the session key; however, they do not treat the problem of replay as such. They suggested two methods to protect the certificate of B against a compromised session key. One of these, called the *split-certificate method*, is to have the mobile station send at least half of its certificate encrypted together with K_{AB} under the base station's public key. The other, called the *split-key method*, is to divide K_{AB} into two subkeys, one of which is used to encrypt the protocol message that authenticates B to A, and the other is used as the session key proper. Both these methods can also be used to overcome the weakness of Protocol 4.30.

Beller–Yacobi Protocol

In a separate publication, Beller and Yacobi [83] suggested a further variation on the IMSR protocol. Like the MSR+DH protocol discussed below, the Beller–Yacobi protocol employs a public key for the mobile as well as the base station. The mobile station's private key is used to implement digital signatures using the ElGamal algorithm [267]. The main reason for choosing this algorithm is that most of the computations required for signature generation can be executed prior to choosing the message to be signed. This means that it is easy for the mobile processor to do most of its work offline, during idle time between calls. The basic structure of Protocol 4.31 is similar to Protocol 4.29. The main difference is in the last two messages which implement a challenge–response mechanism based on digital signatures.

In the third message, A sends a nonce N_A encrypted using K_{AB}. B then returns N_A signed using his private key together with his identity, public key and certificate $Cert(B)$, all encrypted under K_{AB}. Finally, A decrypts this message and verifies the signature on N_A.

We now present a potential attack on Protocol 4.31 [141]. Although this attack makes quite strong assumptions, it may be taken seriously because it indicates a flaw in the protocol design. We understand that the same attack was found independently

1. $A \rightarrow B : ID_A, K_A$
2. $B \rightarrow A : \mathsf{Enc}_A(K_{AB})$
3. $A \rightarrow B : \{N_A\}_{K_{AB}}$
4. $B \rightarrow A : \{ID_B, K_B, Cert(B), \mathsf{Sig}_B(N_A)\}_{K_{AB}}$

Protocol 4.31: Beller–Yacobi protocol

by the original authors subsequent to the protocol's publication. The adversary, C, must be a legitimate user known to A. Further, C needs to be able to set up simultaneous sessions with both A and B. (C could be a rogue mobile and base station in collusion.) In Attack 4.10, C is able to convince B that C is A.

1. $A \rightarrow C_B : ID_A, K_A$
2. $C_B \rightarrow A : \mathsf{Enc}_A(K_{AB})$
3. $A \rightarrow C_B : \{N_A\}_{K_{AB}}$
1′. $C \rightarrow B : ID_C, K_C$
2′. $B \rightarrow C : \mathsf{Enc}_C(K'_{AB})$
3′. $C \rightarrow B : \{N_A\}_{K'_{AB}}$
4′. $B \rightarrow C : \{ID_B, K_B, Cert(B), \mathsf{Sig}_B(N_A)\}_{K'_{AB}}$
4. $C_B \rightarrow A : \{ID_B, K_B, Cert(B), \mathsf{Sig}_B(N_A)\}_{K_{AB}}$

Attack 4.10: Attack on Beller–Yacobi protocol

The essence of the attack is that C starts a parallel session with B in order to obtain B's signature on A's challenge N_A. At the end of the attack, A accepts K_{AB} as a session key with B, whereas in fact it is shared with C. The session started between C and B can be dropped after the receipt of message 4′. Note that message 3 must precede message 3′, and message 4′ must precede message 4; the remaining messages may overlap each other.

There is a simple way to alter the protocol so as to avoid the attack [141]. Essentially the change is to have B sign the new session key K_{AB} when it is first sent to A, in message 2, together with the challenge N_A, which guarantees its freshness. The key must have its confidentiality protected by a suitable one-way hash function h, but the use of such a function is a standard practice in most digital signature schemes. Since K_{AB} is now authenticated in message 2, message 4 is redundant and message 3 is used simply for B to verify that A has received the key. Protocol 4.32 shows the revised version.

Comparison with Protocol 4.31 shows that Protocol 4.32 is no more costly in either computational or communications requirements than the original. Therefore it appears to be just as suitable as the original for the situation where B has limited computing power.

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : \mathsf{Enc}_A(K_{AB}), \{ID_B, K_B, Cert(B)\}_{K_{AB}}, \mathsf{Sig}_B(h(ID_A, ID_B, N_A, K_{AB}))$
3. $A \rightarrow B : \{N_A\}_{K_{AB}}$

Protocol 4.32: Improved Beller–Yacobi protocol

Beller–Yacobi MSR+DH Protocol

Beller and Yacobi also proposed an extended version of the IMSR protocol which incorporates Diffie–Hellman key exchange [252]. (Diffie–Hellman key exchange and many related protocols are discussed in detail in Chap. 5.) A major improvement is that now the mobile terminal has a public key which means that it no longer needs to reveal its permanent secret to the base.

In this protocol the base station A has two public keys: the Diffie–Hellman key and a public key used for encryption in the modular square root system. The mobile station B has one public key. Carlsen [183] has also suggested an 'improved MSR+DH' protocol by making similar modifications to those carried out in the improved MSR protocol. Protocol 4.33 shows the improved MSR+DH protocol [183].

1. $A \rightarrow B : ID_A, N_A$
2. $B \rightarrow A : \mathsf{Enc}_A(x), \{N_A, ID_B\}_x$

Protocol 4.33: Carlsen's improved Beller–Chang–Yacobi MSR+DH protocol

The static Diffie–Hellman key S_{AB} (see Sect. 5.2) is used to derive the session key as $K_{AB} = \{x\}_{S_{AB}}$. Although the security of the MSR+DH protocol appears far improved over the other MSR variants, it carries a computational price. Now both parties need to calculate a full modular exponentiation at session set-up leading, as per the calculations of Beller *et al.*, to a 100 times increase in the required computing power.

4.3.8 TMN Protocol

One of the earliest key establishment protocols designed for use in a mobile environment was that of Tatebayashi, Matsuzaki and Newman [708], which has widely become known as the TMN protocol. A favourite with protocol analysts due to its many vulnerabilities, we include it for its historical importance. The principals are two mobile stations A and B who wish to exchange a session key to provide end-to-end security, and a server S with whom they share distinct long-term secrets. The design takes account of the limitations in mobile station computational ability by requiring the mobile stations only to encrypt with short RSA [630] public exponents. A number of attacks have been published on the TMN protocol, some of which rely

on the specific cryptographic algorithms used, and others which exploit problems in the message structures [424].

The TMN paper [708] includes two protocols. The first protocol (called KDP1) contains no authentication information and was found by the designers to be vulnerable to certain attacks. As an improvement, Protocol 4.34 (called KDP2) was proposed. The field s_A is a shared secret value between S and A while T_A is a timestamp generated by A. The fields s_B and T_B are defined analogously. The session key K_{AB} is generated by B, while A generates a one-time key-encrypting key KEK. The second protocol message is simply a request from S for B to respond.

1. $A \rightarrow S: \mathsf{Enc}_S(T_A, s_A, KEK)$
2. $S \rightarrow B: RSVP$
3. $B \rightarrow S: \mathsf{Enc}_S(T_B, s_B, K_{AB})$
4. $S \rightarrow A: \{K_{AB}\}_{KEK}$

Protocol 4.34: Simplified TMN protocol (KDP2)

The encryption in message 4 is carried out using a symmetric cryptosystem. The identities of A and B are relevant to the intended meaning of messages 3 and 4, respectively, although they are not included within the encrypted fields of these messages. As a result, neither A nor B has any assurance of who else has the session key K_{AB}. Since S has a shared secret with both A and B, it is questionable whether the use of public key cryptography in the TMN protocol is justified.

A different attack, based on the algebraic properties of the encryption algorithms, was found by Park *et al.* [600]. These authors proposed a variant protocol with different algorithms, but its general structure is identical to Protocol 4.34. Consequently it suffers from the same weaknesses.

4.3.9 AKA Protocol

The Texas A&M University Anarchistic Key Authorization (AKA) protocol was proposed by Safford *et al.* [643]. The name of the protocol reflects the use of informally certified public keys in the style of PGP [783] although that does not seem to influence the design in any particular way. AKA employs an unusual mechanism to provide forward secrecy; instead of Diffie–Hellman key agreement, short-term RSA keys are used.

A number of variations on the basic idea were given by Safford *et al.* providing greater efficiency and flexibility. Protocol 4.35 is one simple version in which only B chooses a short-term public key; in other variants both parties choose and exchange a short-term public key. The public keys K_A and K_B are long-term public keys of A and B while K'_B is a short-term public key chosen by B for this session; encryption of M using K'_B is denoted by $\mathsf{Enc}'_B(M)$.

1. $A \rightarrow B : K_A$
2. $B \rightarrow A : K_B, K'_B$
3. $A \rightarrow B : \mathsf{Enc}'_B(N_A, ID_A)$
4. $B \rightarrow A : \mathsf{Enc}_A(N_B)$
5. $B \rightarrow A : \mathsf{Enc}_A(\mathsf{Sig}_B(N_A))$
6. $A \rightarrow B : \mathsf{Enc}'_B(H(N_B))$

Protocol 4.35: AKA protocol

The session key is defined as a function of the shared secret $N_A \oplus N_B$. The point of using the temporary public key can be seen if we consider the result of a compromise of either A's or B's long-term private key. Since the private key corresponding to K'_B is not compromised then N_A cannot be recovered and so forward secrecy is provided.

Abadi [1] has shown that this protocol is vulnerable to an attack due to the lack of sufficient authenticating information inside the signature of B in message 5. This means that an adversary C can interleave a run of the protocol with B with another run with A in which C masquerades as B. Attack 4.11 shows a specific attacking run: C replaces B's short-term public key with a new short-term key K'_C. C can use the signature of message 5 to convince A that C is in fact B. C simply aborts the run with B after capturing message 6 from A.

1. $A \rightarrow C_B : K_A$
1′. $C_A \rightarrow B : K_A$
2′. $B \rightarrow C_A : K_B, K'_B$
2. $C_B \rightarrow A : K_B, K'_C$
3. $A \rightarrow C_B : \mathsf{Enc}'_C(N_A, ID_A)$
3′. $C_A \rightarrow B : \mathsf{Enc}'_B(N_A, ID_A)$
4′. $B \rightarrow C_A : \mathsf{Enc}_A(N_B)$
4. $C_B \rightarrow A : \mathsf{Enc}_A(N_C)$
5′. $B \rightarrow C_A : \mathsf{Enc}_A(\mathsf{Sig}_B(N_A))$
5. $C_B \rightarrow A : \mathsf{Enc}_A(\mathsf{Sig}_B(N_A))$
6. $A \rightarrow C_B : \mathsf{Enc}'_C(H(N_C))$

Attack 4.11: Abadi's attack on AKA protocol

The result of the attack is that A and C share the secret $N_A \oplus N_C$ but A believes that this secret is shared with B. Abadi pointed out that this attack can be prevented by including more fields in the signature in message 5; specifically the nonce N_B and the identities of A and B should be included.

A similar attack applies to the other AKA variants. In the versions where both A and B choose a short-term key, the adversary can replace them both with new short-

term keys and obtain all encrypted information before sending it on re-encrypted with the expected key.

4.3.10 Comparison of Key Transport Protocols

Table 4.4 summarises the main features of the main key transport protocols examined in this chapter. Some additional variants of the protocols listed in the table are included earlier in this chapter. We record the properties of key control, key freshness, key authentication and key confirmation in each case, even though in many cases these properties are not formally proven.

Table 4.4: Summary of major properties of key transport protocols using public key cryptography

Properties → ↓ *Protocol*	Key control	Key freshness	Key auth.	Key conf.	Attack	Security proof
11770-3 Mechanism 1 (4.11)	*A*	*A*	*A*	No	No	No
11770-3 Mechanism 2 (4.12)	*A*	*A*+*B*	*A*+*B*	*B*	No	No
11770-3 Mechanism 3 (4.13)	*A*	*A*+*B*	*A*+*B*	*B*	No	No
11770-3 Mechanism 4 (4.15)	*B*	*A*+*B*	*A*+*B*	*A*	No	Yes
11770-3 Mechanism 5 (4.16)	*A*+*B*	*A*+*B*	*A*+*B*	*A*	No	No
11770-3 Mechanism 6 (4.17)	*A*+*B*	*A*+*B*	*A*+*B*	No	No	No
Blake-Wilson–Menezes (4.19)	*B*	*A*+*B*	*A*+*B*	*A*	No	Yes
Needham–Schroeder (4.20)	*A*+*B*	*A*+*B*	*A*+*B*	*A*+*B*	Yes	Yes
Needham–Schroeder key server (4.23)	*A*+*B*	*A*+*B*	*A*+*B*	*A*+*B*	Yes	No
X.509 one-pass (4.24)	*A*	*A*	*A*	No	Yes	No
X.509 two-pass (4.26)	*A*	*A*	*A*	No	Yes	No
X.509 three-pass (4.27)	*A*+*B*	*A*+*B*	No	No	No	No
Public key Kerberos (4.28)	*S*	*A*+*B*	*B*	*A*+*B*	Yes	No
(I)MSR (4.30)	*B*	*B*	*A*+*B*	*A*	Yes	No
Improved Beller–Yacobi (4.32)	*A*	*A* + *B*	*A* + *B*	*A*+*B*	No	No
TMN (4.34)	*B*	*B*	No	No	Yes	No
AKA (4.35)	*A*+*B*	*A*+*B*	No	No	Yes	No

Because key transport protocols cannot in general provide it, we do not include forward secrecy in the table. The same applies to resistance to key compromise impersonation. If we restrict to basic key transport protocols, where all keying material is chosen by one party and sent encrypted with the other party's public key, then neither forward secrecy nor KCI resistance can be provided. This is because knowledge

of just one party's private key is then sufficient to obtain the session key. Note that some of the protocols which we examined in this chapter, such as Protocols 4.22 and 4.25, do not fall into this category. The session key may be based on private inputs of both parties, as in Protocol 4.22; such a protocol is really key agreement, not key transport. Also the session key may be only indirectly encrypted with the receiver's long-term key, as in Protocol 4.25.

The protocols in the ISO/IEC 11770-3 standard have benefited from the formal analysis by Cremers and Horvat [229] and have been updated to avoid the problems they identified. For many applications they are perhaps the best choice of protocol if key transport is needed.

As remarked at the end of Sect. 4.3.1, the intuitive key confirmation property of 11770-3 Mechanism 6 has been shown not to hold in general, if optional text fields are implemented. We also recall that Chen and Mitchell [197] have pointed out that all protocols in the ISO/IEC 11770 and 9798 series of standards are potentially vulnerable to parsing ambiguity attacks unless appropriate precautions are taken (see Sect. 1.4.7).

4.4 Conclusion

The ISO/IEC 11770-3 standard specifies a variety of key transport protocols using asymmetric cryptography. Blake-Wilson and Menezes provided a proof for a simplified version of one of these which provides extra confidence in their security. Later, Cremers and Horvat [229] provided a formal analysis of all the ISO/IEC 11770-3 protocols and found some weaknesses. They proposed changes to avoid the problems by adding message tags preventing interchanging of messages, and by preventing arbitrary usage of optional text fields. It would be useful to have security proofs for each of the protocols. The TLS protocol, discussed in detail in Chap. 6, has been widely scrutinised and provides an alternative to some of the ISO/IEC protocols.

The other key transport protocols examined in this chapter all seem to have some problems, and they cannot be recommended over the ISO standard solutions. We think that the study of these protocols can nevertheless be instructive in understanding typical mistakes in protocol design. Basin, Cremers and Horvat [59] have proposed improvements to several existing protocols to avoid the problem of key compromise impersonation.

Recent attention in the research community has focused on key agreement rather than key transport. One reason for this is that key transport protocols do not usually provide forward secrecy, which is often possible with key agreement. Key agreement is the topic of the following chapter.

5

Key Agreement Protocols

5.1 Introduction

Key agreement, as the name implies, is a process in which principals cooperate in order to establish a session key. Amongst the class of public key protocols for key establishment without a server, key agreement has become much more popular than key transport in recent years. There is an intuitive feeling that key agreement is 'fairer' than key transport and can result in higher-quality random keys than key transport can. In addition, by basing key agreement on the Diffie–Hellman protocol, forward secrecy can often be achieved. We will consider these points further below. Notice that key agreement does not have to use public key cryptography, but most examples do so. In this chapter we look only at key agreement based on public key cryptography; some examples of key agreement using symmetric cryptography were discussed in Chap. 3.

The definition of key agreement given by Menezes *et al.* [550] is as follows:

> A key agreement protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these (ideally) such that no party can predetermine the resulting value.

A similar definition is given in the international standard ISO/IEC 11770-3 [383], although it insists that neither party can predetermine the shared secret. In this chapter we are concerned with key agreement between any two principals A and B. The general format of such protocols requires each principal to select an independent input to the key. For our two principals, these will be denoted r_A and r_B, respectively. The principals will then send each other messages depending on r_A and r_B, and possibly depending on other values too.

The plan for the rest of the chapter is as follows. Before looking at specific key agreement protocols, we consider some of the special properties and attacks that can apply to them. Of course, all the general attacks on key establishment protocols discussed in Chap. 1 are relevant too.

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_5

Section 5.2 looks at the basic Diffie–Hellman protocol and emphasises its properties and limitations. Then, in Sect. 5.3, we examine in detail a set of protocols based on Diffie–Hellman which are known as the MTI protocols. This set of protocols was designed relatively early and serves to illustrate many of the properties of and potential attacks on key agreement protocols. Section 5.4 includes a number of more recent protocols whose exchanged messages are identical to those of Diffie–Hellman. Extra information, particularly public and private keys, is used in the calculation of the shared secret. Section 5.5 is devoted to protocols that add extra authentication information to the exchanged messages rather than (or in addition to) the definition of the shared secret. International standard ISO/IEC 11770-3 specifies key agreement protocols in an abstract format. These are summarised in Sect. 5.6. Up to this point all the detailed protocols have been described in the setting of the multiplicative group of integers modulo a prime p. In Sect. 5.7 we list some alternative groups that have been proposed for the Diffie–Hellman protocol. Finally, we look at some key agreement protocols that do not use the Diffie–Hellman technique in Sect. 5.8.

5.1.1 Key Derivation Functions

There are usually two stages to forming the session key.

1. The random inputs r_A and r_B, and possibly the long-term public/private keys, are combined to form a *shared secret*, $\mathbf{Z}$.
2. The session key $\mathbf{K}$ is formed from $\mathbf{Z}$, and possibly other inputs, using a *key derivation function*.

In different protocols, the key derivation function and its inputs will generally be different. A typical key derivation function is a one-way hash function of the shared secret and other data such as an algorithm identifier for the session key, a counter and public information about A and B. A generic key derivation function, known as KDF1, is specified in the IEEE P1363-2000 standard [372]; although KDF1 does specify the hash function to be used, the inputs to the function are left open.

Earlier protocols tended to focus only on the shared secret and leave the key derivation function unspecified. However, since security proofs started to be a major focus, it has become normal to be more specific regarding the properties of both the function and its inputs. Krawczyk [454] proposed a formal definition for secure key derivation functions and proposed a concrete construction. The ISO/IEC 11770-6 standard [384] specifies two-step key derivation functions applying a key extraction function followed by a key expansion function, the latter of which can be repeated to obtain further keys. The standard specifies one specific extraction function which can be combined with any one of four expansion functions. The functions all make use of a suitable MAC algorithm and follow the general pattern specified by Krawczyk [454].

Many protocol designers have simply used a hash function for key derivation, often modelled as a random oracle in the proofs. In our descriptions in this chapter,

we have usually included the inputs to the key derivation function when they are specified by the designers. However, we often focus on the shared secret **Z**, since this is often the simplest way to highlight the differences between many protocols.

5.1.2 Key Control

One potential benefit of key agreement is that each principal does not have to rely on any other party to generate appropriate keys. As long as neither party is malicious, it can often be guaranteed that the session key is sufficiently random if at least one of the principals is able to generate sufficiently random inputs. A related benefit is that principals can often be sure that the session key is fresh by ensuring that their own input is fresh. For this to be true, neither of the principals must be able to force the key to be any chosen value, otherwise one party could force use of an old key.

Key control is a term used to describe the extent to which principals have the ability to choose or influence the value of the shared key (or session key). As expressed in the definitions of key agreement given above, it is usually desired that neither principal can control the shared secret value.

In most practical situations one party will receive the random input of the other party before it has had to reveal anything about its own random input. This gives that party an 'advantage' in that it can effectively choose a number of bits of the session key. Mitchell *et al.* [559] have pointed out that, by choosing about 2^s random values, the party with the advantage can effectively choose any s bits of the key by generating new keys until the desired bits occur. Although s will typically be much less than the total key length, it is important to be aware of this possibility in assessing the properties of a key agreement protocol.

Mitchell *et al.* pointed out that such an attempt to control the key can be prevented by ensuring that both parties fix their random input before information about the other party's input is known. Ways to achieve this include strict use of timeouts and use of a third party, but the most reasonable seems to be to have the first party send a hash of its random input as a *commitment*, which will be opened in a later message after the second party's random element is received. A major drawback of this approach is that an extra message is required in the protocol.

A lack of strict key control applies to most published key agreement protocols. At present, designers do not seem to be concerned enough about it to propose countermeasures.

5.1.3 Unknown Key-Share Attacks

Unknown key-share attacks are applicable in a security model that allows malicious insiders. The aim of the adversary C is to make one principal, say A, believe that the session key is shared with C when it is in fact shared with a different principal, B. The adversary need not, and usually does not, obtain the session key. Nevertheless, such an attack is profitable to the adversary in an application where B will deliver some information of value (such as electronic cash) to principal A. Since A believes

the session key is shared with C, credit for this deposit will rest with C. (This means that this key is not good to establish credit in the sense of Abadi [2].)

Unknown key-share attacks seem to have first been described by Diffie *et al.* [253]. Their importance is somewhat controversial, since there are some general methods that can be used to avoid them. Furthermore, the assumptions made in many of the proposed attacks are rather unusual. A common scenario is for the adversary to obtain a public key certificate that has the same public key value as another principal. (The adversary will not know the corresponding secret key.) There are a number of methods that can help to defeat the attack.

- Certifiers of public keys can ensure that each entity is in possession of the corresponding private key before a certificate is issued. Some authors assume that this precaution is always taken and do not regard an unknown key-share attack as valid if the adversary does not know the private key corresponding to the certified public key.
- Key confirmation can often defeat the attack. The confirmation messages should include the identities of both principals so that the key is confirmed to be held by specific claimed entities, rather than known only to some unidentified party.
- A general method to ensure that unknown key-share attacks do not apply is to include both principal identities within the key derivation function. As long as the function used is collision-resistant then A, if she believes the key is shared with C, will not derive the same session key as B, who believes the session key is shared with A. Somewhat surprisingly, Blake-Wilson and Menezes [110] deprecated this method of avoiding unknown key-share attacks on the grounds that the requirements of key derivation functions have not been widely studied.

5.1.4 Classes of Key Agreement

The most well-known technique used in key agreement protocols is Diffie–Hellman key exchange [252]; indeed, sometimes key agreement is even used synonymously with the Diffie–Hellman technique. There are some special advantages of basing key agreement on Diffie–Hellman.

- Most of the protocols can be generalised to work in any Abelian (commutative) group. This allows a flexible choice of groups, including some that are particularly efficient in terms of computational and storage requirements.
- Many protocols based on Diffie–Hellman have the forward secrecy property, which is costly to achieve any other way. (However, several examples in this chapter show that basing a protocol on Diffie–Hellman does not guarantee that forward secrecy is achieved.)

It should be remembered that there are other ways of designing key agreement protocols apart from using the Diffie–Hellman primitive. A widely used alternative is to use a one-way function of user inputs to derive the session key. There can be advantages in this approach too.

- There can be computational savings over Diffie–Hellman by reducing (or eliminating) the number of expensive exponentiations.
- Keys can be guaranteed to be random even if one input becomes known – a property lacking in Diffie–Hellman-based protocols.

In Sect. 5.8, we will examine protocols using encryption and key encapsulation in place of Diffie–Hellman.

5.1.5 Protocol Compilers for Key Agreement

Our focus in most of this chapter is on concrete key agreement protocols proposed in standards and the academic literature. There are also generic methods available to construct protocols from other primitives or from protocols with weaker security properties. These are often called *protocol compilers*. Here we mention some compilers designed for two-party key agreement. In Chap. 9, we will also describe a compiler due to Katz and Yung designed for group key agreement, which, of course, includes two-party key agreement as a special case.

Jager *et al.* [391] designed compilers to combine key agreement protocols secure against passive adversaries with dedicated authentication protocols in such a way as to achieve protocols secure against active adversaries. They used a BR-style model but did not deal with forward secrecy. Their first compiler requires only standard model arguments, while their second is more efficient but uses a random oracle in the proof.

Li *et al.* [488] presented two compilers which take in a protocol Π, secure against passive adversaries, and add either (deterministic) signatures or CCA-secure encryption of the transcript of Π. They provided a formal analysis in a BR-style model including state reveals and forward secrecy (it was required that Π did not use long-term keys). One potentially useful aspect of these compilers is that they do not require any modification to the input protocol Π, allowing them to be applied to existing implementations of Π without modification.

Generally, such compilers do not achieve the same efficiency as the dedicated protocols which we examine in this chapter. However, they allow flexible combination of protocols and the ability to plug in new protocols whenever they become available. We emphasise also that the above compilers are not limited to application to Diffie–Hellman-based protocols, but apply to any key agreement protocol.

5.2 Diffie–Hellman Key Agreement

Diffie–Hellman key agreement was published in 1976 [252].[1] This elegant and simple construction has been the basis for a vast range of protocols, but on its own lacks any authentication. Later in this chapter we examine different ways to add authentication to produce more effective key agreement protocols. In this section we

[1] It seems that the idea was invented previously in 1974 but was not made public at that time [734].

introduce Diffie–Hellman and mention some protocols in which one or both of the Diffie–Hellman inputs are fixed, and which therefore lack some of the usual benefits of the technique.

In the basic Diffie–Hellman protocol, two principals A and B agree publicly on an element g that generates a multiplicative group $\mathbb{G}$. They then select random values r_A and r_B, respectively, in the range between 1 and the order of $\mathbb{G}$. A calculates $t_A = g^{r_A}$ and B calculates $t_B = g^{r_B}$, and they exchange these values as shown in Protocol 5.1. The shared secret is $\mathbf{Z} = g^{r_A r_B}$. This value can be calculated by both A and B owing to the commutative property of exponentiation: $\mathbf{Z} = t_A^{r_B} = t_B^{r_A}$.

Shared information: Generator g of $\mathbb{G}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{t_B}$	$\mathbf{Z} = t_A^{r_B}$

Protocol 5.1: Diffie–Hellman key agreement

Diffie–Hellman key agreement was originally described in the multiplicative group $\mathbb{Z}_p^*$ of non-zero integers modulo a large prime p. It has become more usual to define the group $\mathbb{G}$ in which the protocol takes place to be a subgroup of $\mathbb{Z}_p^*$ of prime order q. (Note that the order of $\mathbb{Z}_p^*$ is $p-1$, which cannot be prime.) There are two potential advantages of this. Firstly, several attacks (to be described shortly) can be avoided. Secondly, the size of the group $\mathbb{G}$ can usually be made much smaller than $\mathbb{Z}_p^*$, which results in computational savings. Typical sizes in use today are 2048 bits for the length of p and 256 bits for the length of q. Several other algebraic groups have been proposed as the setting for Diffie–Hellman key exchange. Examples are given in Sect. 5.7. In particular, elliptic curve groups are popular today. To enable a uniform presentation, all the Diffie–Hellman-based protocols in this chapter are described with $\mathbb{G}$ as a subgroup of $\mathbb{Z}_p^*$, even in cases when the protocol designers have prescribed other groups.

Diffie–Hellman is secure against passive eavesdroppers on the widely accepted assumption that it is infeasible to recover $g^{r_A r_B}$ from the values of g^{r_A} and g^{r_B}. This is often referred to as the *computational Diffie–Hellman* (CDH) assumption. Breaking the Diffie–Hellman problem is clearly no harder than solving the discrete logarithm problem, since by finding the discrete logarithm of either of the exchanged values the Diffie–Hellman key can be found. It is a long-standing open question as to whether the Diffie–Hellman problem is really as hard as the discrete logarithm problem, de-

spite considerable research on the topic [403, 527]. In formal analysis of Diffie–Hellman-based protocols, it is often possible only to obtain a proof based on the generally stronger *decisional Diffie–Hellman* (DDH) assumption. This states that it is a hard problem to distinguish between a genuine Diffie–Hellman triple (g^x, g^y, g^{xy}) and a triple (g^x, g^y, g^z) for random exponents x, y and z.

The fundamental limitation of the basic Diffie–Hellman protocol is that there is no authentication of the messages sent. This is illustrated by the well-known 'man-in-the-middle' attack in which the adversary C masquerades as B to A and masquerades as A to B. Attack 5.1 shows how both A and B complete a normal run, but both share keys with C, namely $g^{r_A r_C}$ and $g^{r_C r_B}$, respectively.

A		C		B
$r_A \in_R \mathbb{Z}_q$				
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_C \in_R \mathbb{Z}_q$		
		$t_C = g^{r_C}$	$\xrightarrow{t_C}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{t_C}$	$\mathbf{Z}_{AC} = t_A^{r_C}$	$\xleftarrow{t_B}$	$t_B = g^{r_B}$
$\mathbf{Z} = t_C^{r_A}$		$\mathbf{Z}_{CB} = t_B^{r_C}$		$\mathbf{Z} = t_C^{r_B}$

Attack 5.1: Man-in-the-middle attack on basic Diffie–Hellman

The shared secret $g^{r_A r_B}$ derived in basic Diffie–Hellman is called an *ephemeral* Diffie–Hellman key, since it depends only on randomly chosen values and lasts only until the session key is derived. In contrast, if A and B exchange their respective long-term public keys $y_A = g^{x_A}$ and $y_B = g^{x_B}$ then both can calculate the value $S_{AB} = g^{x_A x_B}$. This is often called a *static* Diffie–Hellman key, since it does not depend on any random input. As we will see in Sect. 5.4, it is a common method to design protocols based on Diffie–Hellman by mixing the ephemeral and static values in such a way as to obtain the desired properties.

Static Diffie–Hellman is one example, indeed the most widely seen example in practice, of *non-interactive key exchange* (NIKE). A NIKE protocol enables principals to derive a shared secret without any protocol messages being exchanged. (Of course, the principals still need to obtain the static keys by some means, but for NIKE there are no direct protocol messages between principals.) Another example of NIKE is the SOK identity-based protocol (see Sect. 7.1.3). Freirre *et al.* [284] proposed several different formal security definitions, based on different assumptions regarding what long-term keys may be registered by the adversary. Their definition requires the specification of exactly how the session key should be derived from the shared secret. Use of static Diffie–Hellman, or NIKE in general, on its own to derive session keys falls outside the main focus of this chapter, since any such protocol does not allow generation of new session keys. However, NIKE can still be a useful building

block in the generation of key agreement protocols. We mention one example of this in Sect. 5.5.1.

The static Diffie–Hellman key can be used in a very simple protocol by incorporating it together with a fresh value in a one-way function. Suppose that k is a fixed key derived from S_{AB}. Rueppel and van Oorschot [637] noted that k could be used to transport a random session key $\mathbf{K}$ by sending $\{\mathbf{K}\}_k$ or, alternatively, the session key could be defined as $\mathbf{K} = \mathsf{MAC}_K(r)$, where r is a sequence number or nonce sent in cleartext. Such protocols could provide implicit key authentication, but fail to provide some of the typical advantages of key agreement. In particular, there is no joint key control and neither forward secrecy nor resistance to key compromise impersonation is provided.

The notation used in this section is included in Table 5.1 and will be used throughout this chapter for describing Diffie–Hellman-based protocols. We will also continue to use notation introduced in Table 4.1.

Table 5.1: Notation used throughout Chap. 5

p	A large prime (typically between 1024 and 3072 bits).
q	A prime (typically between 160 and 256 bits) with $q \mid p-1$.
$\mathbb{G}$	A group whose order divides $p-1$. $\mathbb{G}$ is often a group of order q, and may be a subgroup of $\mathbb{Z}_p^*$ or an elliptic curve group.
g	A generator of $\mathbb{G}$.
r_A, r_B	Random integers, typically of the same size as the order of $\mathbb{G}$, chosen by A and B, respectively. Sometimes we will call these *ephemeral private keys*.
t_A, t_B	Ephemeral public keys, $t_A = g^{r_A}$ and $t_B = g^{r_B}$. All computations take place in $\mathbb{Z}_p$.
x_A, x_B	The private long-term keys of A and B, respectively.
y_A, y_B	Long-term public keys of A and B, $y_A = g^{x_A}$ and $y_B = g^{x_B}$. These public keys will have to be certified in some standard way that we usually ignore.
$\mathbf{Z}$	The shared secret calculated by the principals. This may be computed by the principals in different ways.
$\mathbf{K}$	The derived session key.
S_{AB}	The static Diffie–Hellman key of A and B, $S_{AB} = g^{x_A x_B}$.
N_A, N_B	Nonces chosen by A and B, respectively.
$H(.)$	A one-way hash function. Certain protocols may require specific properties and may specify particular functions.
$x \in_R X$	The element x is chosen uniformly at random from the set X.
$F \stackrel{?}{=} G$	Verify that F and G evaluate to the same value.

5.2.1 Small Subgroup Attacks

Small subgroup attacks seem to have been first recognised by Vanstone (attributed in later descriptions [478, 594]). The idea of small subgroup attacks is to exploit the structure of the group $\mathbb{G}$ in which Diffie–Hellman key agreement takes place. If the order of $\mathbb{G}$ is composite then $\mathbb{G}$ will have subgroups; furthermore, if g^{r_A} lies in some subgroup then so does $g^{r_A r_B}$. The idea of the small subgroup attack is to force the shared secret to lie in a small set. Then there will be relatively few possible values available for the session key, which will help the adversary and possibly allow exhaustive search.

One way to avoid small subgroup attacks is to make $\mathbb{G}$ of prime order. This is frequently done by choosing g to have prime order q, where $q|p-1$. In this case the only proper subgroup of $\mathbb{G}$ consists of the single identity element. In order to avoid attacks, it may still be necessary to check that received elements do lie in the correct group (and are not equal to the identity). See Sect. 5.3.1 for a specific example of a small subgroup attack.

A related type of attack, first proposed by Lim and Lee [492], exploits small subgroups that are outside the subgroup generated by $\mathbb{G}$. In these attacks the adversary, who is an insider, sends the recipient a value that should be in the group $\mathbb{G}$, but is mixed with a value outside $\mathbb{G}$. This allows the adversary to gain information about the value of the victim's private key. This type of attack can still work when $\mathbb{G}$ is a prime-order subgroup of $\mathbb{Z}_p^*$. It can be prevented if the principals ensure that all received elements are inside $\mathbb{G}$. Further details, including a specific example, are given in Sect. 5.3.3.

Zuccherato [784] has summarised the situations in which small subgroup attacks are a threat, and proposed different ways in which they can be avoided.

5.2.2 ElGamal Encryption and One-Pass Key Establishment

Before considering protocols in which both principals contribute a random value, we first look at the situation where only one principal does so. These protocols can be considered as halfway between using static Diffie–Hellman keys and the inclusion of ephemeral keys. These protocols are useful when it is possible only to have communications in one direction; a typical application scenario would be secure electronic mail.

ElGamal encryption [267] was not conceived as a key establishment protocol, yet we can view it in this manner. The sender A forms a shared secret using her random input r_A in combination with B's long-term y_B by calculating $\mathbf{Z} = y_B^{r_A}$. On receipt of an encrypted message and the ephemeral public key $t_A = g^{r_A}$, principal B is able to reconstruct the same secret $\mathbf{Z} = t_A^{x_B}$ and so decrypt the message. Evidently B receives no authentication regarding the session key and cannot even check the freshness of $\mathbf{Z}$; however, A does obtain implicit key authentication.

Protocol 5.2 shows one-pass key establishment as proposed by Agnew *et al.* [22]. The static Diffie–Hellman key is used together with a nonce k_A to form the shared

Shared information: Static Diffie–Hellman key, S_{AB}.

A		B
$r_A, k_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}, s_A = y_B^{r_A} \cdot k_A$		
	$\xrightarrow{t_A, s_A}$	$k_A = s_A / t_A^{x_B}$
$\mathbf{Z} = S_{AB}^{k_A}$		$\mathbf{Z} = S_{AB}^{k_A}$

Protocol 5.2: Agnew–Mullin–Vanstone protocol

secret. The nonce k_A is sent encrypted from A to B using B's public key y_B: the ElGamal encryption algorithm is used for this purpose.

The shared secret is $\mathbf{Z} = S_{AB}^{k_A}$. Implicit key authentication is provided for both parties, since knowledge of either x_A or x_B is required to form $\mathbf{Z}$. However, B has no way of knowing that the shared secret is fresh. Entity authentication is not achieved for either party, and neither is forward secrecy nor protection against key compromise impersonation. The encryption of k_A prevents the adversary from using a known $\mathbf{Z}$ value to obtain S_{AB} and mounting a future active attack.

Nyberg and Rueppel [585, 587] investigated ways of incorporating message recovery into signatures based on the discrete logarithm. As an application of this technique, they suggested the use of ElGamal encryption of a signed session key as a one-pass key establishment protocol. Protocol 5.3 is their initial protocol [585]. If the protocol runs correctly then both A and B calculate the shared secret $\mathbf{Z} = g^{r_A x_B}$.

A		B
$r_A, k \in_R \mathbb{Z}_q$ $\mathbf{Z} = y_B^{r_A}$		
$r = g^{r_A - k}, s = k - x_A r \bmod q$	$\xrightarrow{r, s}$	$\mathbf{Z} = (g^s y_A^r r)^{x_B}$

Protocol 5.3: Original Nyberg–Rueppel protocol

Later, Nyberg [584] pointed out that it is possible for an adversary who finds an old $\mathbf{Z}$ value to replay the old r value and, by replacing the corresponding s with $s+u$, to establish a new key $\mathbf{K}' = \mathbf{K} \cdot y_B^u$. She therefore designed the enhanced Protocol 5.4. This employs a time-varying parameter, which could be a counter. Neither of these protocols provides forward secrecy, since knowledge of x_B allows the adversary to compute $\mathbf{Z}$ in the same way as B.

Shared information: Time-varying parameter t.

A		B
$r_A \in_R \mathbb{Z}_q$		
$\mathbf{Z} = y_B^{r_A}$		
$r = g^{r_A}, r' = H(\mathbf{Z}, t)$		
$s = r_A - x_A r' \bmod q$	$\xrightarrow{r,s}$	$\mathbf{Z} = r^{x_B}$
		$r' = H(\mathbf{Z}, t)$
		$r \stackrel{?}{=} g^s y_A^{r'}$

Protocol 5.4: Revised Nyberg–Rueppel protocol

Other schemes for authenticated message exchange that are suitable for one-pass key establishment have been proposed by Horster *et al.* [365]. (A revised version of this paper, not formally published, includes an attack on some of the schemes, which was found by C. H. Lim.) Schemes for combined encryption and signature, known as *signcryption*, due originally to Zheng [779], can also be used for this purpose. Gorantla *et al.* [324] formally analysed this connection and showed that, with suitable conditions, secure signcryption schemes can be converted to secure one-pass key establishment protocols and vice versa.

In Sect. 5.4, we will examine several two-pass key agreement protocols. Most of these can be converted into one-pass key establishment protocols by replacing the random input of the receiving party with that principal's long-term secret. Blake-Wilson and Menezes [109] discussed this procedure and illustrated its application.

Chalkias *et al.* [187] proposed a dedicated one-pass protocol and claimed stronger properties than, for example, one-pass HMQV (see Sect. 5.4.6) particularly with respect to KCI resistance. This interpretation depends on the definition of what constitutes a KCI attack. As Chalkias *et al.* discussed [187], an adversary who has the long-term key of recipient B can always replay any one-pass protocol run with B and obtain the session key. In this sense, no one-pass protocol can achieve KCI resistance. The protocol of Chalkias *et al.* aims to limit this attack by preventing the replay from being used to allow the adversary to masquerade as *different* entities from the one that originally sent the replayed message. Their protocol is relatively computationally expensive, requiring three exponentiations on the sender side and an elliptic curve pairing on the receiver side. It also uses a timestamp indextimestamp which many designers prefer to avoid.

5.2.3 Lim–Lee Protocol Using Static Diffie–Hellman

We have already seen a one-pass protocol that uses the static Diffie–Hellman key in Protocol 5.2 above. Lim and Lee [491] proposed a three-pass protocol using the static Diffie–Hellman key together with random inputs from both principals. This

allows both parties to be sure that the key is fresh. In Protocol 5.5, the symmetric key K is a static key derived in some (unspecified) way from the static Diffie–Hellman key S_{AB}.

Shared information: Shared key K derived from S_{AB}.

A		B
$N_A \in_R \mathbb{Z}_q$	$\xrightarrow{\{N_A\}_K}$	
	$\xleftarrow{\{ID_A, N_B\}_{K \oplus N_A}}$	$N_B \in_R \mathbb{Z}_q$
$\mathbf{Z} = N_A \oplus N_B$	$\xrightarrow{\{N_A\}_{K \oplus N_A}}$	$\mathbf{Z} = N_A \oplus N_B$

Protocol 5.5: Lim–Lee protocol using static Diffie–Hellman

Lim and Lee suggested that the session key may be defined either as $\mathbf{K} = N_A \oplus N_B$ or as $\mathbf{K} = K \oplus N_A \oplus N_B$. As long as A and B both use random inputs, this ensures the freshness of the key. However, with either of these choices for $\mathbf{K}$ it is possible for B to completely control the session key value. As is usual with protocols using only static keys, Protocol 5.5 does not provide even partial forward secrecy, since knowledge of one of the long-term keys is sufficient to find S_{AB}. Key compromise impersonation is possible too.

5.3 MTI Protocols

Matsumoto, Takashima and Imai (MTI) [526] showed in 1986 how to define three classes of authenticated key agreement protocols. These incorporate authentication into the Diffie–Hellman exchange in a very elegant manner by combining the long-term and ephemeral inputs into a single equation. Although many protocols have been designed using the same ideas as in the MTI protocols, in their original form the protocols have various shortcomings. Nevertheless, we look at them in some detail in this section for two reasons. Firstly, a detailed knowledge of these protocols will be very helpful in understanding the many protocols based on them. Secondly, they form a useful vehicle to explain many types of attack on key agreement protocols.

The MTI protocols are divided into three families: A, B and C. Protocol 5.6 shows the basic protocol of type A, denoted A(0). In the original specifications the subgroup $\mathbb{G}$ in which Diffie–Hellman exchange takes place is equal to the whole of $\mathbb{Z}_p^*$; as we will see below, a better choice is to make $\mathbb{G}$ a subgroup of prime order q. When both principals follow the protocol, the shared secret is $\mathbf{Z} = g^{x_A r_B + x_B r_A}$.

If we accept that knowledge of either x_A or x_B is required in order to compute $\mathbf{Z}$, then implicit key authentication follows. Below we will mention a number of

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{x_A} y_B^{r_A}$	$\xleftarrow{t_B}$	$\mathbf{Z} = t_A^{x_B} y_A^{r_B}$

Protocol 5.6: MTI A(0) protocol

potential attacks on the scheme that show that this assumption is not always valid. As with any two-pass key agreement protocol, key confirmation is not achieved in the basic protocol.

Matsumoto *et al.* [526] showed that there is an infinite sequence of protocols with a related format. Protocol 5.7 shows the A(k) class of protocols, defined for any integer k. When both principals follow the protocol, the shared secret is $\mathbf{Z} = g^{x_A r_B x_B^k + x_B r_A x_A^k}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_A = g^{x_A^k r_A}$	$\xrightarrow{z_A}$	$r_B \in_R \mathbb{Z}_q$
		$z_B = g^{x_B^k r_B}$
$\mathbf{Z} = z_B^{x_A} y_B^{x_A^k r_A}$	$\xleftarrow{z_B}$	$\mathbf{Z} = z_A^{x_B} y_A^{x_B^k r_B}$

Protocol 5.7: MTI A(k) protocol

This sequence of protocols constitutes one of the three classes of MTI protocols. These classes are all of the same basic format: they involve only two messages and achieve implicit key authentication but no key confirmation. Table 5.2 summarises the base ($k = 0$) protocols for each of the classes A, B and C; in this table, z_A and z_B are the messages sent from A to B and from B to A, respectively. The computational effort needed by each principal in protocols A(0) and B(0) is the same and consists of one exponentiation before message exchange and one multi-exponentiation to obtain the key. Protocol C(0) is slightly less complex, since an ordinary exponentiation only is required to obtain the key.

For each sequence, to obtain the k'th protocol from the base protocol the exponent in z_A must be multiplied by x_A^k and that in z_B by x_B^k. The equation used to compute $\mathbf{Z}$ by A must use $r_A x_A^k$ in place of r_A, and the equation for B to compute $\mathbf{Z}$ must use $r_B x_B^k$ in place of r_B. The exponents needed to calculate the shared secret for

Table 5.2: Base protocols for each class of MTI key exchange

Type	z_A	z_B	$\mathbf{Z}$	Computed by A	Computed by B
A(0)	g^{r_A}	g^{r_B}	$g^{x_A r_B + x_B r_A}$	$z_B^{x_A} y_B^{r_A}$	$z_A^{x_B} y_A^{r_B}$
B(0)	$y_B^{r_A}$	$y_A^{r_B}$	$g^{r_A + r_B}$	$z_B^{x_A^{-1}} g^{r_A}$	$z_A^{x_B^{-1}} g^{r_B}$
C(0)	$y_B^{r_A}$	$y_A^{r_B}$	$g^{r_A r_B}$	$z_B^{x_A^{-1} r_A}$	$z_A^{x_B^{-1} r_B}$

each protocol are shown in Table 5.3. Note that the protocols for negative k values are only well defined in the case that x_B is chosen to be invertible in $\mathbb{G}$ (for example, x_B must be prime to $p-1$ in the case that $\mathbb{G} = \mathbb{Z}_p^*$). The extra computational effort required by the variants with parameter k is one exponentiation with an exponent of size $|k|$.

Table 5.3: Exponent of shared secret for each MTI protocol

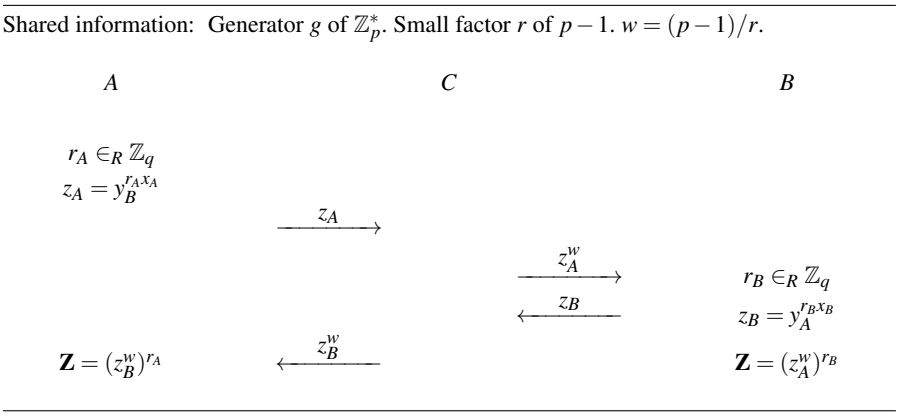
k	A(k)	B(k)	C(k)
-1	$x_A x_B^{-1} r_B + x_B x_A^{-1} r_A$	$x_A^{-1} r_A + x_B^{-1} r_B$	$x_A^{-1} r_A x_B^{-1} r_B$
0	$x_A r_B + x_B r_A$	$r_A + r_B$	$r_A r_B$
1	$x_A x_B r_B + x_B x_A r_A$	$x_A r_A + x_B r_B$	$x_A r_A x_B r_B$
2	$x_A x_B^2 r_B + x_B x_A^2 r_A$	$x_A^2 r_A + x_B^2 r_B$	$x_A^2 r_A x_B^2 r_B$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	$x_A x_B^k r_B + x_B x_A^k r_A$	$x_A^k r_A + x_B^k r_B$	$x_A^k r_A x_B^k r_B$

The MTI protocols provide an elegant and flexible approach to authenticated key agreement and have been the subject of considerable scrutiny in the research community. A number of attacks have been proposed which we examine now. It is worth noting that all known attacks can be prevented by use of suitable countermeasures.

5.3.1 Small Subgroup Attack

A small subgroup attack (see Sect. 5.2.1) applies to the MTI protocol sequence C(k) in the situation that the group $\mathbb{G}$ is the whole of $\mathbb{Z}_p^*$, as originally proposed. We suppose that the factorisation of $p-1$, which is the order of $\mathbb{G}$, is known to the adversary. The attack is easiest in the case that $p-1$ has a very small factor r; let us write $w = (p-1)/r$. The attack works by raising the exchanged messages to the

power w, which moves these elements into the small subgroup of $\mathbb{G}$ of order r. Attack 5.2 shows a small subgroup attack on MTI protocol C(1). The adversary C plays in the middle between A and B.



Attack 5.2: Small subgroup attack on MTI C(1)

The shared secret calculated by A and B is

$$\mathbf{Z} = g^{x_A r_B x_B r_A w}.$$

Since this is an element in the small subgroup, C can easily find the shared secret by exhaustive search and verify it from subsequent use in communications between A and B. Notice that in the extreme case when $r = 1$ it follows that $w = p - 1$, and so the element received by both A and B is 1.

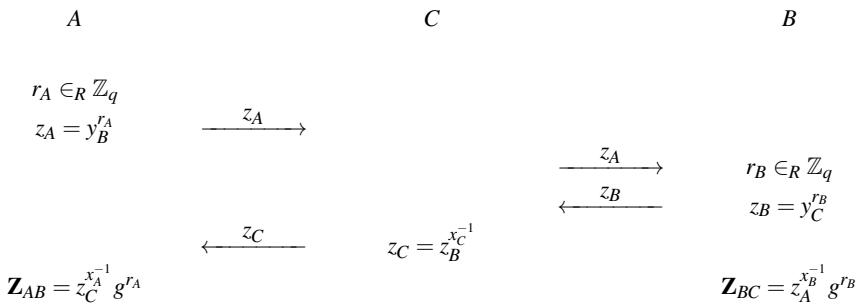
This small subgroup attack can be prevented by making $\mathbb{G}$ a subgroup of prime order q. In addition, it is necessary to check that the received elements really are in the group $\mathbb{G}$ and are not equal to the identity.

5.3.2 Unknown Key-Share Attacks

Menezes *et al.* [551] discovered unknown key-share attacks on all the classes of MTI protocols. All the attacks require the adversary C to obtain a certificate for a long-term key y_C which is related to the public key of A by the equation $y_C = y_A^{x_C} = g^{x_A x_C}$. This means that C cannot know the private key $x_A x_C$ corresponding to the public key y_C. Attack 5.3 shows an unknown key-share attack on the MTI protocol B(0).

The shared secret calculated by A is $(y_C^{r_B x_C^{-1}})^{x_A^{-1}} g^{r_A} = g^{r_B + r_A}$, while B calculates $(y_B^{r_A})^{x_B^{-1}} g^{r_B} = g^{r_A + r_B}$ to get the same value. Although A and B both have the same session key, A believes it to be shared with B, while B believes it to be shared with C. There are several ways to avoid the attack, including:

Shared information: Public key of C is $y_C = y_A^{x_C}$.

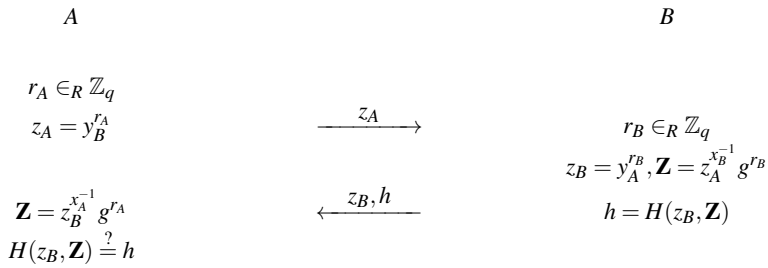


Attack 5.3: Unknown key-share attack on MTI B(0)

- having certification authorities check that principals know the corresponding private key before issuing a public key certificate;
- including the principal identities in the key derivation function.

Menezes *et al.* [551] suggested adding to the message returned by B a hash of the key generated and B's randomised reply. A must check this on receipt of message 2 and abort the protocol if the check fails. Protocol 5.8 shows MTI B(0) modified in this way. Just and Vaudenay [407] pointed out that the protocol is still insecure if degenerate values such as 0 or 1 are accepted by A. For example, in Protocol 5.8 the adversary C can masquerade as B if A will accept the response $(u_B, h) = (0, H(0,0))$; the shared secret is calculated by A as 0 and is known to C.

Shared information: Hash function H.



Protocol 5.8: Modified MTI B(0) protocol

5.3.3 Lim–Lee Attack

Lim and Lee [492] devised ingenious attacks on interactive protocols that work in prime-order subgroups. Their attack is applicable to MTI variants in which $\mathbb{G}$ is a prime-order subgroup. As already mentioned, this is desirable in order to avoid small subgroup attacks on C(k) protocols and is also very beneficial in terms of the savings in computational requirements due to smaller exponent sizes.

The idea of the attack is that the adversary C will engage in a run of the protocol with the victim B. For B the run will seem normal, but in the first message C sends a value that is not in the group $\mathbb{G}$; consequently, the key calculated by B will give away information about B's long-term secret key x_B. In an interesting echo of the prime-order subgroup attack discussed in Sect. 5.3.1, this requires that $(p-1)/q$, rather than q itself, contains many small factors. Attack 5.4 illustrates the procedure on MTI protocol A(0).

Information known to C: β of small order r with $r|(p-1)/q$.

C		B
$r_C \in_R \mathbb{Z}_q$		
$t_C = g^{r_C}$	$\xrightarrow{\beta t_C}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z}_{CB} = t_B^{x_C} y_B^{r_C} = \mathbf{Z}_{BC}/\beta^{x_B}$	$\xleftarrow{t_B}$	$\mathbf{Z}_{BC} = (\beta t_C)^{x_B} y_C^{r_B}$
Test each value of x_B mod r		

Attack 5.4: Lim–Lee attack on MTI A(0)

Suppose that β is an element whose order r is a small factor of $(p-1)/q$. The shared secret is calculated by B as $\mathbf{Z}_{BC} = (\beta t_C)^{x_B} y_C^{r_B}$. Now, since $t_C^{x_B} = y_B^{r_C}$ and $y_C^{r_B} = t_B^{x_C}$, C can calculate $\mathbf{Z}_{BC}/\beta^{x_B}$ and, since there are only r possible values for β^{x_B}, C can try out each of these in turn. There are a number of ways that C can verify whether the correct value has been found. One is if a check function is returned by B as in the modified MTI protocol of Menezes *et al.* described in Protocol 5.8 above. Another possibility is if A waits for an authenticated message sent by B following the protocol. Whatever the method used, having identified the value of $\mathbf{Z}_{BC}$, C can obtain β^{x_B}, which reveals the value of x_B mod r. To complete the attack, C repeats this procedure with new factors of $(p-1)/q$ in place of r until the value of x_B is obtained. A very similar attack applies to all the A(k) and B(k) MTI protocols.

Lim and Lee suggested two ways to avoid this attack. The first is that each recipient of a protocol message must check that the received value lies in $\mathbb{G}$. The cost of this is an exponentiation, which is a significant extra computational burden. The second method, which they favour, is to choose the prime p so that $(p-1)/q$ has no

small factors apart from 2. In this case the attack will give away one bit of information about the principal's secret: the adversary chooses $\beta = -1$, with order $r = 2$, to obtain $x_A \bmod 2$.

5.3.4 Impersonation Attack of Just and Vaudenay

Just and Vaudenay [407] found an impersonation attack on the MTI A(0) protocol on the assumption that the adversary can claim to have the same identity as the attacked principal A. This may be possible, for example, in an implementation where several devices share the same identity. Attack 5.5 shows how C can impersonate A, to A herself, by choosing t_C based on both the message received from A and a random input. At the end of the attack, both principals calculate the shared secret as $y_A^{r_C}$.

A		C_A
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_C \in_R \mathbb{Z}_q$
		$t_C = t_A^{-1} g^{r_C}$
$\mathbf{Z}_{AA} = t_C^{x_A} y_A^{r_A}$	$\xleftarrow{t_C}$	$\mathbf{Z}_{AC} = y_A^{r_C}$

Attack 5.5: Just–Vaudenay impersonation attack on MTI A(0)

The attack also extends to the whole of the MTI A(k) class of protocols. The adversary simply returns $z_C = z_A^{-1} g^{r_C}$, and A will calculate the shared secret as $\mathbf{Z}_{AA} = y_A^{r_C}$. A similar attack is also applicable to the MTI B(0) protocol, although it does not seem to be quite so strong in this case. If C masquerades as A to A and returns $z_C = z_A^{-1} \cdot g^{r_C}$ then A will calculate $\mathbf{Z}_{AA} = g^{x_A^{-1} r_C}$. Although C cannot calculate this value directly, C could replay it to get the same key any number of times. Also, if $r_C = 0$ is chosen, the shared secret becomes $\mathbf{Z}_{AA} = 1$. The obvious way to avoid all these attacks is for both A and B to ensure that the other has a different identity.

5.3.5 Triangle Attacks

Burmester [167] has shown that a 'triangle' attack can be mounted on MTI A(0), given certain assumptions about the release of session keys. The attack also applies to the MTI B(0) protocol, as well as to several protocols related to MTI A(0) which will be mentioned later. The format of the attack is as follows.

1. The adversary C eavesdrops on a session between A and B.
2. C starts separate sessions with A and B in which C uses information gained during step 1. (C does not obtain the session key used in these sessions as a result.)

3. C now induces A and B to reveal the keys used in the sessions between them. Because A and B believe that the session key should be known to C, this may be a reasonable assumption in certain application scenarios.
4. With this information, C can recover the key used in step 1.

It can be seen that this attack requires more assumptions than usual. It illustrates how difficult it is to consider all possible attacks on cryptographic protocols. A specific example of the triangle attack on the MTI protocol A(0) is as follows.

1. C records the values t_A and t_B used by A and B to form $\mathbf{Z} = g^{r_B x_A + r_A x_B}$.
2. C uses t_A as its input in a run with B. The agreed key calculated by B is $\mathbf{Z}' = g^{\tilde{r}_B x_C + r_A x_B}$ where $\tilde{t}_B = g^{\tilde{r}_B}$ is the value sent by B in this run. Similarly, C uses t_B in a session with A, which A uses together with its new value $\tilde{t}_A = g^{\tilde{r}_A}$ to generate a key $\mathbf{Z}'' = g^{\tilde{r}_A x_C + r_B x_A}$.
3. C somehow obtains $\mathbf{Z}'$ and $\mathbf{Z}''$.
4. C can now calculate $\mathbf{Z} = \mathbf{Z}' \cdot \tilde{t}_A^{-x_C} \cdot \mathbf{Z}'' \cdot \tilde{t}_B^{-x_C}$.

Burmester discussed ways to prevent the attack. Perhaps the simplest is the sensible precaution never to reveal previous session keys; good practice is to destroy session keys immediately after use. Another way is to insist on key confirmation before using a session key. A generic method to incorporate key confirmation into key agreement protocols is explained in Sect. 5.4.13.

5.3.6 Yacobi's Protocol

Yacobi [748] proposed a protocol identical to the MTI protocol A(0) except that a composite modulus is used. He supplied a proof of security of this protocol based on the idea that the exchanged messages are independent of the private keys and hence protocol runs may be perfectly simulated by anyone; therefore a passive attacker should gain nothing from observing a previous protocol run. Yacobi also claimed that this argument extends to the case of an active attacker who is able to obtain old session keys.

Subsequently, Desmedt and Burmester [241] pointed out that it cannot be assumed that a malicious protocol partner will act according to the protocol. They showed that the protocol must leak information unless the Diffie–Hellman assumption is false. This theoretical result shows that the proof is flawed, but does not result in any practical attack on the Yacobi or MTI protocols. Desmedt and Burmester proved that a modified protocol is indeed secure in this model. The messages exchanged are identical, but after sending $t_A = g^{r_A}$ and $t_B = g^{r_B}$ both principals engage in a zero knowledge protocol to show that they know r_A and r_B, respectively. Now the protocol will only complete if the principals act 'correctly', and so it can always be simulated.

As well as the drawback of the extra interaction required, the proof of security of Burmester and Desmedt's variant does not encompass all the possible actions of an adversary, and also partial information about the secret is not accounted for. Therefore it is difficult to say how useful the proof of security is. Indeed, a generic

unknown key-share attack applies (Attack 5.7), as does Burmester's triangle attack (see Sect. 5.3.5).

5.3.7 Forward Secrecy and Key Compromise Impersonation

Since Diffie–Hellman key exchange is known to have the attractive property of forward secrecy, it is natural to expect that the MTI protocols will have this property. However, this turns out not to be always the case.

The shared secret in MTI protocol A(0) is $g^{x_A r_B + x_B r_A}$, which can be found from knowledge of the long-term keys x_A and x_B and the exchanged messages. The same is true for all protocols in the sequence A(k). Again, for the protocols in the sequence B(k) it is not necessary to know either r_A or r_B to find the shared secret when both x_A and x_B are known; therefore these protocols do not provide forward secrecy either. (Both of the sequences A(k) and B(k) do provide partial forward secrecy, since compromise of only one of x_A and x_B does not reveal past session keys.)

The shared secret in the MTI protocol C(0) is $g^{r_A r_B}$, the ephemeral Diffie–Hellman key. The Diffie–Hellman assumption asserts that this secret is hard to compute without knowledge of either r_A or r_B. Noting that neither r_A nor r_B can be found from the exchanged messages, we conclude that C(0) does provide weak forward secrecy. A similar argument applies to all the protocols in the protocol sequence C(k).

We now turn our attention to key compromise impersonation. Consider again the MTI protocol A(0) and suppose that an adversary C has obtained the long-term secret x_A of A. In order to masquerade as B to A, C must send some value X in message 2 in such a way that C can find the value $X^{x_A} y_B^{r_A}$ calculated by A as the session key. If we assume that A will reject degenerate values such as 0 or 1 then the value cannot be found without knowledge of either x_B or r_A, neither of which is available to C. Therefore we deduce that protocol A(0) is not vulnerable to key compromise impersonation.[2] Similarly, in order to attack any protocol in the sequence A(k) the adversary must find an X such that $X^{x_A} y_B^{r_A x_A^k}$ can be found. Again, this requires knowledge of either r_A or x_B and so key compromise impersonation seems impossible.

A similar property holds for the protocol sequence B(k), the difference being that now the adversary cannot find g^{r_A} without knowledge of either x_B or r_A. Therefore we conclude that the MTI B(k) family is also not vulnerable to key compromise impersonation.

However, the situation with C(0) is different. Here C needs to find an X such that $X^{x_A^{-1} r_A}$ may be calculated. With knowledge of x_A, C can construct an appropriate value for X. This observation leads to Attack 5.6. After receiving C's reply, A calculates the shared secret as $y_B^{r_A r_C}$, which can also be calculated by C as $z_A^{r_C}$. Therefore protocol C(0) is vulnerable to key compromise impersonation. A similar attack also applies to any protocol in the sequence C(k).

In summary, we find that the sequences A(k) and B(k) provide protection against key compromise impersonation but do not provide even weak forward secrecy, while

[2] Although Just and Vaudenay [407] stated that A(0) is vulnerable to key compromise impersonation, this statement was later retracted.

Information known to C: Private key of A, x_A.

A		C
$r_A \in_R \mathbb{Z}_q$		
$z_A = y_B^{r_A}$	$\xrightarrow{z_A}$	$r_C \in_R \mathbb{Z}_q$
		$z_B = y_B^{x_A r_C}$
$\mathbf{Z} = z_B^{x_A^{-1} r_A}$	$\xleftarrow{z_B}$	$\mathbf{Z} = z_A^{r_C}$

Attack 5.6: Key compromise impersonation attack on MTI C(0)

the situation for the sequence C(k) is exactly the opposite. A natural question that arises is whether it is possible to achieve both properties at the same time. In Sect. 5.4, we will examine several protocols which achieve this.

5.4 Diffie–Hellman-Based Protocols with Basic Message Format

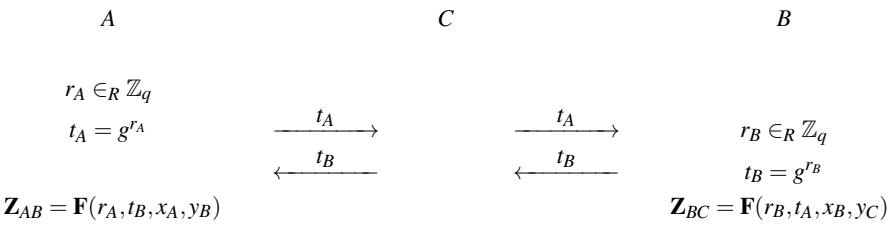
In this section, a number of protocols are examined for which the messages exchanged are the same as in the basic Diffie–Hellman protocol. Only two message passes are involved: A sends $t_A = g^{r_A}$ and B sends $t_B = g^{r_B}$. In contrast to the basic Diffie–Hellman protocol, the calculation of the shared key involves other components in order to achieve authentication of the key. This other information typically includes the public and private keys of the parties involved, so there is some function $\mathbf{F}$ such that A calculates the shared secret as $\mathbf{Z} = \mathbf{F}(r_A, t_B, x_A, y_B)$ while B calculates in the symmetrical fashion $\mathbf{Z} = \mathbf{F}(r_B, t_A, x_B, y_A)$. The MTI A(0) protocol discussed above fits into this class but has a number of potential weaknesses.

We remark that sometimes our presentation of protocols is not identical to that given in the original sources. In particular, we sometimes drop some fields such as plaintext identities and returned outgoing messages. Such fields may be useful and important in practical implementations but do not usually affect the security. Our aim is to help the reader to see similarities and differences between protocols by making the presentation as consistent as possible.

One general property is that *all* protocols in this class are vulnerable to the basic unknown key-share attack unless extra precautions are taken. Suppose that the adversary C can obtain a certificate for the public key used by B. Then C may sit between A and B and masquerade as B to A as shown in Attack 5.7.

Principal A calculates the shared secret as $\mathbf{Z}_{AB} = \mathbf{F}(r_A, t_B, x_A, y_B)$, while B calculates $\mathbf{Z}_{BC} = \mathbf{F}(r_B, t_A, x_B, y_C) = \mathbf{F}(r_B, t_A, x_B, y_B) = \mathbf{Z}$ since $y_C = y_B$. This attack is prevented if certificates are issued only to users who have shown that they know the private key corresponding to their public key. However, even this is not sufficient to prevent unknown key-share attacks in all cases. Other countermeasures which

Shared information: $y_C = y_B$.



Attack 5.7: Unknown key-share attack on generic protocol

can defeat such attacks were mentioned in Sect. 5.1.3, in particular using key confirmation (see Sect. 5.4.13) or including the principal identities in a key derivation function.

5.4.1 KEA Protocol

Goss [332] was awarded a US patent covering a protocol that is extremely similar to the MTI protocol A(0). The difference is that the shared secret is defined as $\mathbf{Z} = g^{x_A r_B} \oplus g^{x_B r_A}$ instead of $\mathbf{Z} = g^{x_A r_B} \cdot g^{x_B r_A}$ as in MTI A(0). The similarity carries over into many of the protocol properties: forward secrecy is not provided, but key compromise impersonation seems impossible. It is also vulnerable to Burmester's triangle attack (see Sect. 5.3.5).

The Key Exchange Algorithm (KEA) protocol was designed by the US National Security Agency for use with the SKIPJACK algorithm in key escrow implementations [572]. Originally classified, the protocol was released in June 1998 [580]. The KEA protocol is, like the Goss protocol, a variant of the MTI protocol A(0), as shown in Protocol 5.9. The differences are that the shared secret is defined as $\mathbf{Z} = g^{x_A r_B} + g^{x_B r_A} \bmod p$ and that there are extra checks in place. As with the Goss protocol, KEA inherits resistance to key compromise impersonation, but does not provide forward secrecy.

The KEA protocol specification includes exact parameter sizes: p is a 1024-bit prime and $\mathbb{G}$ is a subgroup of order q for a 160-bit prime q with $q|p-1$. There is also a particular key derivation function specified, which makes use of the SKIPJACK algorithm itself to form the 80-bit session key. As usual, we have omitted the processing of the certificates which is an essential part of the protocol.

Before calculating the shared secret, several checks should be made. A checks that the following are true.

1. t_B and y_B are integers greater than 1 and less than p.
2. $t_B^q \bmod p = 1$ and $y_B^q \bmod p = 1$, which ensures that t_B and y_B are both in $\mathbb{G}$.

B makes the analogous checks. If any check fails then the checking party halts. These checks prevent most of the attacks described for the MTI protocols. The specification

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$\mathbf{Z} = t_B^{x_A} + y_B^{r_A} \bmod p$		$\mathbf{Z} = t_A^{x_B} + y_A^{r_B} \bmod p$

Protocol 5.9: KEA protocol

also requires each party to check that $\mathbf{Z}$ does not equal 0 before accepting the key. However, Blake-Wilson and Menezes [109] have pointed out that this check seems unnecessary. If t_B and y_B are in the prime-order subgroup $\mathbb{G}$ then $t_B^{x_A} = -y_B^{r_A}$ would imply that -1 is an element of order 2 in $\mathbb{G}$. The checks already made before calculating $\mathbf{Z}$ ensure that t_B is in $\mathbb{G}$, and as long as y_B is a genuine public key it must also be in $\mathbb{G}$.

Lauter and Mityagin [477] analysed Protocol 5.9 and pointed out that it is vulnerable to Attack 5.7 because the principal identities are missing from the specific KEA key derivation function. They therefore revised the protocol with a key derivation function H and defined the session key as $\mathbf{K} = H(g^{x_A r_B}, g^{x_B r_A}, ID_A, ID_B)$. With this change, they renamed the protocol KEA+ and were able to provide a security proof in a Canetti–Krawczyk style model including weak forward secrecy and KCI resistance, assuming that H is a random oracle and that the gap Diffie–Hellman problem is hard. They also suggested a variant with key confirmation using MACs in the manner shown in Sect. 5.4.13.

The Open Protocol for Access Control Identification and Ticketing with privacY, or OPACITY, is a suite of security protocols designed for use with smart cards [645]. The OPACITY key agreement protocol has similarities with KEA, as well as other protocols such as the Unified Model protocol (see Sect. 5.4.4), in that the shared secret has inputs $t_B^{x_A}$ and $t_A^{x_B}$. Some authors [754] have commented on this similarity. However, the similarity applies more at a conceptual level than in the details. It should also be noted that the OPACITY specification describes much more than key agreement, including renegotiation of keys, channel security employing session keys, and privacy enhancements. We do not include details here, since it is hard to isolate the key establishment aspects from the other aspects. Dagdelen *et al.* [238] performed a detailed analysis of OPACITY and showed that it is secure in a Bellare–Rogaway-style model assuming the difficulty of the Gap Diffie–Hellman problem.

5.4.2 Ateniese–Steiner–Tsudik Protocol

Ateniese *et al.* [41, 42] examined key agreement for groups. Their general protocol will be examined in Chap. 6, but here we consider their two-party key agreement pro-

tocol which was used as a building block. Protocol 5.10 gives a slightly rearranged, but equivalent, description.

Shared information: Static Diffie–Hellman key S_{AB}.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$K = H(S_{AB})$	$\xleftarrow{t_B}$	$K = H(S_{AB})$
$\mathbf{Z} = t_B^{r_A \cdot K}$		$\mathbf{Z} = t_A^{r_B \cdot K}$

Protocol 5.10: Ateniese–Steiner–Tsudik key agreement

The shared secret is $\mathbf{Z} = g^{r_A r_B K}$, where $K = H(S_{AB})$. In the initial paper [41], the function H is specified as either a hash function with range in $\mathbb{Z}_q^*$ or simply reduction modulo q. However, the later version of the paper [42] specifies that H should be a bijection from $\mathbb{G}$ to $\mathbb{Z}_q$ in the case that $p = 2q + 1$. This latter choice ensures that all possible chosen secrets will be equally likely.

There are many similarities with the Unified Model protocol (Protocol 5.12) and the security properties are similar. Specifically, forward secrecy is provided and unknown key-share attacks are avoided if the principals are guaranteed to know the private keys corresponding to their public keys. Key compromise impersonation attacks are possible, since knowledge of either of the long-term private keys is sufficient to complete the protocol as either initiator or responder.

5.4.3 Just–Vaudenay–Song–Kim Protocol

Recognising that the MTI A(k) protocols do not provide forward secrecy, Just and Vaudenay [407] proposed a variant of MTI A(0) with this property. Their protocol also includes a key confirmation handshake. Later, Song and Kim [685] proposed a very similar protocol designed for use on elliptic curves and with optimised computation, but the shared key is the same in both protocols. The protocol has some enhanced properties over MTI A(0) but unfortunately a security weakness is also introduced, as explained below.

Protocol 5.11 shows a combined version of the two protocols that we call the Just–Vaudenay–Song–Kim (JVSK) protocol. In keeping with our custom in this section we omit the handshake for key confirmation included in the Just–Vaudenay protocol (Song and Kim also provide variants with key confirmation). Just and Vaudenay also proposed a variant of the MTI C(0) protocol.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = y_B^{r_A} \cdot t_B^{x_A + r_A}$	$\xleftarrow{t_B}$	$\mathbf{Z} = y_A^{r_B} \cdot t_A^{x_B + r_B}$

Protocol 5.11: Just–Vaudenay–Song–Kim protocol

The shared secret is $\mathbf{Z} = g^{x_A r_B + x_B r_A + r_A r_B}$. Note that $\mathbf{Z}$ is the key of the MTI A(0) protocol multiplied by the ephemeral Diffie–Hellman key. This change allows the protocol to provide forward secrecy. However, contrary to a claim of Song and Kim [685], the protocol becomes vulnerable to key compromise impersonation. Attack 5.8 shows how this can be implemented with the adversary C masquerading as B, using knowledge of x_A. The attack still applies if key confirmation messages are included, as in the Just–Vaudenay version of the protocol.

Information known to C: Private key of A, x_A.

A		C_B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_C \in_R \mathbb{Z}_q$
		$t_C = g^{r_C} y_B^{-1}$
$\mathbf{Z}_{AB} = y_B^{r_A} \cdot t_C^{x_A + r_A}$	$\xleftarrow{t_C}$	$\mathbf{Z}_{CA} = t_A^{r_C} \cdot t_C^{x_A}$

Attack 5.8: Key compromise impersonation attack on Just–Vaudenay–Song–Kim protocol

There is another flaw in Protocol 5.11, in which an active adversary can replace t_A with y_A^{-1} and t_B with y_B^{-1}. In this case B will calculate $\mathbf{Z} = y_A^{r_B} \cdot y_A^{-x_B - r_B} = y_A^{-x_B} = S_{AB}^{-1}$. Similarly, A calculates $\mathbf{Z} = y_B^{r_A} \cdot t_B^{x_A + r_A} = S_{AB}^{-1}$ and so both principals have agreed the same key, which is the inverse of their static Diffie–Hellman key. If, as we usually assume, the agreed key becomes known to the adversary then the attack can be run again to compromise new sessions. The attack can be avoided if B checks that $t_A \neq y_A^{-1}$ and A does similarly, which adds to the required computation. However, there is no guarantee that other attacks are not possible even if these checks are included. This attack will not work in the version of the protocol that includes key confirmation,

as long at the values of t_A and t_B are included in the confirmation messages. Such messages were included in the Just–Vaudenay version of the protocol.

5.4.4 Unified Model Protocol

The *Unified Model* is a protocol in the NIST SP-800 56A standard [576] which apparently originates in a standards committee document due to Ankney, Johnson and Matyas in 1995 (cited by Blake-Wilson and Menezes [109]). It has a very simple design and attractive security properties. As shown in Protocol 5.12, the shared secret is the concatenation of the static and ephemeral Diffie–Hellman keys: $\mathbf{Z} = g^{r_A r_B}, g^{x_A x_B}$.

Shared information: Static Diffie–Hellman key S_{AB}.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{r_A}, S_{AB}$	$\xleftarrow{t_B}$	$\mathbf{Z} = t_A^{r_B}, S_{AB}$

Protocol 5.12: Unified Model key agreement protocol

Before accepting the shared key, A must make the following checks. B makes the analogous checks.

1. $1 < t_B < p$. In particular, degenerate values such as 0 and p should not be allowed.
2. $t_B^q \bmod p = 1$. This ensures that both components of $\mathbf{Z}$ are in $\mathbb{G}$ as long as A has chosen t_A correctly.

The Unified Model protocol provides forward secrecy, since it is necessary to know one of the ephemeral private keys to find $\mathbf{Z}$. The direct inclusion of the static Diffie–Hellman key prevents unknown key-share attacks if the principals have shown knowledge of the private keys corresponding to their public keys. This is because derivation of the shared secret requires knowledge of one of the long-term private keys. However, the protocol does not prevent key compromise impersonation, since knowledge of either of the long-term keys is sufficient to calculate $\mathbf{Z}$.

Security proofs for the Unified Model protocol were first provided by Blake-Wilson *et al.* [107] in the Bellare–Rogaway model. The basic version shown in Protocol 5.12 was proven secure as long as the Diffie–Hellman assumption holds, but only with a weakened adversary unable to reveal keys from other sessions. Indeed, Blake-Wilson *et al.* [107] pointed out an explicit attack in which the adversary starts

two sessions with the same party and then reflects the first message from each session back to the other session. The two sessions will then accept the same session key, and knowledge of one session key trivially reveals the other.

Jeong *et al.* [397] provided a security proof of Protocol 5.12 in the Bellare–Rogaway model for the case that the session key derivation includes the exchanged messages, so that $\mathbf{K} = H(t_A, t_B, \mathbf{Z})$ for a key derivation function H modelled as a random oracle. Note that this variation requires that the initiator and responder are differentiated, which breaks the symmetry and prevents the attack mentioned above.

The three-round version with key confirmation added (see Sect. 5.4.13) was proven secure by Blake-Wilson *et al.* [107] against an adversary able to obtain session keys from other sessions. Certainly, in this model key confirmation provides a useful security function, rather than simply being a convenient hint to the partner that the key is ready to use. Menezes and Ustaoglu [547] also provided a security proof in a stronger (eCK-style) model (but still not as strong as eCK) for the three-round version of the protocol including key confirmation but now relying on the gap Diffie–Hellman assumption.

5.4.5 MQV Protocol

The MQV protocol was originally due to Menezes *et al.* [551]. It was later improved by these authors plus Law and Solinas [478] and standardised in the IEEE P1363-2000 standard [372].

A special operation is defined on any element t of $\mathbb{Z}_p$, which results in the output $\bar{t} = t \bmod 2^w + 2^w$. The outputs of the operation are of fixed size w, which must be large enough to prevent exhaustive search of 2^w elements. Typically, w would be 80. Protocol 5.13 shows the message exchange in our usual discrete log setting; the protocol is often described in an elliptic curve setting, where by convention the group is written additively.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$S_A = r_A + \bar{t}_A x_A \bmod q$	$\xleftarrow{t_B}$	$S_B = r_B + \bar{t}_B x_B \bmod q$
$\mathbf{Z} = (t_B y_B^{\bar{t}_B})^{S_A}$		$\mathbf{Z} = (t_A y_A^{\bar{t}_A})^{S_B}$

Protocol 5.13: MQV protocol

The shared key is $\mathbf{Z} = g^{(r_A + \bar{t}_A x_A)(r_B + \bar{t}_B x_B)}$. Before accepting the shared key, A must make the following checks. B makes the analogous checks.

1. $1 < t_B < p$. In particular, degenerate values such as 0 and p should not be allowed.
2. $t_B^q \bmod p = 1$. This ensures that $\mathbf{Z} \in \mathbb{G}$ as long as A has chosen t_A correctly.
3. $\mathbf{Z} \neq 1$. Together with the previous check, this ensures that $\mathbf{Z}$ has order q, preventing any small subgroup attacks.

MQV is designed to provide forward secrecy and also protects against key compromise impersonation. The special operation on the exponents destroys the algebraic structure; this may have benefits for practical security but at the same time it obstructs a security proof in terms of any established hard problems. It was shown by Kaliski [409] that MQV is vulnerable to unknown key-share attacks even in the case that users have shown possession of their private keys. Attack 5.9 shows Kaliski's attack, in which the adversary C intercepts and modifies the message sent by A to B.

A	C	B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$\xrightarrow{t_A}$	$u \in_R \mathbb{Z}_q$	
	$t_C = t_A y_A^{\bar{t}_A} g^{-u}$	
	$x_C = (\bar{t}_C)^{-1} u \bmod q$	
	$y_C = g^{x_C}$	$\xrightarrow{t_C}$ $r_B \in_R \mathbb{Z}_q$
$\xleftarrow{t_B}$		$\xleftarrow{t_B}$ $t_B = g^{r_B}$
$S_A = r_A + \bar{t}_A x_A \bmod q$		$S_B = r_B + \bar{t}_B x_B \bmod q$
$\mathbf{Z}_{AB} = (t_B y_B^{\bar{t}_B})^{S_A}$		$\mathbf{Z}_{CB} = (t_C y_C^{\bar{t}_C})^{S_B}$

Attack 5.9: Kaliski's unknown key-share attack on MQV protocol

Although C is able to find the private key x_C corresponding to y_C, it can only be calculated once the first message t_A from A has been seen. An implementation of the attack would therefore require C to get y_C certified before sending on the message t_C to B, a scenario that sounds slightly far-fetched but should not be ruled out without justification. Kaliski suggested that the attack provides a lesson that an active certification authority should be considered in any protocol description. Indeed, modern certificate authorities (CAs) such as Let's Encrypt using the ACME protocol are definitely active online CAs. As usual, the unknown key-share attack may be prevented by including the identities in the key derivation function. It may also be prevented in MQV by addition of key confirmation.

The point of using the reduced-size exponents $\bar{t}_A$ and $\bar{t}_B$ is that the total calculations required by each principal are reduced, by half of one exponentiation, when compared with most other protocols of this type, such as the Unified Model. On the

other hand, it may be possible to perform some of the computation *offline* by choosing random values in advance and assuming the partner's public key is available. In such a case two of the required exponentiations may be performed offline for the Unified Model but only one for MQV, so that MQV requires half of one exponentiation more online computation. A detailed comparison of the Unified Model and MQV was presented by Blake-Wilson and Menezes [109].

Although there has not been any generic security proof for MQV, Kunz-Jacques and Pointcheval [462] did provide a security proof for a version of MQV with key confirmation and in a specific group. Their proof is in a Bellare–Rogaway-style model and relies on a 'custom' variant of the Diffie–Hellman problem tied to the special definition of $\bar{t}$, known as the f-randomized computational Diffie–Hellman problem. The same authors [463] also formally analysed an MQV variant protocol designed to be secure in a model differentiating secure storage from less secure computation.

5.4.6 HMQV Protocol

The general MQV protocol has never benefited from a security proof. However, in 2005 Krawczyk [453] proposed a variant protocol, named HMQV, and provided an extensive security analysis with proofs of various properties.

The essential difference between HMQV and the original MQV (Protocol 5.13) lies in the way that the values S_A and S_B are calculated. Specifically, the special operation $\bar{t} = t \bmod 2^w + 2^w$ used in MQV is replaced by application of a hash function H modelled as a random oracle, and with output size $|q|/2$. The shared secret thus becomes $\mathbf{Z} = g^{(r_A + dx_A)(r_B + ex_B)}$ where $d = H(t_A, ID_B)$ and $e = H(t_B, ID_A)$. In the description in Protocol 5.14, notice that the session key $\mathbf{K}$ is directly specified as the output of the hash function H' with any chosen output length.

Information computed during protocol: $d = H(t_A, ID_B)$, $e = H(t_B, ID_A)$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$S_A = r_A + dx_A \bmod q$		$S_B = r_B + ex_B \bmod q$
$\mathbf{Z} = (t_B y_B^e)^{S_A}$		$\mathbf{Z} = (t_A y_A^d)^{S_B}$
	$\mathbf{K} = H'(\mathbf{Z})$	

Protocol 5.14: HMQV protocol

Publication of the HMQV protocol was followed by significant controversy concerning the security properties of the protocol and their proofs, particularly in comparison with the original MQV protocol [543, 545]. One of the main issues relates to the circumstances in which the public keys, both static keys and received ephemeral keys, need to be checked to lie within the group $\mathbb{G}$. Sometimes these checks are needed in order to avoid small subgroup attacks. We refer to such tests as *group membership tests*.

Group membership for long-term keys (y_A and y_B) could be checked by a certification authority and therefore implicitly provided in the key's certificate. This is more efficient than asking the protocol principals to check, since certificates remain valid for a long period. However, practice shows that certification authorities may not be diligent in carrying out such checkss so it is not universally accepted that this is a good solution.

Group membership tests can be costly, but the cost depends to a large extent on the structure of the group $\mathbb{G}$. HMQV is defined to run in any $\mathbb{G}$ with prime order q. Typical choices for $\mathbb{G}$ would be subgroups of $\mathbb{Z}_p^*$ and elliptic curve groups. In the former case the group membership check is relatively expensive, close to the cost of an additional exponentiation, while in the latter case the check is usually cheap or even free.

HMQV Security

Krawczyk [453] provided several different security proofs for HMQV, depending on what security properties were considered and which version of the protocol was analysed. Table 5.4 summarises the different properties for four main situations. For the two-pass version, as shown in Protocol 5.14, security in the CK model is achieved under the computational Diffie–Hellman assumption. In order to achieve security against ephemeral-key disclosure, the assumptions are strengthened to the gap Diffie-Hellman (GDH) assumption and the knowledge-of-exponents (KEA) assumption. In addition, as observed by Menezes [543], the protocol must include checks that the received ephemeral keys lie in $\mathbb{G}$. (The requirements for group membership are mentioned in the revised HMQV paper, but only in the preface.) All proofs require that the hash function H has the properties of a random oracle.

Sarr and Elbaz-Vincent [651] found a KCI attack on HMQV in the case that group membership tests are not applied. The attack is similar to the Lim–Lee attacks in Sect. 5.3.3, in which the attacker chooses an ephemeral key outside the subgroup in which the protocol operates. Some special properties of the chosen parameters need to be satisfied which do not hold in general, but that attack shows at the least that the security results claimed cannot hold generically. The attack will be detected and prevented if group membership tests are used. Including both t_A and t_B in the computation of d and e also prevents the attack.

Menezes and Ustaoglu [546, 548] observed that HMQV may not be secure in the post-specified peer model. They described an attack in which A starts the protocol, sending t_A without selecting its peer entity. An adversary finds an identity M so that $d = H(t_A, ID_B) = H(t_A, ID_M)$ for some honest party B. The adversary then registers

Table 5.4: Security of HMQV protocols

Protocol version	Security property	Computational assumption	Group membership tests needed
two-pass HMQV	CK-secure with weak forward secrecy	CDH	No
two-pass HMQV	Leakage of ephemeral secrets	GDH and KEA	Yes
three-pass HMQV-C	CK-secure plus full forward secrecy and key confirmation	CDH	No
one-pass HMQV	CK-secure without replay protection	CDH	Yes

M as a legitimate party. This allows the adversary to complete an unknown key-share attack in which A and B compute the same session key, but A believes that the key is shared with M while B believes it is shared with A. Since this requires finding a collision in H it is not necessarily practical, but it exploits the birthday paradox to achieve the attack in less than the expected time. Krawczyk [453, Remark 7.2] suggested that in cases where the output size of the hash H was small a random nonce could be chosen and included in the e and d hashes at the time they were first computed, which would prevent the Menezes and Ustaoglu attack. Another defence is to include the party identities in the key derivation function used to compute $\mathbf{K}$.

Hao [344] proposed an attack in which the adversary chooses an invalid public key within a small subgroup and can complete the protocol without possessing any private key. While this is a surprising property, it does not violate any claimed security property so it is debatable whether it constitutes a valid attack.

One-Pass and Three-Pass HMQV

Table 5.4 mentions the three-pass and one-pass variants of HMQV. The three-pass variant, called HMQV-C, where the C denotes confirmation, adds MACs in the manner of Sect. 5.4.13 and provides full forward secrecy as well as key confirmation. Note that group membership may be required to avoid the Sarr and Elbaz-Vincent attack [651], unless other measures are taken as in the FHMQV variant mentioned below.

The one-pass variant of HMQV consists of a single message t_A sent from A to B. The session key is computed as in Protocol 5.14 with the adjustments $t_B = e = 1$, $S_B = x_B$ and $d = H(t_A, ID_A, ID_B)$. Thus A computes $\mathbf{Z} = y_B{}^{S_A}$ and B computes $\mathbf{Z} = (t_A y_A^d)^{x_B}$. (A later version of the one-pass protocol [343] also adds the identities of the parties and the protocol message to the key derivation function as recommended by Menezes [543].) The protocol provides the same security as the two-pass protocol except that B has no way of checking whether or not the received message is replayed. Such protection can never be provided in a one-pass protocol without extra mechanisms such as timestamps or counters.

HMQV variants

Ustaoglu [719] described a protocol called the Unified Protocol, or UP. The shared secret has two parts, $\mathbf{Z}_1$ and $\mathbf{Z}_2$, where $\mathbf{Z}_1 = g^{(r_A+x_A)(r_B+ex_B)}$ and $\mathbf{Z}_2 = g^{(r_A+dx_A)(r_B+x_B)}$ with $d = H(t_A)$ and $e = H(t_B)$. At the cost of one additional exponentiation, UP allows a security proof in the eCK model in a relatively straightforward manner and with a tighter reduction than in the HMQV proof. Indeed, the security proof of Ustaoglu [719] is in the model of Menezes and Ustaoglu [548], called eCK+ by Ustaoglu [719], which is a stronger model than eCK. The eCK+ model incorporates security in both pre- and post-specified models.

Sarr *et al.* [652] explored attacks on HMQV in which the adversary is allowed to obtain (perhaps partial) information about the internal computed values of the targeted principal, specifically the secret exponents S_A and S_B and the shared secret input to the key derivation function H'. They showed that if such information is available then there are attacks on HMQV and they therefore proposed a variant protocol, which they called FHMQV. The changes in FHMQV compared with Protocol 5.14 consist of increasing the set of inputs to the two hash computations. Specifically the new values of d and e become $d = H(t_A, t_B, ID_A, ID_B)$ and $e = H(t_B, t_A, ID_A, ID_B)$, while the computation of the session key becomes $\mathbf{K} = H'(\mathbf{Z}, t_A, t_B, IB_A, ID_B)$. In addition, group membership tests are required for the received ephemeral keys. Later, Liu *et al.* [498] criticised the analysis of FHMQV, observing that the model used was incomparable with that used by Krawczyk, and also claiming gaps in the proof for FHMQV. This latter claim was later addressed by Sarr and Elbaz-Vincent [651], who provided new proofs.

Zhao and Zhang [775] proposed a protocol which they called sHMQV, with strong similarities to FHMQV. Again the differences from HMQV are the inputs to the hash functions, which are identical to those used in FHMQV except that ID_A is dropped from d and ID_B is dropped from e. The main difference comes in how sHMQV is analysed. Zhao and Zhang [775] used a model involving a trusted hardware module, which was used to hide the ephemeral secrets r_A and r_B from the adversary while allowing access to the exponents S_A and S_B.

Pan and Wang [598] described a variant of HMQV using the twin Diffie–Hellman technique of Cash *et al.* [184]. They called the resulting protocol TMQV. The advantage of this idea is that it removes the GDH assumption in favour of the CDH assumption. However, it does double the size of the public key and, as pointed out by Pan and Wang, gives a non-tight security reduction.

5.4.7 NAXOS Protocol

As discussed in Chap. 2, LaMacchia *et al.* [470] introduced the eCK security model in 2007 and accompanied it with a protocol design known as NAXOS.[3] Protocol 5.15 shows the protocol messages and the computation of the shared secret.

[3] The protocol name is not an acronym, but a Greek island different from Kea.

A		B
$r'_A \in_R \mathbb{Z}_q$ $r_A = H(r'_A, x_A)$ $t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r'_B \in_R \mathbb{Z}_q$ $r_B = H(r'_B, x_B)$ $t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$\mathbf{Z} = t_B^{x_A}, y_B^{r_A}, t_B^{r_A}$		$\mathbf{Z} = y_A^{r_B}, t_A^{x_B}, t_A^{r_B}$
	$\mathbf{K} = H'(\mathbf{Z}, ID_A, ID_B)$	

Protocol 5.15: NAXOS protocol

The shared secret consists of three components: $g^{x_A r_B}, g^{r_A x_B}, g^{r_A r_B}$. The specification requires that these are combined with the principal identities to form the session key using a key derivation function. There is some similarity between NAXOS and earlier protocols such as those of Just and Vaudenay and of Song and Kim (see Sect. 5.4.3). In particular, the components are almost the same as those used in Protocol 5.11, but there are two significant differences.

- The exponents are not added together, but rather simply concatenated in a specific order. This means that the protocol principals must be aware of some ordering of who should take the role of A and who should take the role of B.
- Instead of using the random values r'_A and r'_B directly as ephemeral Diffie–Hellman values, they are first hashed together with the long-term keys x_A and x_B respectively. The reason for this is that an adversary who obtains the random values r'_A and r'_B does not obtain the shared secret, and this allows the NAXOS protocol to be secure in the eCK model.

Intuitively, it can be seen that any adversary who lacks one of the pairs (x_A, r'_A) or (x_B, r'_B) out of the set $\{x_A, r'_A, x_B, r'_B\}$ is unable to compute the shared secret. LaMacchia *et al.* [470] provided a security proof of Protocol 5.15 in the eCK model assuming that the gap Diffie–Hellman problem is hard.

A variant protocol known as NAXOS+ was proposed by Lee and Park [480], which adds the static Diffie–Hellman value into the computation of the shared secret. Lee and Park showed that this change, which adds one exponentiation per principal, allows the protocol to be proven secure in the eCK model under the computational Diffie–Hellman assumption. A different protocol due to Huang and Cao [367] has similar properties to NAXOS+ but requires a pair of long-term keys for each party. Barthe *et al.* [57] showed, with the help of machine support, that security of the original NAXOS can be proven assuming only the computational Diffie–Hellman assumption if long-term keys are always generated honestly (i.e. not by the adversary).

The NAXOS Trick

The technique of hashing together the random value and the long-term key has become known as the *NAXOS trick* and has been repeated in several later protocols. As mentioned above, the technique allows protocols such as Protocol 5.15 to be proven secure in the eCK model, where the adversary can obtain r'_A and r'_B but is still unable to find useful information about the shared secret.

In some ways, the NAXOS trick seems artificial. It is justified by an assumption that an attacker may have access to the random values generated by protocol principals but not to other values computed using those random values. Note that it is possible, and often specified for particular protocols, that the values r_A and r_B are recomputed just before they are needed in the computation of t_A and t_B and, later, in the computation of $\mathbf{Z}$. This means that the r_A and r_B values are never *stored*. But does this necessarily mean that they are more secure? That must depend on the physical implementation and what parts of the system may be controlled or accessed by the adversary. It has been pointed out by Ustaoglu [719] that side-channel attacks may be an effective way to obtain r_A and r_B values, especially when pre-computation is used. This has been used as an argument to avoid the NAXOS trick, but eCK-secure protocols avoiding such tricks seem to be less efficient. Examples are the HMQV variant Unified Protocol [719] (see Sect. 5.4.6) and a variant of Protocol 5.21 [566].

The NAXOS trick uses a hash function modelled as a random oracle for the security proof. A related idea, sometimes called the *twisted PRF* trick, avoids using a hash function to allow the trick to work within a standard-model proof. In the twisted PRF trick, the long-term and ephemeral keys are used alternately as the key and the input to independent pseudo-random functions (PRFs). Later we will see the twisted PRF trick applied in Protocols 5.21 and 5.44.

We note that, strictly speaking, protocols using the NAXOS trick do not fit into this section, since they do not use the basic Diffie–Hellman messages. However, since they use no explicit authenticating information, we feel that it makes sense to set them beside other protocols whose messages do have the basic Diffie–Hellman format.

5.4.8 CMQV Protocol

In comparison with the HMQV protocol, Ustaoglu [718] identified two disadvantages of NAXOS. First, the protocol is less efficient, requiring in total four exponentiations per party as compared with the 2.5 needed in HMQV. Second, there is no natural way to derive a one-pass version of NAXOS; for example, if $t_B = 1$ is chosen, then the shared secret essentially reduces to $g^{r'_A x_B}$, allowing an attacker to freely choose r'_A and to masquerade as A. At the same time, Ustaoglu recognised the additional security properties of NAXOS as well as the simplicity of the proof in comparison with HMQV. This motivated the protocol CMQV [718] (*combined* MQV), which aims to achieve the efficiency and flexibility of HMQV with the security and ease of proof of NAXOS. Protocol 5.16 show the message flows and secret key computation for CMQV.

Information computed during protocol: $d = H(t_A, ID_A, ID_B)$, $e = H(t_B, ID_A, ID_B)$.

A		B
$r'_A \in_R \mathbb{Z}_q$ $r_A = H(r'_A, x_A)$ $t_A = g^{r'_A}$	$\xrightarrow{t_A}$	$r'_B \in_R \mathbb{Z}_q$ $r_B = H(r'_B, x_B)$ $t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$S_A = r_A + dx_A \bmod q$ $\mathbf{Z} = (t_B y_B^e)^{S_A}$		$S_B = r_B + ex_B \bmod q$ $\mathbf{Z} = (t_A y_A^d)^{S_B}$
$\mathbf{K} = H'(\mathbf{Z}, t_A, t_B, ID_A, ID_B)$		

Protocol 5.16: CMQV protocol

We can say that Protocol 5.16 essentially uses the shared-secret derivation from HMQV together with the NAXOS trick. Thus the shared secret for CMQV is $\mathbf{Z} = g^{(r'_A + dx_A)(r'_B + ex_B)}$, but note that now d and e depend on both identities, a difference from HMQV. Ustaoglu presented a proof of security for CMQV in the eCK model under the gap Diffie–Hellman assumption. In comparison with that for HMQV, the proof is quite short. However, like that for HMQV, the proof requires application of the *forking lemma*, which impacts the tightness of the reduction. It is required that the message recipients check membership of the group $\mathbb{G}$. Despite the similarity with HMQV, the hash function H used in CMQV is assumed to map to the whole of $\mathbb{G}$, rather than mapping to bit strings of half the length of the size of $\mathbb{G}$ as in HMQV. This means that CMQV does not always achieve the same efficiency as HMQV, although it is always more efficient than NAXOS.

5.4.9 NETS and SMEN

Lee and Park [479] continued on from the CMQV design, looking for efficient protocols which can satisfy eCK security with a simple security proof. In particular, they addressed the undesirable use of the forking lemma in the security proof of CMQV, which results in a less tight security reduction. They defined a new protocol, NETS, shown as Protocol 5.17.

Protocol 5.17 makes use of the NAXOS trick to compute the ephemeral exponents, and the shared secret consists of two components: $\mathbf{Z} = g^{(x_A + r_A)(x_B + r_B)}, g^{r_A r_B}$. Lee and Park proved security of NETS in the eCK model assuming the difficulty of gap Diffie–Hellman but without relying on the forking lemma. Subsequently, Barthe *et al.* [57] showed, with the help of machine support, that security can also be proven

A		B
$r'_A \in_R \mathbb{Z}_q$		
$r_A = H(r'_A, x_A)$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r'_B \in_R \mathbb{Z}_q$
		$r_B = H(r'_B, x_B)$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$\mathbf{Z} = (t_B y_B)^{x_A + r_A}, {t_B}^{r_A}$		$\mathbf{Z} = (t_A y_A)^{x_B + r_B}, {t_A}^{r_B}$
$\mathbf{K} = H'(\mathbf{Z}, t_A, t_B, ID_A, ID_B)$		

Protocol 5.17: NETS protocol

only under the computational Diffie–Hellman assumption if long-term keys are generated honestly.

The efficiency of Protocol 5.17 is an improvement upon NAXOS and matches CMQV when one simply counts the number of exponentiations. However, as acknowledged by Lee and Park [479], CMQV (and also (H)MQV) can use techniques for multi-exponentiation which are not available for NETS. At the same time, the tighter security reduction should mean that smaller keys are possible for NETS than for CMQV, so it is not easy to compare precisely the efficiency for the same security level.

A later protocol with, at least superficially, similarities with Protocol 5.11 is due to Wu and Ustaoglu [738]. Known as SMEN (Secure MQV or Efficient NAXOS), Protocol 5.18 uses the NAXOS trick and incorporates two independent Diffie–Hellman ephemeral values. The main theoretical advance of SMEN compared with NETS is that SMEN can exploit multi-exponentation to obtain efficiency improvements. Apart from that, it has similar properties to NETS, particularly those of having a compact proof without the forking lemma and using the NAXOS trick.

The shared secret is $\mathbf{Z} = g^{x_A r_B + x_B r_A + \tilde{r}_A \tilde{r}_B}$, which looks similar to that of Protocol 5.11, but note the differences due to both the NAXOS trick and the use of two different ephemeral secrets. Wu and Ustaoglu proved that SMEN is secure in the eCK model based on the gap Diffie–Hellman assumption. Later, Lu *et al.* [505] demonstrated a KCI attack on SMEN, but the attack applies only in a different security model in which the session state from non-target sessions is allowed to be revealed to the adversary. Wu and Ustaoglu [738] also define a protocol called SMEN− which avoids using the NAXOS trick at the cost of a small reduction in efficiency.

A		B
$r'_A, \tilde{r}'_A \in_R \mathbb{Z}_q$ $r_A = H(r'_A, x_A)$ $\tilde{r}_A = H(\tilde{r}'_A, x_A)$ $t_A = g^{r_A}, \tilde{t}_A = g^{\tilde{r}_A}$		
	$\xrightarrow{t_A, \tilde{t}_A}$	$r'_B, \tilde{r}'_B \in_R \mathbb{Z}_q$ $r_B = H(r'_B, x_B)$ $\tilde{r}_B = H(\tilde{r}'_B, x_B)$ $t_B = g^{r_B}, \tilde{t}_B = g^{\tilde{r}_B}$
$\mathbf{Z} = y_B^{r_A} \cdot t_B^{x_A} \cdot \tilde{t}_B^{\tilde{r}_A}$	$\xleftarrow{t_B, \tilde{t}_B}$	$\mathbf{Z} = y_A^{r_B} \cdot t_A^{x_B} \cdot \tilde{t}_A^{\tilde{r}_B}$
$\mathbf{K} = H'(\mathbf{Z}, ID_A, ID_B, t_A, \tilde{t}_A, t_B, \tilde{t}_B)$		

Protocol 5.18: SMEN protocol

5.4.10 Protocol of Kim, Fujioka, and Ustaoglu

Kim *et al.* [429] proposed Protocol 5.19 in order to achieve a protocol which avoids using the NAXOS trick while at the same time maintaining a compact proof in the eCK model.

Shared information: Public keys: $y_A, \tilde{y}_A$ of A and $y_B, \tilde{y}_B$ of B
Information of A: Private keys: $x_A, \tilde{x}_A$ with $y_A = g^{x_A}, \tilde{y}_A = g^{\tilde{x}_A}$
Information of B: Private keys: $x_B, \tilde{x}_B$ with $y_B = g^{x_B}, \tilde{y}_B = g^{\tilde{x}_B}$

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$\mathbf{Z} = (t_B y_B)^{x_A + r_A}, (t_B \tilde{y}_B)^{\tilde{x}_A + r_A}$		$\mathbf{Z} = (t_A y_A)^{x_B + r_B}, (t_A \tilde{y}_A)^{\tilde{x}_B + r_B}$
$\mathbf{K} = H'(\mathbf{Z}, t_A, t_B, ID_A, ID_B)$		

Protocol 5.19: Protocol of Kim, Fujioka, and Ustaoglu (KFU)

The shared secret consists of two components:

$$\mathbf{Z} = g^{(x_A + r_A)(x_B + r_B)}, g^{(\tilde{x}_A + r_A)(\tilde{x}_B + r_B)}.$$

The format of the shared secret has a similarity with NETS (Protocol 5.17) but the protocol avoids using the NAXOS trick to derive the exponents for t_A and t_B. Protocol 5.19 requires each principal to have two long-term public/private key pairs. The computation of $\mathbf{Z}$ consists of the same basic operation run twice, once with the first long-term key pair of both parties, and once with the second.

Kim *et al.* [429] proved that Protocol 5.19 is secure in the eCK model based on the gap Diffie–Hellman assumption. They also defined a second protocol which uses the same pair of long-term keys for each principal but computes $\mathbf{Z}$ with four components:

$$\mathbf{Z} = g^{(x_A+r_A)(x_B+r_B)}, g^{(x_A+r_A)(\bar{x}_B+r_B)}, g^{(\bar{x}_A+r_A)(x_B+r_B)}, g^{(\bar{x}_A+r_A)(\bar{x}_B+r_B)}.$$

They proved security of this variant assuming the difficulty of the computational Diffie–Hellman problem. Although it adds to the computational cost, this variant still uses only a single basic Diffie–Hellman message exchange. Pan and Wang [598] pointed out attacks on the KFU protocols in other security models, particularly the seCK model (see page 79).

5.4.11 OAKE Protocol

The OAKE (Optimally-balanced Authenticated Key Exchange[4]) protocol was designed by Yao and Zhao [754] as a compromise between the HMQV and KEA protocols, aiming to benefit from the best aspects of each. Based on the observation that HMQV has the best known performance in total, while KEA has the best known performance *online* (see Sect. 5.4.14), OAKE combines the methods of forming the shared secret $\mathbf{Z}$ in HMQV and in KEA as shown in Protocol 5.20 (compare Protocols 5.14 and 5.9). OAKE even marginally improves on the overall performance of HMQV by removing one of the hash computations.

The shared secret in Protocol 5.20 is $\mathbf{Z} = g^{r_A x_B + r_B x_A + e r_A r_B}$. Yao and Zhao [754] proved the security of OAKE in the Canetti–Krawczyk computational model based on either the gap Diffie–Hellman problem or a combination of the gap discrete logarithm problem and the knowledge-of-exponent assumption.

In addition to computational concerns, Yao and Zhao [754] also considered privacy issues. They pointed out that KEA, and related protocols whose secret inputs are of the form $y_B^{r_A}$, can easily be simulated by either party. This means that they provide a form of deniability. In contrast, (H)MQV, like many other protocols, is much harder to simulate because the static Diffie–Hellman value $g^{x_A x_B}$ seems to be required in order to compute a valid transcript and key. OAKE is easy to simulate, like KEA. In case deniability was seen as undesirable, Yao and Zhao also defined a variant protocol called T-OAKE (traceable OAKE) with similar computational properties to OAKE but which now includes $g^{x_A x_B}$ in the shared secret component.

[4] The authors of the OAKE protocol actually specified a much longer version of the protocol name: '(toward) Optimally-balanced (implicitly) Authenticated (Diffie–Hellman) Key-Exchange (in the integrity of protocol efficiency, security, privacy, and easy deployment)'.

Information computed during protocol: $e = H(ID_A, y_A, ID_B, y_B, t_A, t_B)$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$S_A = x_A + er_A \bmod q$ $\mathbf{Z} = y_B^{r_A} \cdot t_B^{S_A}$		$S_B = x_B + er_B \bmod q$ $\mathbf{Z} = y_A^{r_B} \cdot t_A^{S_B}$
	$\mathbf{K} = H'(\mathbf{Z})$	

Protocol 5.20: OAKE protocol

5.4.12 Moriyama–Okamoto Protocols

Okamoto [591] designed Protocol 5.21 with the aim of achieving security in the eCK model but without using random oracles, so that it can be proven secure in the *standard model*. This was the first protocol to use the twisted PRF trick in place of the NAXOS trick in order to avoid the random-oracle assumption. The long-term and ephemeral keys are combined using pseudo-random function families $\hat{F}$ and $\tilde{F}$. The hash functions H_A and H_B, used to compute c and d, are chosen from a family of target collision-resistant functions.

The shared secret can be shown to be computed equally by A and B as follows:

$$\begin{aligned}
\mathbf{Z} &= t_{B,1}^{x_{A,1}+cx_{A,3}} \cdot t_{B,2}^{x_{A,2}+cx_{A,4}} \cdot t_{B,3}^{\tilde{r}_A} \cdot y_B^{r_A} \cdot \tilde{y}_B^{dr_A} \\
&= g_1^{r_B(x_{A,1}+cx_{A,3})} \cdot g_2^{r_B(x_{A,2}+cx_{A,4})} \cdot g_1^{\tilde{r}_B\tilde{r}_A} \cdot g_1^{x_{B,1}r_A} \cdot g_2^{x_{B,2}r_A} \cdot g_1^{x_{B,3}dr_A} \cdot g_2^{x_{B,4}dr_A} \\
&= (g_1^{x_{A,1}} g_2^{x_{A.2}})^{r_B} \cdot (g_1^{x_{A,3}} g_2^{x_{A,4}})^{cr_B} \cdot g_1^{\tilde{r}_B\tilde{r}_A} \cdot g_1^{r_A(x_{B,1}+dx_{B,3})} \cdot g_2^{r_A(x_{B,2}+dx_{B,4})} \\
&= y_A^{r_B} \cdot \tilde{y}_A^{cr_B} \cdot t_{A,3}^{\tilde{r}_B} \cdot t_{A,1}^{(x_{B,1}+dx_{B,3})} \cdot t_{A,2}^{(x_{B,2}+dx_{B,4})} \quad .
\end{aligned}$$

From $\mathbf{Z}$, the session key is computed using a special type of function called a πPRF, or *pseudo-random function with pairwise-independent random sources*. The existence of such a function is an assumption of the security proof. In comparison with other protocols in this section, Protocol 5.21 is quite complex, using five private key elements and two public keys elements, exchanging three elements, and requiring eight basic exponentiations per principal.

Later, Moriyama and Okamoto [566] designed a new version of Protocol 5.21 which avoids the twisted PRF (and NAXOS) trick but requires an even longer public and private key pair. They also developed a related protocol secure against side-channel attacks in the so-called leakage resilience model [567].

Pre-shared information: Two generators g_1 and g_2 for $\mathbb{G}$. Public keys: $y_A, \tilde{y}_A$ of A and $y_B, \tilde{y}_B$ of B

Information of A: Private keys: $x_{A,0}, x_{A,1}, x_{A,2}, x_{A,3}, x_{A,4}$ with $y_A = g_1^{x_{A,1}} g_2^{x_{A,2}}, \tilde{y}_A = g_1^{x_{A,3}} g_2^{x_{A,4}}$

Information of B: Private keys: $x_{B,0}, x_{B,1}, x_{B,2}, x_{B,3}, x_{B,4}$ with $y_B = g_1^{x_{B,1}} g_2^{x_{B,2}}, \tilde{y}_B = g_1^{x_{B,3}} g_2^{x_{B,4}}$

Values computed during protocol: $c = H_B(ID_B, t_{B,1}, t_{B,2}, , t_{B,3})$, $d = H_A(ID_A, t_{A,1}, t_{A,2}, t_{A,3})$

A		B
$r'_A, \tilde{r}'_A \in_R \mathbb{Z}_q$		
$x_A = \sum_{i=0}^{4} x_{A,i}$		
$(r_A, \tilde{r}_A) = \hat{F}_{r_{A'}}(1^k) + \tilde{F}_{x_A}(\tilde{r}'_A)$		
$t_{A,1} = g_1^{r_A}, t_{A,2} = g_2^{r_A}, t_{A,3} = g_1^{\tilde{r}_A}$	$\xrightarrow{t_{A,1}, t_{A,2}, t_{A,3}}$	
		$r'_B, \tilde{r}'_B \in_R \mathbb{Z}_q$
		$x_B = \sum_{i=0}^{4} x_{B,i}$
		$(r_B, \tilde{r}_B) = \hat{F}_{r_{B'}}(1^k) + \tilde{F}_{x_B}(\tilde{r}'_B)$
		$t_{B,1} = g_1^{r_B}, t_{B,2} = g_2^{r_B}, t_{B,3} = g_1^{\tilde{r}_B}$
		$\mathbf{Z} = t_{A,1}^{x_{B,1}+dx_{B,3}} \cdot t_{A,2}^{x_{B,2}+dx_{B,4}} \cdot t_{A,3}^{\tilde{r}_B} \cdot y_A^{r_B} \cdot \tilde{y}_A^{cr_B}$
	$\xleftarrow{t_{B,1}, t_{B,2}, t_{B,3}}$	
$\mathbf{Z} = t_{B,1}^{x_{A,1}+cx_{A,3}} \cdot t_{B,2}^{x_{A,2}+cx_{A,4}} \cdot t_{B,3}^{\tilde{r}_A} \cdot y_B^{r_A} \cdot \tilde{y}_B^{dr_A}$		

Protocol 5.21: Okamoto protocol

5.4.13 Adding Key Confirmation

In this section, we have so far considered only key agreement protocols with two message flows without any key confirmation. All the protocols we have described can be extended to provide key confirmation in a generic way, and indeed this technique was explicitly used by many of the protocol authors. Most of the protocols we have described in this section provide weak forward secrecy. This technique not only provides key confirmation but also changes a protocol with weak forward secrecy into one with forward secrecy.

Protocol 5.22 shows the general procedure. In the first two messages A and B exchange their ephemeral Diffie–Hellman keys as usual, while in the second and third they exchange MACs using the derived key K'. To ensure that the protocol does not give away material to help the adversary, two different hash functions are used, H_1 and H_2, which are assumed to be independent. The key K' is calculated using hash function H_1, while H_2 is used to derive the shared session key $\mathbf{K} = H_2(\mathbf{Z})$. This ensures that there is no obstacle to a Bellare–Rogaway-style proof of security as a result of this procedure. Exactly this procedure was used by Blake-Wilson *et al.* [107, 109] to provide key confirmation to the Unified Model and prove its security. Yang [752] provided a formal proof that this process provides explicit entity authentication for any two-message protocol that is secure in the eCK model.

Shared information: Two independent hash functions, H_1 and H_2.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$ Calculate $\mathbf{Z}$
Calculate $\mathbf{Z}$ $K' = H_1(\mathbf{Z})$	$\xleftarrow{t_B, \mathsf{MAC}_{K'}(2, ID_B, ID_A, t_B, t_A)}$	$K' = H_1(\mathbf{Z})$
Verify MAC	$\xrightarrow{\mathsf{MAC}_{K'}(3, ID_A, ID_B, t_A, t_B)}$	Verify MAC

Protocol 5.22: Generic addition of key confirmation to basic Diffie–Hellman protocols

Although the MACs exchanged contain essentially the same fields, the ordering is different. This ensures that no MAC may be replayed to the opposite role, either to the initiator or to the responder of the protocol; in particular, it prevents a simple reflection of the MAC value in the third message. The inclusion of the message flow numbers also ensures this.

5.4.14 Comparison of Basic Diffie–Hellman Protocols

Table 5.5 summarises and compares the main features of some of the protocols we have examined in this section. The table includes only two-pass protocols, although many protocols also have one-pass and three-pass versions. Due to this restriction we expect that in most cases only weak forward secrecy will be achieved. As pointed out in Sect. 5.4.13, such protocols can be converted to ones with strong forward secrecy by adding key confirmation.

In comparing the computed shared secrets in each protocol, note that those protocols which use the NAXOS trick use values of r_A and r_B which have been combined with the respective long-term secrets. SMEN uses two ephemeral keys and KFU uses two long-term keys; the additional keys are denoted by adding a tilde.

Security properties are arguably the most important issues to compare in Table 5.5. Older protocols often lack some of the basic security properties and usually do not have a proof of security. They are still interesting, at the least to understand what can go wrong and why more modern protocols are designed the way they are.

All modern protocols have a proof of security, but the models used are not always the same. The most common is the eCK model, but some analysts use the CK model with enhancements to capture KCI resistance and ephemeral-key leakage, denoted CK^+ in the table. Because the UM protocol does not provide resistance to

Table 5.5: Summary of major properties of key agreement protocols using basic message format

Properties → ↓ *Protocol*	Weak forward secrecy	Resist KCI	Exponentiations offline	Exponentiations online	Formal proof	Shared secret	NAXOS trick
MTI A(0)	No	Yes	2	1	No	$g^{x_A r_B + x_B r_A}$	No
MTI B(0)	No	Yes	2	1	No	$g^{r_A + r_B}$	No
MTI C(0)	Yes	No	1	1	No	$g^{r_A r_B}$	No
KEA (5.9)	No	Yes	2	1	No	$g^{x_A r_B + x_B r_A}$	No
KEA+	Yes	Yes	2	1	CK^+	$g^{x_A r_B}, g^{x_B r_A}$	No
AST (5.10)	Yes	No	2	1	No	$g^{r_A r_B H(S_{AB})}$	No
JVSK (5.11)	Yes	No	2	1	No	$g^{x_A r_B + x_B r_A + r_A r_B}$	No
UM (5.12)	Yes	No	2	1	eCK^-	$g^{x_A x_B}, g^{r_A r_B}$	No
MQV (5.13)	Yes	Yes	1	1.5	BR	$g^{(r_A + \bar{t}_A x_A)(r_B + \bar{t}_B x_B)}$	No
HMQV (5.14)	Yes	Yes	1	1.5	CK^+	$g^{(r_A + d x_A)(r_B + e x_B)}$	No
NAXOS (5.15)	Yes	Yes	2	2	eCK	$g^{x_A r_B}, g^{r_A x_B}, g^{r_A r_B}$	Yes
CMQV (5.16)	Yes	Yes	1	2	eCK	$g^{(r_A + d x_A)(r_B + e x_B)}$	Yes
NETS (5.17)	Yes	Yes	1	2	eCK	$g^{(x_A + r_A)(x_B + r_B)}, g^{r_A r_B}$	Yes
SMEN (5.18)	Yes	Yes	1	2	eCK	$g^{x_A r_B + x_B r_A + \tilde{r}_A \tilde{r}_B}$	Yes
KFU (5.19)	Yes	Yes	1	2	eCK	$g^{(x_A + r_A)(x_B + r_B)}, g^{(\tilde{x}_A + r_A)(\tilde{x}_B + r_B)}$	No
OAKE (5.20)	Yes	Yes	2	1	CK^+	$g^{x_A r_B + x_B r_A + e r_A r_B}$	No
Okamoto (5.21)	Yes	Yes	3	5	eCK	See Protocol 5.21	TPRF

KCI attacks, it cannot be secure in the full eCK model and its proof is in a weakened version, denoted eCK^-. The proof for MQV is in a version of the BR model which is weaker than the eCK or CK^+ models and only applies when key confirmation is added. More details can be found in the descriptions of the individual protocols. Most of the security proofs require a random oracle but there are some exceptions, such as for Protocol 5.21. Many protocols use the NAXOS trick with a random oracle to hash the ephemeral and long-term secrets, while Protocol 5.21 uses the alternative twisted PRF trick instead (denoted TPRF in Table 5.5).

The computational requirements are indicated in Table 5.5 by simply counting the number of exponentiations computed by each principal in a protocol run; this number is divided into those that must be online (after the protocol starts) and those that may be offline, assuming that the partner's public key is available and the ephemeral Diffie–Hellman private key is chosen in advance. It is important to note that we have counted only those computations required to calculate the shared secret. For all protocols, it is typically necessary to perform an extra online exponentiation to prevent small subgroup attacks in the case that $\mathbb{G}$ is a subgroup of much smaller size than $\mathbb{Z}_p^*$ (see Sect. 5.3.1). For MQV and HMQV, the value of one of the exponents used is half the size of the other exponents, which accounts for the half exponentiation shown in the table.

Different implementation optimisations are possible in many of the protocols. Two important ones are multi-exponentiation [550, Algorithm 14.88] and fixed-based exponentiation [550, Sect. 14.6.3], possibly including pre-computation. These optimisations can significantly alter the relative efficiency but may require extra storage. Some researchers [429, 719] have estimated the efficiency for many of the protocols in Table 5.5 taking into account such techniques. However, this is not easy because different levels of tightness in security proofs can influence what security parameters can safely be used [719].

Overall, there seems to be no clear winner amongst this class of protocols. Today, many key exchange protocols are proven to be secure in strong security models. With the emerging threat of quantum computers which could efficiently find discrete logarithms, it may be that none of these protocols will remain secure within a relatively short time.

5.5 Diffie–Hellman Protocols with Explicit Authentication

In this section we look at protocols that add information to authenticate the Diffie–Hellman messages exchanged between the parties. In many cases the additional algorithms used for authentication are made deliberately independent of the Diffie–Hellman exchange, which reduces the chance of 'unfortunate' interaction between different protocol fields. Two features are common in the protocols in this section, in contrast to those in Sect. 5.4.

- The shared secret is usually the ephemeral Diffie–Hellman key. In this case the protocol will usually possess forward secrecy as long as the shared secret and

the long-term keys of the principals are completely independent. There are exceptions in which protocol designers have reused the ephemeral Diffie–Hellman inputs in the authentication information.
- The ephemeral public keys are often signed by the principal generating them. In this case key compromise impersonation is always avoided, since knowing A's private key does not help the adversary to forge B's signature. A similar effect occurs if the ephemeral public keys are encrypted with the partner's public key.

From the range of protocols we have identified in the literature, there are three typical methods of adding authentication to Diffie–Hellman key exchange.

1. The protocols messages, including the ephemeral Diffie–Hellman keys, are signed with a digital signature.
2. A message authentication code (MAC) is computed and added to the messages, including the ephemeral Diffie–Hellman keys. The key used for this MAC is typically derived from long-term keying material of the parties, such as a static Diffie–Hellman key.
3. A proof of knowledge is added to the messages, proving that the message is well-formed. This does not provide explicit authentication, but links the sender to an entity who generated the ephemeral secret key.

It is not immediately obvious why using explicit authentication should be useful in comparison with the protocols in Sect. 5.4. Adding such fields increases message lengths and usually will increase computational requirements. Nevertheless, the type of protocols we examine in this section are more common in the real world than those in Sect. 5.4. Some possible reasons are:

- entity authentication and key confirmation can be provided, even in one round;
- full forward secrecy can be provided in one round;
- generic solutions are available using standard primitives for signatures and MACs, which can be replaced when new algorithms become available.

5.5.1 Generic Constructions for Authenticated Diffie–Hellman

In later subsections we will see several concrete constructions for protocols which add explicit authentication to Diffie–Hellman key exchange in order to provide various properties. Here we mention a few generic constructions which have typically been designed to achieve strong security goals, perhaps with less than optimal efficiency.

Boyd and González-Nieto [142] and, later, Cremers and Feltz [234, 235] provided compilers to convert a one-round protocol into one which provides *full forward secrecy*, still in only one round. The former compiler works by adding a MAC to the exchanged messages using a shared key derived from a static Diffie–Hellman key. The latter compiler uses signatures instead for the same purpose. Although both provide full forward secrecy according to the usual definitions, the Cremers and Feltz compiler based on signatures provides security in a slightly stronger model. Generally, these compilers do not result in very efficient protocols, since they have to be

applied to protocols which are already secure in models, such as the eCK model, that provide weak forward secrecy.

Bergsma *et al.* [90] constructed a generic one-round key agreement protocol relying on any non-interactive key exchange (NIKE) and secure signature scheme. Their protocol is not particularly efficient but it does have the benefit that it has a security proof without relying on random oracles, and in a strong model providing eCK security and full forward secrecy in one round. Protocol 5.23 shows the generic protocol of Bergsma *et al.* instantiated using Diffie–Hellman for the generic NIKE scheme.

Shared information: Pseudo-random function family F.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A, \mathsf{Sig}_A(t_A)}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B, \mathsf{Sig}_B(t_B)}$	
$Z_1 = y_B^{x_A}, Z_2 = t_B^{x_A}, Z_3 = y_B^{r_A}, Z_4 = t_B^{r_A}$		$Z_1 = y_A^{x_B}, Z_2 = t_A^{x_B}, Z_3 = y_A^{r_B}, Z_4 = t_A^{r_B}$
$T = t_A, t_B$		$T = t_A, t_B$

$$\mathbf{K} = F_{Z_1}(T) \oplus F_{Z_2}(T) \oplus F_{Z_3}(T) \oplus F_{Z_4}(T)$$

Protocol 5.23: Bergsma–Jager–Schwenk protocol instantiated with Diffie–Hellman

Bergsma *et al.* [90] proved security of Protocol 5.23 (with any suitable NIKE) in the eCK-PFS model of Cremers and Feltz [235]. The signature scheme applied was required to be *deterministic* in order to avoid problems where leakage of randomness leads to leakage of long-term signing keys.

5.5.2 STS Protocol

The Station-to-Station (STS) protocol, due to Diffie *et al.* [253], adds a digital signature to the exchanged messages to provide authentication for the Diffie–Hellman protocol. In addition, the shared secret is used to provide further assurances. Protocol 5.24 shows the main version of the STS protocol; a variant in which the encryption is replaced by the use of a MAC is shown in Protocol 5.26. The shared secret is $\mathbf{Z} = g^{r_A r_B}$ and the session key $\mathbf{K}$ is derived from $\mathbf{Z}$ in some unspecified way.

Because the shared secret is the ephemeral Diffie–Hellman key, forward secrecy is provided by the STS protocol. Also, the signatures provide protection against key compromise impersonation, since if a long-term key is lost this does not help an

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{t_B, \{\mathsf{Sig}_B(t_B, t_A)\}_{\mathbf{K}}}$	$\mathbf{Z} = t_A^{r_B}$
Decrypt/verify signature		
	$\xrightarrow{\{\mathsf{Sig}_A(t_A, t_B)\}_{\mathbf{K}}}$	Decrypt/verify signature

Protocol 5.24: STS protocol of Diffie, van Oorschot and Wiener

adversary to forge a signature of a different entity. Lowe's attack [502] on the STS protocol was discussed in Sect. 1.5.6. This shows that the protocol does not provide mutual belief in the key, or strong entity authentication.

Diffie *et al.* [253] explained that the symmetric encryption in Protocol 5.24 is essential in order to prevent unknown key-share attacks. Specifically, without the encryption, the adversary C could remove the signature of A in the final message and replace it with C's own. The result is that A and B both complete the protocol but B believes that the key is shared with C, while A believes that it is shared with B. An unknown key-share attack remains possible if the adversary can obtain a certificate for a public key that is identical to that of the victim (who can be either A or B). In a similar way to Attack 5.7, the adversary simply relays messages between A and B.

Unknown key-share attacks can be prevented by including the identity of the partner entity in the signatures exchanged. Moreover, this change provides an explicit indication of the peer entity so that a stronger form of entity authentication is achieved (in particular, Lowe's attack no longer applies). In addition, there no longer seems to be a need for the symmetric encryption in the protocol. This leads to the STS variant shown in Protocol 5.25. An essentially identical protocol has been proven secure using the model of Bellare *et al.* [72, 178] (see also the description by Blake-Wilson and Menezes [109]).

The role of the encryption mechanism in the STS protocol can be taken by a MAC, since its purpose is to ensure that the signing party has possession of the session key and not to provide confidentiality. This leads to the STS variant [253] shown in Protocol 5.26. A potential disadvantage in comparison with Protocol 5.24 is the increased length of the second and third messages.

Blake-Wilson and Menezes [110] proposed an unknown key-share attack on Protocol 5.26. As in the description above, the adversary must register a new public key, but since the adversary can choose the correct private key the certification process cannot be used to prevent the attack. Given a signature $\mathsf{Sig}_A(t_A, t_B)$ used in the protocol run, the adversary C must find a new public key such that $\mathsf{Sig}_C(t_A, t_B) = \mathsf{Sig}_A(t_A, t_B)$. There are two problems that the adversary faces in en-

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
Verify signature	$\xleftarrow{t_B, \mathsf{Sig}_B(t_B, t_A, ID_A)}$	$\mathbf{Z} = t_A^{r_B}$
$\mathbf{Z} = t_B^{r_A}$	$\xrightarrow{\mathsf{Sig}_A(t_A, t_B, ID_B)}$	Verify signature

Protocol 5.25: Modified STS protocol

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{t_B, \mathsf{Sig}_B(t_B, t_A), \mathsf{MAC}_{\mathbf{K}}(t_B, t_A)}$	$\mathbf{Z} = t_A^{r_B}$
Verify signature and MAC	$\xrightarrow{\mathsf{Sig}_A(t_A, t_B), \mathsf{MAC}_{\mathbf{K}}(t_A, t_B)}$	Verify signature and MAC

Protocol 5.26: STS protocol using MACs

gineering this situation. The first is to find such a key that satisfies this *duplicate signature* property, and the second is to have the new key certified in real time.

Blake-Wilson and Menezes showed that the first problem can be solved for many popular signature schemes. The second problem has features in common with Attack 5.9 on the MQV protocol. Once this substitution is achieved, the attack proceeds again, with the adversary simply relaying messages between the parties. However, this attack can work only against the version of STS using MACs (Protocol 5.26), and not against Protocol 5.24, because the adversary needs to see the signature in order to calculate the new public key.

Since the adversary knows the private key of the new public key in this second attack, asking certifiers to check this knowledge is not sufficient to prevent the attack. Instead, Blake-Wilson and Menezes suggested a number of preventative measures. The generic solution of including the principal identities in the key derivation function still works. Blake-Wilson and Menezes suggested including the identities within the signature as a preventative measure. However, Baek and Kim [49] showed that this is still not sufficient, and instead recommended including identities explicitly within the MAC as well as in the signature.

5.5.3 Oakley Protocol

In this section, and the following five, we consider protocols designed for practical use on the Internet. In contrast to the separate key establishment protocols that we mainly focus on in this book, these sets of protocols can be customised by negotiation of various protocol features during the protocol run itself. This flexibility can result in new security threats. More examples of such threats are discussed in Chap. 6 with respect to TLS.

Oakley is defined in the IETF's RFC 2412 [596] from 1998. The basic design is similar to the STS protocol but the specification emphasises the following differences.

- As part of the protocol, principals A and B choose cookies CK_A and CK_B, respectively, which are used to mitigate denial-of-service attacks. The correct cookie must be returned in subsequent messages. The exact format of the cookies is not specified and it is optional whether or not cookies are exchanged before the key exchange begins.
- The protocol includes negotiation of cryptographic algorithms that are used to support the protocol, including the encryption algorithm, key derivation function and authentication method. The group used for Diffie–Hellman itself can also be negotiated.
- There need be no use of encryption (or a MAC) for authentication of the session key. In some versions this allows the session key to be derived after completion of the protocol if desired.

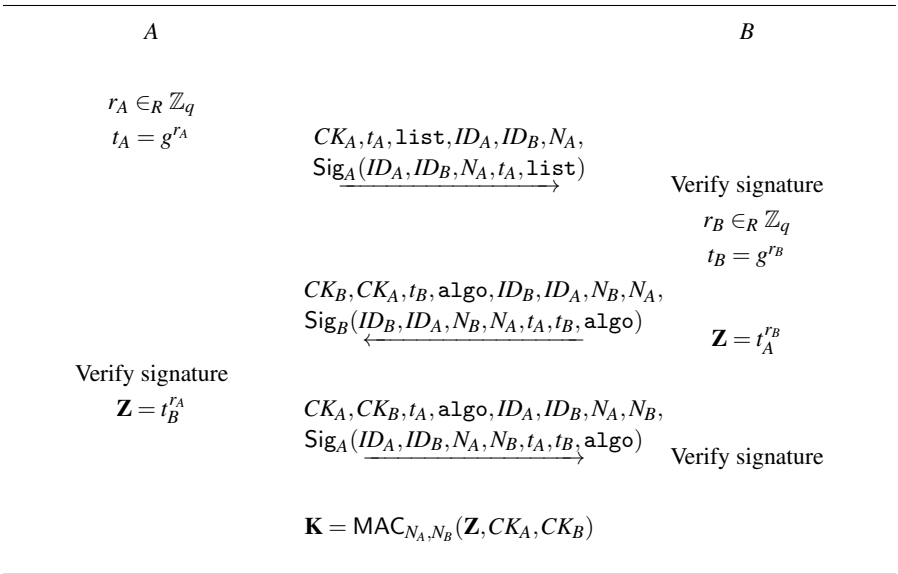
Several specific protocols are given in the Oakley documentation. We examine simplified versions of three of these, ignoring some plaintext fields such as a string indicating the name of the protocol itself. A further option does not use Diffie–Hellman, and therefore fails to provide forward secrecy.

Protocol 5.27 is named 'aggressive' because it anticipates that certain possible problems will not arise. In particular:

- it only works when B can use the same group as initially proposed and used by A in the first message;
- it does not force either principal to respond with a cookie before the other principal stores state information regarding the connection.

In the first message of Protocol 5.27, principal A offers a list of possible algorithms, `list`, for encryption, hashing and authentication, which A is prepared to use for this session. B responds with a particular algorithm set, `algo`, in message 2. The nonces N_A and N_B are chosen by A and B, respectively, and are of unspecified size. Protocol 5.27 has a similar basic structure to Protocol 5.25, but here A needs to calculate two signatures instead of one.

As well as checking all received signatures, both parties must ensure that the received ephemeral values are not the degenerate values 1 or $p-1$. A specific key derivation function is used to calculate the session key from the shared secret $\mathbf{Z} = g^{r_A r_B}$ as follows:



Protocol 5.27: Oakley aggressive-mode protocol

$$\mathbf{K} = \mathsf{MAC}_{N_A,N_B}(\mathbf{Z}, CK_A, CK_B). \tag{5.1}$$

Oakley was specifically designed to provide forward secrecy, which follows from the use of the ephemeral Diffie–Hellman key as the shared secret. In addition, the signatures of both parties prevent key compromise impersonation. Unknown key-share attacks are prevented by inclusion of the principal identities in the signatures, similarly to Protocol 5.25.

Protocol 5.28 is an Oakley variant designed to prevent disclosure of user identities, and for this reason the user identities are encrypted instead of being signed. The identity B' can be thought of as a domain name for B and may be a generic identity at the location of B; for example, if B is an entity in a corporate environment, B' may be the public key of the corporation, with a known public key. The reason for using the generic name is so that the recipient node is able to decrypt without receiving the identity of the recipient in plaintext. An interim key $K = H(N_A, N_B)$ is used with the hash function H in order to authenticate the exchange.

In this variant the shared secret is again $\mathbf{Z} = g^{r_A r_B}$, and the session key is defined using Eqn (5.1). Forward secrecy and protection against unknown key-share attacks are both provided as in Protocol 5.27. Unknown key-share attacks are prevented because of the encryption of the N_A and N_B values used in the MAC calculation.

Protocol 5.29 is a 'conservative' Oakley variant which exchanges cookies before the key exchange itself commences. This is the way that cookies were originally designed to work, so that some assurance about the origin of the request is obtained before the computationally expensive part of the protocol begins, and without requiring the responder to store any state. The first message is simply an indication to

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$CK_A, t_A, \texttt{list}, B',$ $\mathsf{Enc}_{B'}(ID_A, ID_B, \mathsf{Enc}_B(N_A))$ $\longrightarrow$	
		$r_B \in_R \mathbb{Z}_q$ $K = H(N_A, N_B)$ $t_B = g^{r_B}$
	$CK_B, CK_A, t_B, \texttt{algo},$ $\mathsf{Enc}_A(ID_B, ID_A, N_B),$ $\mathsf{MAC}_K(ID_B, ID_A, t_B, t_A, \texttt{algo})$ $\longleftarrow$	
$K = H(N_A, N_B)$ Verify MAC		$\mathbf{Z} = t_A^{r_B}$
	$CK_A, CK_B,$ $\mathsf{MAC}_K(ID_A, ID_B, t_A, t_B, \texttt{algo})$ $\longrightarrow$	
$\mathbf{Z} = t_B^{r_A}$		Verify MAC

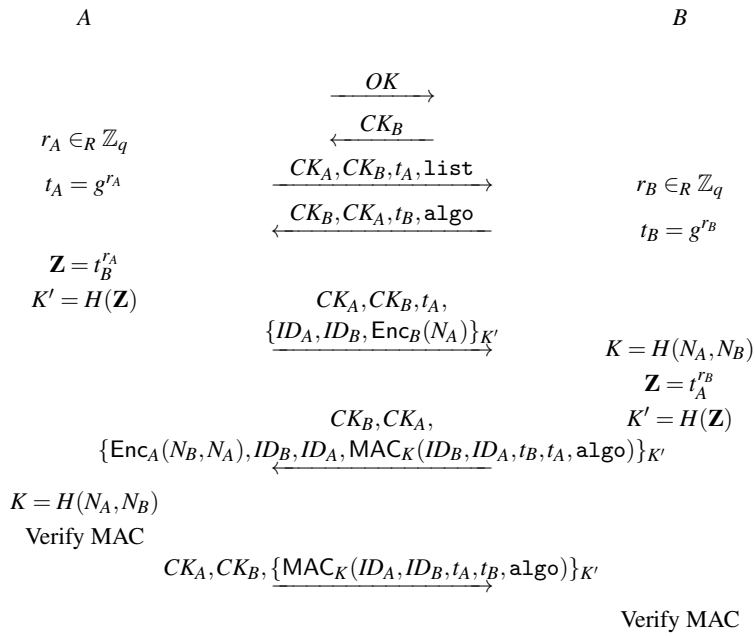
Protocol 5.28: Alternative Oakley protocol

B that the protocol should commence. The consequence of the use of cookies in this manner is the increased number of messages and rounds required in the protocol.

As in Protocol 5.28, the principal identities are protected in Protocol 5.29 but here a temporary key K', derived from the shared secret, is used for this purpose. The first four messages exchanged are therefore anonymous. Once again, the session key is defined using Eqn (5.1). Forward secrecy and protection against key compromise impersonation and unknown key-share attacks appear to be provided as in Protocol 5.28. However, there is an important difference in this protocol which could cause problems, as we now explain.

The return of the encrypted nonce sent from B to A in Protocol 5.29 opens up the possibility of an interesting attack in which the adversary masquerades as A, and subsequently uses the real A as a decryption oracle. The attack depends on an assumption that the message field $\mathsf{Enc}_A(N_B, N_C)$ sent by B will be interpreted by A as the encryption of a single nonce when replayed by C in a second protocol run. This need not be unreasonable, since the length of nonces is variable in the Oakley specification. In a correct implementation, each nonce should have its length specified and therefore A should detect the problem and abort the protocol. On the other hand, the attack would still work if A were to ignore the second nonce and simply decrypt N_B, discarding the second nonce. Therefore we believe that a careless implementation could be vulnerable to the attack. In more detail, the attack proceeds as follows.

1. C masquerades as A and sends $t_C = g^{r_C}$ to B so that B accepts the temporary key $K' = H(\mathbf{Z}_{CB})$ as shared with A.
2. C sends $\{ID_A, ID_B, \mathsf{Enc}_B(N_C)\}_{K'}$ to B in the fifth message flow. B will choose a nonce N_B and calculate $K = H(N_C, N_B)$.



Protocol 5.29: Oakley conservative protocol

3. B sends $\{\mathsf{Enc}_A(N_B, N_C), \ldots\}_{K'}$ to C as part of the sixth message flow. The adversary can remove the symmetric encryption to obtain $\mathsf{Enc}_A(N_B, N_C)$.
4. C starts a second protocol run with A. This proceeds normally until the fifth message flow, when C sends $\{ID_C, ID_A, \mathsf{Enc}_A(N_B, N_C)\}_{K''}$, where K'' is a temporary key shared between C and A.
5. As long as A interprets the pair (N_B, N_C) as a single nonce from C, she will reply with $\{\mathsf{Enc}_C(N_A, N_B, N_C)\}_{K''}$. C can extract N_B and so complete the first protocol run with B.

5.5.4 SKEME Protocol

SKEME is due to Krawczyk [450]. Like Oakley, it is a set of protocols suitable for negotiation of services in a general networked environment. In addition to it providing establishment of a good key, Krawczyk [450] stated additional requirements for SKEME as follows.

Periodic key refreshment: It should be possible to update the session key used at low computational cost. A special mode of the protocol is included for this purpose.

Forward secrecy: This property should be available as an option. In some cases it may be sacrificed for efficiency gains.

Key separation: This is a principle governing how the shared secret is used to obtain session keys. Different cryptographic functions are provided with independent keys.

Privacy and anonymity: As well as providing confidentiality of user data, it may be desirable to hide the identities of the communicating parties. For this reason the SKEME protocol avoids digital signatures, which can always be used at least to confirm the identity of the signer.

Denial-of-service protection: SKEME was designed to employ cookies in the same way as Oakley. This requires an additional initial message exchange that is omitted from the descriptions below (as well as in those of Krawczyk [450]).

Efficiency and simplicity: The design tries to reduce the options and use compact and uniform message formats. In addition, efficient algorithms such as hash functions are preferred over the use of digital signatures.

Support for multiple models and algorithms: Certificate and shared key models are supported. Cryptographic algorithms are specified in a generic sense so as to avoid dependence on particular primitives.

There are a number of possible protocols.

1. The basic mode, shown as Protocol 5.30, which uses the public keys of both parties to provide forward secrecy and anonymity of the principals.
2. A version using public keys but without forward secrecy, which is described as Protocol 5.42 later.
3. A version using an existing shared secret and providing forward secrecy. This is similar to Protocol 5.30, but the encrypted field in messages 1 and 2 is omitted and instead the previously shared key is used as the key K_0 for the MAC.
4. A rekeying mechanism.

In Protocol 5.30, the nonces N_A and N_B are shown as being chosen in $[0, 2^L]$ for some security parameter; the size of L was not specified by Krawczyk, who said only that they should be 'chosen as (pseudo-)random values'. The shared secret is $\mathbf{Z} = g^{r_A r_B}$, with session key $\mathbf{K} = H(\mathbf{Z})$.

Krawczyk presented arguments that the simple modes of SKEME, in which K_0 is previously shared, can be proven secure in the Bellare–Rogaway model. Moreover, Bellare *et al.* [72] showed that Protocol 5.30 can be derived as an output of their modular design method, and therefore the protocol inherits a formal proof of security in their model.

5.5.5 Internet Key Exchange

As part of an initiative to secure the foundations of the Internet, the Internet Engineering Task Force developed an IP security protocol, known simply as IPSec [425], to provide security services to any application running over the Internet Protocol (IP).

Shared information: Security parameter L.

A		B
$N_A \in_R [0, 2^L]$ $r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{\mathsf{Enc}_B(ID_A, N_A), t_A}$	$N_B \in_R [0, 2^L]$ $r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
Verify MAC $K_0 = H(N_A, N_B)$ $\mathbf{Z} = t_B^{r_A}$	$\mathsf{Enc}_A(N_B), t_B,$ $\xleftarrow{\mathsf{MAC}_{K_0}(t_A, t_B, ID_B, ID_A)}$ $\xrightarrow{\mathsf{MAC}_{K_0}(t_B, t_A, ID_A, ID_B)}$	$K_0 = H(N_A, N_B)$ Verify MAC $\mathbf{Z} = t_A^{r_B}$

Protocol 5.30: SKEME protocol, basic mode

The concerns of IPSec go considerably beyond key establishment, so here we examine only a very small part of the development. The reader interested in understanding the complexity of designing an overall system to secure Internet communications is encouraged to consult the considerable literature on IPSec. We refer to a number of useful sources in this section.

During the 1990s a number of different key establishment protocols were proposed for inclusion within the IPSec solution. These included Oakley (see Sect. 5.5.3), SKEME (see Sect. 5.5.4) and Photuris [677]. Through a long and complex evolution a single proposed Internet standard emerged in 1998, known as Internet Key Exchange, or IKE [351]. Owing to many criticisms of the protocols [274, 610, 780, 781], a new version of IKE, IKEv2, was eventually proposed and standardised; we examine IKEv2 in Sect. 5.5.6.

The IKE protocol is strongly related to the Oakley and SKEME protocols. It has two phases. The first phase is concerned with establishing a secure channel by key exchange. The second phase is concerned with using the secure channel to set up sessions known as *security associations*, which can be used to protect the confidentiality and/or integrity of exchanged data. However, the need for this split has been questioned [610] and the indications are that the revised version of IKE will not have two phases. We will not consider the second phase further here.

The IKE protocols employ cookies similar to those of the Oakley protocol. However, the protocol specification requires some state to be recorded even in the first message exchange in order to check a returned cookie. Therefore, although there is some mitigation of denial-of-service attacks, the result is not as effective as when stateless cookies are used. In addition it is optional whether cookies are exchanged at all before the key exchange begins. This weakening of the cookie mechanism has

received considerable criticism [610]. The protocol also includes negotiation of cryptographic algorithms that are used to support the protocol, including the encryption algorithm, key derivation function, and authentication method. The group used for Diffie–Hellman itself can also be negotiated.

Because IKE emerged as a compromise solution, it is not surprising to learn that it contains a lot of different options. It contains no fewer than eight key establishment protocols, which can be divided into two sets of four according to whether *main mode* or *aggressive mode* is used. The protocols in main mode use six messages, which can be divided into three pairs with the following aims.

Stage 1. Exchange cookies and agree on the algorithms to be used.
Stage 2. Exchange ephemeral Diffie–Hellman keys and nonces.
Stage 3. Perform entity authentication and key confirmation.

This division presents an attractive framework, but the large number of exchanges is another point of significant criticism. The aggressive mode collapses most of the functionality into only three message exchanges in a more conventional manner. In general the protocols in aggressive mode suffer from two limitations in comparison with the main mode.

- Use of cookies is of limited value, since the responder needs to start the computationally expensive part of the protocol before the cookie exchange is complete.
- Because the Diffie–Hellman exchange begins in the first message, the responding party can only complete the protocol if the group used is supported by the responding party. (This can be negotiated in the main mode.)

Protocol 5.31 shows one of the four main versions of the IKE protocol; this one uses digital signatures for authentication. In Stage 1, principal A offers a list of acceptable algorithms, `list`, that A is prepared to use, and B responds with a particular algorithm set, `algo`. Once the ephemeral Diffie–Hellman keys have been exchanged in the second pair of messages, both parties can derive the shared secret and form the temporary key $KM = \mathsf{MAC}_{N_A,N_B}(\mathbf{Z})$, which is used to form the MAC of the protocol parameters in the final message exchange. We have used the suggestion of Ferguson and Schneier [274] in replacing the more general keyed hash function used in the IKE specification with a MAC, since it seems that a MAC is what is required here. Indeed, HMAC [71] is the only explicit suggestion in the standard for this function. In addition to KM, several other keys are derived from $\mathbf{Z}$. One of these is used to encrypt the final message exchange and is denoted by K' in Protocol 5.31. We have not specified the function used to derive K', but the IKE specification does so in terms of a generic keyed hash function. Although examples of possible functions are given, Ferguson and Schneier [274] criticised the lack of any description in the specification of the properties that such a function should have.

Since the principal identities are hidden in all messages, the protocol provides anonymity against a passive eavesdropper. However, Perlman and Kaufman [610] pointed out that an active adversary can discover the identity of A by masquerading as B during the first two phases, and thereby obtain the correct value of K'. Of

Shared information: Symmetric key derivation function $SKDF$. Security parameter L with $64 \leq |L| \leq 2048$.

A		B
	Stage 1	
	$\xrightarrow{CK_A, \texttt{list}}$	
	$\xleftarrow{CK_B, \texttt{algo}}$	
	Stage 2	
	$CK = CK_A, CK_B$	
$N_A \in_R [0,L], r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{CK, t_A, N_A}$	$N_B \in_R [0,L], r_B \in_R \mathbb{Z}_q$
$\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{CK, t_B, N_B}$	$t_B = g^{r_B}$
		$\mathbf{Z} = t_A^{r_B}$
	Stage 3	
	$KM = \mathsf{MAC}_{N_A, N_B}(\mathbf{Z})$	
	$K' = SKDF(KM)$	
	$M_{AB} = \mathsf{MAC}_{KM}(t_A, t_B, CK_A, CK_B, \texttt{list}, ID_A)$	
	$M_{BA} = \mathsf{MAC}_{KM}(t_B, t_A, CK_B, CK_A, \texttt{list}, ID_B)$	
	$\xrightarrow{CK, \{ID_A, \mathsf{Sig}_A(M_{AB})\}_{K'}}$	
		Decrypt and verify MAC and signature
	$\xleftarrow{CK, \{ID_B, \mathsf{Sig}_B(M_{BA})\}_{K'}}$	
Decrypt and verify MAC and signature		

Protocol 5.31: IKE main protocol using digital signatures

course, the adversary cannot forge B's signature and so is unable to complete the final message of the protocol. But if A were a mobile station who wished to protect her location then the attack could be deemed successful. Perlman and Kaufman suggested that, to protect A's identity, the contents of the final message should be moved back to the fourth message. Although this means that a similar active attack is now possible against B's identity, they argued that the initiator is more vulnerable to such an attack in a typical application involving a mobile client connecting to a server. Moreover, this suggestion reduces the number of messages by one.

Ferguson and Schneier [274] pointed out that a reflection attack is possible on Protocol 5.31. The adversary masquerades as B and simply copies and returns the values CK_A and t_A in place of CK_B and t_B in the first two stages. Then A will also accept her own signature reflected back in the final message. Of course, the attack is only meaningful if A will accept her own identity as the identity of her peer principal. Since IKE may be configured only to authenticate IP (machine) addresses, this may not be an unreasonable assumption. This is an attack against the key confirmation and entity authentication properties; the adversary does not obtain the shared secret. The attack also applies to the main mode of IKE, when authentication is based on a previously shared secret.

The signatures in the final two messages are intended to provide authentication and integrity of the parameters that have been agreed earlier in the protocol. However, it is an oversight that the list of algorithms that was offered by A is included in the MAC, while the set, `algos`, accepted by B is not part of the calculation. Potential problems with this omission were described by both Zhou [780] and Ferguson and Schneier [274]. The adversary can cause A and B to believe that different sets of algorithms were accepted, or make A accept a weak set and attack the algorithms in a subsequent session.

Meadows [536] performed a formal analysis of IKE using the NRL Analyzer. She identified a number of ambiguities in the specification which could lead to insecure implementations. Indeed, the incompleteness of the specification has been a recurring theme in the several analyses of IKE that have been published. Meadows also pointed out that several of the IKE protocols do not provide strong entity authentication in the sense discussed in Chap. 2. (Meadows refers instead to 'penultimate authentication' as the property that, when an entity accepts a key, the peer entity should have taken part in the earlier part of the protocol. Here we reinterpret this to mean that B does not achieve assurance that A has knowledge of B as the peer entity.) In Protocol 5.31, the adversary C can sit between A and B masquerading as A to B while A believes the protocol is being executed with C. The protocol messages are relayed unchanged by C, but the final message from B is simply deleted. Consequently, A will not accept but B completes the protocol properly. The effect is the same as in Lowe's attack on STS discussed in Chap. 2 and cannot be accepted as an attack unless strong entity authentication is a protocol goal. Nevertheless, this extra property could easily be achieved by including the peer identity in the signatures of the last two messages of the protocol.

5.5.6 SIGMA and Internet Key Exchange v2 (IKEv2)

In response to the serious deficiencies in the original IKE protocols, highlighted in the previous section, a new version of IKE (IKEv2) was introduced. Originally proposed in 2005, the protocol has been updated since and is currently a proposed standard in RFC 7296 [420]. According to the standard, the main differences between IKEv2 and the original IKE are simplification, increased efficiency, increased robustness against denial-of-service, and repair of cryptographic weaknesses.

There are numerous variants and extensions of the IKEv2 protocol. All versions employ Diffie–Hellman key exchange and later authenticate this initial exchange using one of several options. We look only at the version which uses signatures to authenticate the Diffie–Hellman values. The signatures are encrypted with a key derived from the initial ephemeral Diffie–Hellman exchange in order to hide the identities of the principals. A passive adversary cannot obtain either of the signatures, but an active adversary can masquerade during the initial unauthenticated messages, obtain the encryption key and reveal the signature (and so the identity) of the first party to sign. In IKEv2, the initiator sends the first signed message, so it is the responder that has stronger assurance against disclosure of identity.

The signature version of IKEv2 is based on a design principle for protocols called the 'Sign-and-MAC' approach, or SIGMA, due to Krawczyk and Canetti [180, 452]. Protocol 5.32 shows one specific instance of the SIGMA approach, designed to protect the identity of the initiator, and hence it is called SIGMA-I.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{t_B, \{ID_B, \mathsf{Sig}_B(t_A,t_B), \mathsf{MAC}_{K_M}(ID_B)\}_{K_E}}$	
Decrypt then verify signature and MAC $\mathbf{Z} = t_B^{r_A}$		$\mathbf{Z} = t_A^{r_B}$
	$\xrightarrow{\{ID_A, \mathsf{Sig}_A(t_B,t_A), \mathsf{MAC}_{K_M}(ID_A)\}_{K_E}}$	
		Decrypt then verify signature and MAC

Protocol 5.32: SIGMA-I protocol

Protocol 5.32 can be seen to have strong similarities with Protocol 5.26, the MAC variant of STS, but there are also some important differences. First, note that there are three symmetric keys derived from the shared secret $\mathbf{Z} = g^{r_A r_B}$:

- **K**, the session key for use after the protocol is complete;
- K_M, the key used for the MAC within the protocol;
- K_E, the key used to encrypt the final two messages.

Each of these keys must be independently derived from **Z** so that revealing one does not compromise any other. The encryption algorithm, denoted $\{\cdot\}_{K_E}$ is used to hide the identities of the participants, and must provide authenticated encryption in order to hide the initiator identity against active adversaries.

Krawczyk [452] explained the design principles behind the SIGMA protocols. One requirement is that each principal should be able to authenticate even without knowing the identity of its peer. This allowed Canetti and Krawczyk [180] to provide security proofs in the post-specified peer model. It may seem strange that the SIGMA protocol uses both signatures and MACs together. The idea [452] is that the signature is used only to authenticate the ephemeral Diffie–Hellman values, while the MAC is used to bind the identities to the key.

Protocol 5.33 shows the (simplified) initial exchanges in the IKEv2 protocol [420] when signatures are used for authentication. The full protocol is much more complex than shown and proceeds with subsequent exchanges used to derive *child security associations*. In contrast to Protocol 5.31, cookies for denial-of-service mitigation are not shown in Protocol 5.33. Although cookies are allowed in the standard, they are not used in the basic protocol, since they would add an extra round of communication; in IKEv2, cookies are added only on demand when a denial-of-service attack is suspected.

As in the SIGMA protocols, the keys K_E and K_M are independently derived from the shared secret and nonces. Also similarly, encryption with K_E should provide authenticated encryption independently from the MAC keyed by K_M.

There are a number of different possible concrete groups specified in RFC 7296 [420] (and related RFC documents cited within it) in which the Diffie–Hellman key exchange can be run with a range of parameter sizes. The proposed groups are included in the set `list` sent in the first message and chosen by the recipient. Note that A has to send t_A before the Diffie–Hellman group is fixed so A must make an assumption that her preferred group will be chosen by B. If this does not happen, the first two messages need to be run again.

Reuse of ephemeral Diffie–Hellman keys is allowed in RFC 7296 [420]; this destroys forward secrecy for the window in which the keys are reused but saves computation. Menezes and Ustaoglu [549] described how small subgroup attacks may be possible in some scenarios with ephemeral-key reuse. They remarked, however, that their attacks do not apply to IKEv2 as long as checks are in place to validate group membership, or groups without small subgroups are used.

In addition to the computational analysis of the core SIGMA protocol of Canetti and Krawczyk [180], Cremers [233] carried out a symbolic analysis of IKEv2 (and also IKEv1) using the Scyther tool. The analysis covered many variations of IKEv2, although this did not include analysis of the identity protection property. Cremers reported a number of previously unknown weaknesses, including a reflection attack on subsequent protocol messages following the initial authentication.

A		B
Stage 1: IKE_INIT		
$N_A \in_R [0,L], r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{\texttt{list}, t_A, N_A}$	$N_B \in_R [0,L], r_B \in_R \mathbb{Z}_q$
$\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{\texttt{algo}, t_B, N_B}$	$t_B = g^{r_B}$
		$\mathbf{Z} = t_A^{r_B}$
Stage 2: IKE_AUTH		
Derive keys K_M, K_E from $\mathbf{Z}, N_A, N_B$		
	$\{ID_A, \mathsf{Sig}_A(\texttt{list}, t_A, N_A, N_B, \mathsf{MAC}_{K_M}(ID_A))\}_{K_E} \xrightarrow{}$	
		Decrypt then verify MAC and signature
	$\{ID_B, \mathsf{Sig}_B(\texttt{algo}, t_B, N_B, \mathsf{MAC}_{K_M}(ID_B))\}_{K_E}$	
Decrypt then verify MAC and signature		

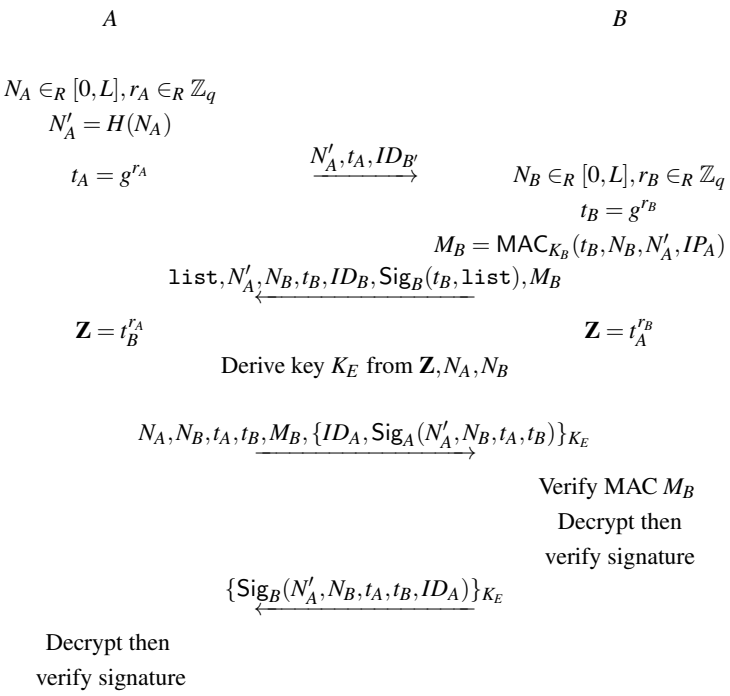
Protocol 5.33: IKEv2 protocol, initial exchanges

5.5.7 Just Fast Keying

A protocol strongly related to IKEv2, called Just Fast Keying (JFK), was proposed by Aiello *et al.* [23]. Like the SIGMA protocols examined in Sect. 5.5.6, JFK comes in two versions, called JFKi and JFKr, each designed for identity protection against one of the two parties involved. The protocols have in-built mechanisms to protect against denial of service. Protocol 5.34 shows a simplified version of the JFKi protocol, designed to protect the identity of the initiator A, who may be a client interacting with a server B. In our description we have omitted the security association information which includes details of the cryptographic services to be provided in the subsequent connection.

There is some different notation used in this description. In the first message, $ID_{B'}$ is sent by A. This is treated as 'an indication' of the identity which A wishes to connect to. This can be the same as or different from the identity ID_B sent back in the second message. The IP address, IP_A, is used in the additional MAC sent in message 2 and returned in message 3. This is part of the denial-of-service-resistance mechanisms discussed further below. The field `list` contains all Diffie–Hellman groups supported by B, and the algorithms to be used for authenticated encryption and key derivation during the protocol. If A has already chosen a group disallowed

Information known to B: MAC key K_B



Protocol 5.34: JFKi protocol

by B for its choice of t_A in the first message, then B will implicitly reject and A will need to restart using an acceptable group.

As is typical of other protocols in Sect. 5.5, JFK is based on the idea of signing an ephemeral Diffie–Hellman exchange. Note that although JFKi makes use of a MAC, independent of the authenticated encryption as in IKEv2, the MAC key K_B is a private one known only to B, in contrast to the shared MAC key used in the IKEv2 and SIGMA protocols. Indeed, this MAC serves a rather different purpose, namely to allow B to avoid saving per-session state, since B can verify the authenticity of the parameters returned by A in the third message.

In the first message, the initiator A does not initially send its clear nonce N_A but instead sends a hash of this, N'_A. This is part of the denial-of-service mitigation measures of JFK. The idea is that B will return N'_A to the specified IP address IP_A and only the genuine source of N'_A will be able to provide the correct inverse value N_A, which is checked by B before engaging in the computationally expensive parts of the protocol.

Although we have not indicated it in Protocol 5.34, it is worth mentioning that a design goal of the JFK protocols is to provide flexibility in terms of forward se-

crecy. It is expected that principals, especially servers in a client–server scenario, will sometimes reuse ephemeral Diffie–Hellman keys. Aiello [23] and Abadi *et al.* [4] discussed a *forward secrecy interval*, namely the window between generation of new ephemeral keys. During the interval the ephemeral keys will be stored like long-term keys, and their compromise would lead to compromise of all keys agreed during the interval, but not those agreed in earlier intervals.

Abadi *et al.* [4] applied the pi calculus to analyse the security of JFK. They reported generally positive results with some minor exceptions regarding identity protection. Smith *et al.* [682] analysed the denial-of-service-resistance properties of JFK using a framework proposed by Meadows [539] and also proposed additional denial-of-service resistance mechanisms using client puzzles.

5.5.8 Arazi's Protocol

Arazi's protocol [37] was designed to integrate Diffie–Hellman with the Digital Signature Standard (DSS) [577] in such a way that computation is saved. This is done by overloading the random value chosen by each party to serve two roles: one as the ephemeral Diffie–Hellman private key and one as part of the signature. The protocol has an elegant design and is efficient, but pays for these advantages with some security weaknesses.

Shared information: Hash function H for DSS signature.

A		B
$r_A \in_R \mathbb{Z}_q, t_A = g^{r_A}$		
$s_A = r_A^{-1}(H(t_A) + x_A t_A) \bmod q$		
	$\xrightarrow{t_A, s_A}$	Verify signature $(t_A \bmod q, s_A)$
		$r_B \in_R \mathbb{Z}_q, t_B = g^{r_B}$
		$s_B = r_B^{-1}(H(t_B) + x_B t_B) \bmod q$
Verify signature $(t_B \bmod q, s_B)$	$\xleftarrow{t_B, s_B}$	
$Z = t_B^{r_A}$		$\mathbf{Z} = t_A^{r_B}$

Protocol 5.35: Arazi's key agreement protocol

In Protocol 5.35, the value t_A is the ephemeral Diffie–Hellman public value, while s_A is calculated so as to make the pair $(t_A \bmod q, s_A)$ the DSS signature of t_A. The values t_B and s_B are calculated in a symmetrical fashion. Both A and B are able to check the DSS signatures from the received messages and, if they are correct, they calculate the shared secret as $\mathbf{Z} = g^{r_A r_B}$.

Forward security is not provided, since if x_B, say, becomes known then r_B may be found from s_B and t_B using the same equation used by B to calculate s_B. Furthermore,

Nyberg and Rueppel [586] showed that knowledge of one shared secret allows all subsequent shared secrets to be found by a passive eavesdropper. This may be seen from the following equality:

$$\mathbf{Z}^{s_A s_B} = g^{H(t_A)H(t_B)+x_A x_B t_A t_B} y_B^{H(t_A)t_B} y_A^{H(t_B)t_A}.$$

If $\mathbf{Z}$ is known then $g^{x_A x_B}$ (which is the static Diffie–Hellman key) may be found, since all other values are known. When a new instance of the protocol is run between A and B, the equation can be used again to find the new $\mathbf{Z}$ value. Nyberg and Rueppel suggested that the overloading of the random values r_A and r_B was the root of the problem.

Later, Brown and Menezes [159] showed that a key recovery attack is also possible on Protocol 5.35. This attack follows the same basic idea as the attacks of Lim and Lee discussed in Sect. 5.3.3, but in addition uses recent lattice-based attacks on digital signatures to recover the key when only the lower significant bits of r_B are known in several runs.

Several follow-up papers have attempted to improve upon the original design of Protocol 5.35 by adding extra components or computing in different algebraic groups [361, 398, 612]. The drawback of most of these variants is that they lose the original attractive properties of the Arazi protocol, particularly with respect to efficiency. Even though they may offer improvements in some sense compared with the original protocol, it is questionable whether there is any advantage compared with more modern protocols such as those examined in Sect. 5.4.

5.5.9 Lim–Lee Protocols

Lim and Lee [491] proposed five key agreement protocols based on Diffie–Hellman key exchange. One of these that uses only static Diffie–Hellman was described above as in Protocol 5.5. A second is essentially the MTI A(0) key agreement protocol with a special key confirmation mechanism. The remaining three protocols are similar to Arazi's protocol [37] in that they include a signature by each party that reuses the Diffie–Hellman parameters.

Protocol 5.36 uses the signature scheme of Schnorr [657] to authenticate the messages. The shared secret is the ephemeral Diffie–Hellman key but in this case forward secrecy is not provided. This is because if x_B becomes known then r_B can be recovered from the public values s_B and E and then $\mathbf{Z} = t_A^{r_B}$ can be calculated.

Lim and Lee [491] showed that if an adversary obtains one $\mathbf{Z}$ value then all subsequent values may be found in a similar way to the attack of Nyberg and Rueppel described in Sect. 5.5.8. They pointed out that this attack may be avoided if a one-way key derivation function is used, and the adversary is assumed to be able to obtain only $\mathbf{K}$ rather than $\mathbf{Z}$.

A variant of Protocol 5.36 redefines the value E as $E = ((\mathbf{Z} \oplus t_A) \bmod q) \bmod 2^t$ for a security parameter t and adjusts the checks made by A and B accordingly. This has the advantage that there is no extra exponentiation required to find the shared secret, and so computation is saved.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$E = H(t_A, t_B)$
	$\xleftarrow{s_B, E}$	
$t_B = g^{s_B} y_B^E$		$s_B = r_B - x_B E \bmod q$
$E \stackrel{?}{=} H(t_A, t_B)$		$\mathbf{Z} = t_A^{r_B}$
$s_A = r_A - x_A E \bmod q$		
$\mathbf{Z} = t_B^{r_A}$	$\xrightarrow{s_A}$	$g^{s_A} y_A^E \bmod p \stackrel{?}{=} t_A$

Protocol 5.36: Lim–Lee Schnorr-based protocol

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$\mathbf{Z} = (y_A t_A)^{x_B r_B}$
		$E = ((\mathbf{Z} \oplus r_B) \bmod q) \bmod 2^t$
$\mathbf{Z} = y_B^{(x_A + r_A) r_B}$	$\xleftarrow{r_B, E}$	
$E \stackrel{?}{=} ((\mathbf{Z} \oplus r_B) \bmod q) \bmod 2^t$		
$s_A = r_A - x_A E \bmod q$	$\xrightarrow{s_A}$	$g^{s_A} y_A^E \stackrel{?}{=} t_A$

Protocol 5.37: Lim–Lee Schnorr-based variant

Protocol 5.37 is the final Lim–Lee variant. The shared secret is $\mathbf{Z} = g^{(r_A + x_A) x_B r_B}$, which is unusual in that it is asymmetric with respect to A and B. Notice that q and t are both much smaller than p, so knowledge of E and r_B does not give a simple way to recover $\mathbf{Z}$. Unfortunately this protocol also fails to provide forward secrecy since knowledge of either x_A or x_B alone allows the calculation of $\mathbf{Z}$. If x_A becomes known then r_A can be found from s_A and E and then $\mathbf{Z} = y_B^{(x_A + r_A) r_B}$ can be recalculated. If x_B becomes known then $\mathbf{Z} = (y_A t_A)^{x_B r_B}$ can be recalculated. Since $\mathbf{Z} = S_{AB}^{r_B (E+1)} \cdot y_B^{r_B s_A}$ this protocol shares with Protocol 5.35 the property that knowledge of one $\mathbf{Z}$ value is sufficient to find all subsequent values.

5.5.10 Hirose–Yoshida Protocol

The protocol of Hirose and Yoshida [357] uses a novel signature scheme designed to include elements from the ephemeral keys of both entities. Although the signatures seem similar to those used in the protocols in Sect. 5.5.9, they use an extra random parameter and this allows the weaknesses of the former protocols to be avoided. The description in Protocol 5.38 includes some changes of sign in order to maintain our standard notation for public keys.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{t_A, ID_A}$	$r_B, s_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$ $e_B = H(g^{s_B}, t_B, t_A)$ $w_B = s_B - e_B r_B - e_B^2 x_B \bmod q$
$e_B \stackrel{?}{=} H(g^{w_B}(t_B y_B^{e_B})^{e_B}, t_B, t_A)$ $s_A \in_R \mathbb{Z}_q$ $e_A = H(g^{s_A}, t_A, t_B)$ $w_A = s_A - e_A r_A - e_A^2 x_A \bmod q$ $\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{t_B, e_B, w_B, ID_B}$ $\xrightarrow{e_A, w_A}$	$e_A \stackrel{?}{=} H(g^{w_A}(t_A y_A^{e_A})^{e_A}, t_A, t_B)$ $\mathbf{Z} = t_A^{r_B}$

Protocol 5.38: Hirose–Yoshida key agreement protocol

The signature parameters formed by B are defined as $e_B = H(g^{s_B}, t_B, t_A)$ and $w_B = s_B - e_B r_B - e_B^2 x_B \bmod q$, where s_B is randomly chosen for this signature. The values e_A and w_A are formed by A in an analogous fashion. On receipt of message 2, A must verify the signature. A will respond with message 3 only if this signature is correct. On receipt of message 3, B must verify the signature from A. If the protocol completes successfully then both A and B calculate the shared secret as the ephemeral Diffie–Hellman key $\mathbf{Z} = g^{r_A r_B}$. The extra random values s_A and s_B prevent r_A and r_B becoming known even if x_A and x_B are known, and therefore forward secrecy is provided. As long as the signatures cannot be forged, resistance to key compromise impersonation is provided too.

Unknown key-share attacks are possible on Protocol 5.38 if an adversary can arrange a certified public key equal to that of a victim. The adversary can then simply change either of the identifiers in the first two messages and make the other party believe that the secret is shared with the adversary. Baek and Kim [49] have shown that the more sophisticated unknown key-share attacks constructed by Blake-Wilson and Menezes [110] apply to this protocol too, by showing that duplicate signatures can be found for the signatures used.

5.5.11 Jeong–Katz–Lee TS3 Protocol

Jeong *et al.* [397] proposed three one-round protocols making use of the static Diffie–Hellman value in different ways. The first two protocols are roughly equivalent to having a shared secret equal to the static Diffie–Hellman protocol, and to the Unified Model protocol (Protocol 5.12). Protocol 5.39 shows their third protocol, TS3, which makes use of a MAC to authenticate the ephemeral Diffie–Hellman key. The MAC key, K_M, is derived from the static Diffie–Hellman value – although the details of how to derive K_M are left open, one suggestion is to simply hash the static key $g^{x_A x_B}$ using a cryptographically strong hash function.

Shared information: MAC key K_M derived from static Diffie–Hellman value, $g^{x_A x_B}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$t_A, \mathsf{MAC}_{K_M}(ID_A, ID_B, t_A) \longrightarrow$	Verify MAC
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
Verify MAC	$\longleftarrow t_B, \mathsf{MAC}_{K_M}(ID_B, ID_A, t_B)$	
$\mathbf{Z} = t_B^{r_A}$		$\mathbf{Z} = t_A^{r_B}$

Protocol 5.39: Jeong–Katz–Lee protocol TS3

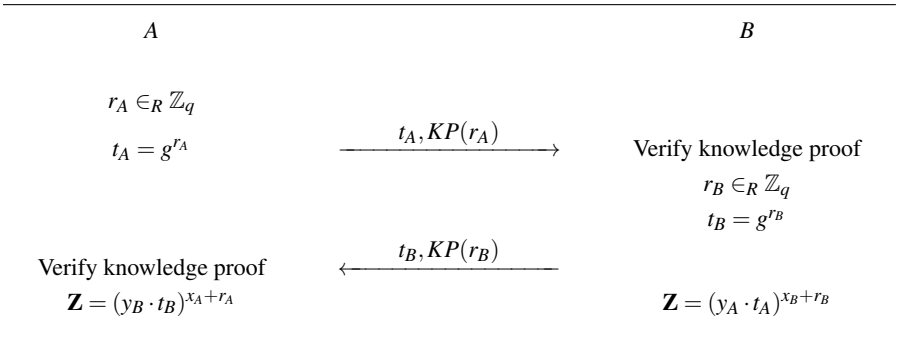
Jeong *et al.* [397] provided a security proof for Protocol 5.39 in a Bellare–Rogaway-style model, including forward secrecy and assuming the difficulty of the decision Diffie–Hellman problem. The protocol does not protect against key compromise impersonation, since an adversary who obtains either one of the long-term keys x_A and x_B can masquerade as either A or B.

A protocol very similar to Protocol 5.39 was derived by Boyd *et al.* [139] by a rather different route (and published, by coincidence, at the same conference). They proposed a generic authenticator based on applying a MAC derived from the static Diffie–Hellman value. When applied to the basic (ephemeral) Diffie–Hellman protocol this yields the same structure as that of Protocol 5.39, but over three rounds instead of one and providing explicit entity authentication.

5.5.12 YAK Protocol

Hao [344, 345] highlighted the general principle of checking the format of received protocol messages and applied this principle to design a protocol known as YAK. As shown in Protocol 5.40, the key computation in YAK is quite similar to that in Protocol 5.11. The shared secret computed by A and B is $\mathbf{Z} = g^{x_A r_B + x_B r_A + r_A r_B + x_A x_B}$,

so that it now includes the static Diffie–Hellman key as one component. There is also a similarity with HMQV; indeed, YAK can be seen as a special variant of HMQV (Protocol 5.14) in which a degenerate hash function is used so that the values d and e in HMQV are both equal to 1.



Protocol 5.40: YAK protocol

A significant difference from Protocols 5.11 and 5.14 is that YAK includes a so-called *knowledge proof* in the exchanged messages. This component should allow the recipient to check that the sender has knowledge of the ephemeral secret key chosen. In Protocol 5.40, the notation $KP(r)$ denotes a knowledge proof of the value r. While the implementation of $KP(r)$ is left flexible in the protocol specification, a concrete suggestion in the paper is to use a Schnorr signature on a message which contains the sender identity and other protocol details. Since Schnorr signatures are designed as non-interactive proofs of knowledge of a discrete logarithm this intuitively provides the requires proof of knowledge of r. This knowledge proof is essential to the security of the YAK protocol. For example, if it were omitted then a key compromise impersonation, very similar to Attack 5.8, would be possible.

While Hao provided a detailed security analysis of the YAK protocol [345], this did not use any of the common computational models or provide a reductionist proof. Because an impostor can generate a valid protocol message while choosing the ephemeral secret key, YAK can only achieve weak forward secrecy.

Toorani [713] pointed out that a malicious principal can force the agreed key to take on a fixed value. Specifically, A could choose $t_A = y_A^{-1}$ and then B computes the shared secret to the value 1. This is a severe failure of the key control property. Toorani also described various attacks on the YAK protocol and explained how to prevent them. Specific measures proposed by Toorani were to include the identities of the principals and the exchanged messages in the key derivation function (omitted in Protocol 5.40) and to check that the static and ephemeral Diffie–Hellman values are of order q.

5.5.13 DIKE Protocol

As its name implies, the Deniable Internet Key Exchange (DIKE) protocol of Yao and Zhao [755] focuses on privacy aspects, particularly strong deniability. Instead of using signatures for authentication, the protocol makes use of proofs of knowledge of the private ephemeral and long-term keys rather like Protocol 5.40. Protocol 5.41 shows the protocol messages.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{\mathsf{sid}, t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$ $KP(r_B) =$ $H(\mathsf{sid}, ID_B, t_B, t_A, t_A{}^{r_B})$
Verify knowledge proof $KP(x_A, r_A) =$ $H(\mathsf{sid}, ID_A, t_A, t_B, t_B{}^{x_A}, t_B{}^{r_A})$	$\xleftarrow{\mathsf{sid}, ID_B, t_B, KP(r_B)}$	
	$\xrightarrow{\mathsf{sid}, ID_A, KP(x_A, r_A)}$	Verify knowledge proof $KP(x_B, r_B) =$ $H(\mathsf{sid}, ID_B, t_B, t_A, t_A{}^{x_B}, t_A{}^{r_B})$
Verify knowledge proof $\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{\mathsf{sid}, KP(x_B, r_B)}$	$\mathbf{Z} = t_A^{r_B}$
	$\mathbf{K} = H'(\mathbf{Z}, t_A, t_B)$	

Protocol 5.41: DIKE protocol

Yao and Zhao provided a security proof in the post-specified peer model of Canetti and Krawczyk [180]. This includes forward secrecy but not KCI resistance; indeed, it is easily checked that knowledge of the long-term key of A is sufficient for an adversary to masquerade as any other party to A. The protocol specification makes use of a session identifier, sid, which must be unique for each session. as required in the CK model. The concrete specification of sid is left open but one suggestion of the designers is to use the concatenation of nonces exchanged prior to the protocol run.

The use of proofs of knowledge is reminiscent of the YAK protocol (Protocol 5.40), but here the proofs are simply hashes of relevant fields, with the hash function H treated as a random oracle. The fields to be hashed are chosen in such a way that either of the two parties is able to construct the proofs in order to aid deniability. Neither party uses its long-term key before knowledge of the peer's ephemeral

key has been checked, guaranteeing that the peer can indeed compute the proofs for either side.

Yao and Zhao [755] also specified an ID-based version of Protocol 5.41. It has a similar structure but uses pairings to compute the static key, replacing the static Diffie–Hellman value used in the knowledge proofs in Protocol 5.41.

5.5.14 Comparison of Authenticated Diffie–Hellman Protocols

Table 5.6 compares some of the major properties of the protocols of this section. Many of these protocols are aimed at practical application on the Internet and provide extended properties such as identity protection and denial-of-service resistance. This makes comparison of the protocols difficult, but explains why these may be of significant interest even if they are less efficient than many of the protocols examined in Sect. 5.4.

Table 5.6: Summary of major properties of key agreement protocols using explicit authentication

Properties →	No. of	Denial-of-service	Resists	Forward	Security
↓ *Protocol*	passes	mitigation	KCI	secrecy	proof
STS (5.24, 5.26)	3	No	No	Yes	No
Oakley aggressive (5.27)	3	No	Yes	Yes	No
Oakley conservative (5.29)	7	Yes	Yes	Yes	No
SKEME (5.30)	3	No	Yes	Yes	Yes
IKE main (5.31)	6	Yes	Yes	Yes	Yes
SIGMA (5.32)	3	Yes	Yes	Yes	CK
IKEv2 (5.33)	4	Yes	Yes	Yes	CK
JFK (5.34)	4	Yes	Yes	Yes	CK
Lim–Lee (5.36, 5.37)	3	No	Yes	No	No
Hirose–Yoshida (5.38)	3	No	No	Yes	No
JKL TS3 (5.39)	2	No	No	Yes	CK
YAK (5.40)	2	No	No	Weak	Custom
DIKE (5.41)	4	No	No	Yes	CK

A feature of many of these protocols is that either a signature is required from each principal, or each principal is required to decrypt some protocol parameters with their private key. For this subclass of protocols, key compromise impersonation will always be resisted. Protocols which use a MAC, which can be computed by either party, or a knowledge proof typically do not achieve KCI resistance.

We have included mitigation of denial-of-service attacks in Table 5.6 since this is a common feature sought for protocols used on the Internet, several of which are listed. However, it should be noted that the mechanisms employed for denial-of-service resistance, such as cookies for reachability and puzzles for proofs-of-work, are typically independent of the other protocol messages and can be added generically, even if they may require additional messages.

Most of the protocols in Table 5.6 achieve full forward secrecy, either by the use of signatures or by providing key confirmation using more than one round. An exception is YAK, which is a one-round protocol without explicit authentication. Several protocols fail to achieve resistance to KCI, in particular when authentication is achieved using a MAC keyed from the static Diffie–Hellman shared secret. This can be viewed as a trade-off between security and efficiency.

As is typical today, the more recent protocols come with a security proof in some computational security model. It is striking that most of the protocols in this category do not provide protection against leakage of ephemeral secrets, which prevents them being secure in eCK-type models. This also applies to the most popular Internet protocol, TLS.

5.6 Protocols in ISO/IEC 11770-3

The international standard ISO/IEC 11770-3 [383] is devoted to key management techniques using public key (asymmetric) techniques. The key transport protocols in this standard were discussed in Chap. 4. There are 12 key agreement mechanisms in the standard; these may be regarded as frameworks for detailed protocols into which fall many of the protocols we have examined. All the key agreement protocols in the standard, except for Mechanisms 11 and 12, are based on the Diffie–Hellman exchange.

Cremers and Horvat [229] analysed all of the key agreement protocols in the previous, 2008, version of the ISO/IEC 11770-3 standard, using the Scyther tool. Both versions of the standard contain very similar protocols, except that Key Agreement Mechanism 12 is not included in the earlier version. Cremers and Horvat confirmed the properties claimed in the standards for all the protocols with the exception of Key Agreement Mechanism 11, which we discuss below.

Key Agreement Mechanisms 1, 8 and 12: These are non-interactive key exchange protocols. A typical concrete example of Mechanism 1 is the static Diffie–Hellman protocol, although other NIKE protocols also fit the standard. Mechanism 8 is differentiated by requiring protocol messages to be elliptic curve points. Mechanism 12 is an abstract version of the Joux protocol (see Sect. 9.2.7).

Key Agreement Mechanisms 2 and 3: These have one message exchange. Mechanism 2 involves a random input from A to form an ElGamal encryption key. Mechanism 3 adds authentication information from A. The Nyberg–Rueppel protocol (Protocol 5.4) fits into this category.

Key Agreement Mechanisms 4 and 5: These are two message exchanges, in which both principals send an ephemeral public key. Mechanism 4 is simply ephemeral Diffie–Hellman. Mechanism 5 uses the long-term and ephemeral keys in the calculation of the shared secret. The MTI protocols (see Sect. 5.3) and KEA (Protocol 5.9) fit into Mechanism 5.

Key Agreement Mechanism 6: This unusual mechanism is related to Protocol 4.13 examined in Chap. 4, but instead of transporting a generated key, the signature used to respond to a challenge is used as the session key. Without any special properties of the signature, this is a questionable approach. The protocol cannot provide forward secrecy. The standard claims that an example of this mechanism is the Beller–Yacobi protocol [83], but it is difficult to recognise any close similarity.

Key Agreement Mechanism 7: This three-message protocol is the same as Protocol 5.25, with the exception that a MAC is added to the last two messages to provide explicit key confirmation in the manner described in Sect. 5.4.13.

Key Agreement Mechanisms 9 and 10: Mechanism 9, with two messages, is an abstract version of the MQV protocol (Protocol 5.13) or its many variants, including HMQV. The standard requires this to take place in elliptic curve groups and includes cofactor multiplication. Mechanism 10 is a three-pass version of the same protocol, adding MACs to the second and third messages.

Key Agreement Mechanism 11: This four-message protocol is an abstract version of the TLS handshake protocol described in detail in Chap. 6. However, the standard fits only with the RSA version of the handshake protocol, not the Diffie–Hellman version, since it uses encryption. Cremers and Horvat [229] pointed out that the protocol provides only unilateral authentication; indeed a long-term key for authentication is specified only for the party corresponding to the server in TLS. They also pointed out that the protocol does not provide forward secrecy.

Cremers and Horvat were analysing the second edition of the ISO/IEC 11770-3 standard in which it was claimed that mutual authentication is provided as well as forward secrecy. The claim regarding mutual authentication has been removed in the third edition of the standard, but, curiously, the claim that the protocol provides forward secrecy remains. This is well known not to be the case, and is perhaps the main reason that this encryption-based protocol is no longer widely used in TLS and has been removed in TLS version 1.3.

5.7 Diffie–Hellman Key Agreement in Other Groups

Diffie–Hellman key agreement was originally proposed in the algebraic setting of the multiplicative group $\mathbb{Z}_p^*$ and we have used this setting in all our descriptions so far. It has long been known that the basic structure can be generalised to any commutative group. In this section we mention some of the most prominent alternative groups that have been proposed.

Elliptic curve groups have significant potential advantages over using $\mathbb{Z}_p^*$, because of their greater efficiency and compact representation. Many recent protocols have been specially designed with elliptic curve implementation in mind rather than using prime fields. Examples include the MQV (Protocol 5.13) and Oakley (Protocol 5.27) protocols. The Oakley specification includes a number of candidate elliptic curves and also provides for negotiation of new curves during the protocol. It is sometimes possible to avoid certain attacks because of the structure of the curve used. For example, elliptic curve groups can be chosen to have prime order so that there is no need to check whether elements are in a particular subgroup. There are now a variety of standardised elliptic curve groups available to protocol designers.

Hyperelliptic curves are generalisations of elliptic curves. The *Jacobian group* of a hyperelliptic curve is another setting that has been suggested for the Diffie–Hellman technique [437]. It seems that there are many open questions with respect to the security and best implementations in this setting. Smart and Siksek [681] have proposed the use of a different structure on hyperelliptic curves.

Extension fields have been used by several authors. For example, Scheidler *et al.* [655] gave details of Diffie–Hellman in real quadratic fields. Brouwer *et al.* [158] designed a Diffie–Hellman version using a subgroup of the extension field $GF(p^6)$ which has attractive performance properties. This is related to Lenstra and Verheul's XTR group [481] which can also be used for Diffie–Hellman.

$\mathbb{Z}_n^*$. McCurley [532] designed a special version of Diffie–Hellman key agreement in $\mathbb{Z}_n^*$ when n is the product of two primes, i.e. $n = pq$. He showed that breaking this version is equivalent to both factorising n and breaking Diffie–Hellman in the two subgroups $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$. Scott [659] extended Protocol 5.21 to this setting to obtain an identity-based key agreement protocol for which it was proven that obtaining the shared secret from the messages exchanged (and any other public information) is equivalent to factorising the modulus.

Isogenies between elliptic curves can be used to provide Diffie–Hellman analogues [395, 698]. One promising aspect of this idea is that such variants may be immune to attacks from quantum computers.

5.8 Protocols Based on Encryption or Encapsulation

Many key agreement protocols exist which do not use Diffie–Hellman but instead rely on encryption. One reason for this can be to achieve a computational advantage, but simple protocols in this class lack forward secrecy. More recently, key encapsulation mechanisms (KEMs) (see Definition 8) have been applied in place of encryption, with the specific aim of achieving security proofs without requiring an assumption of random oracles. Note that protocols in this section still provide key agreement since both parties have an input to the session key. This differs from the protocols in Chap. 4, where the session key is generally chosen directly by one party. There are, however, some cases which are not so clear; for example, Protocol 4.17 can also be regarded as a key agreement protocol.

5.8.1 SKEME without Forward Secrecy

The SKEME protocol [450] described in Sect. 5.5.4 has versions with and without forward secrecy. Protocol 5.42 shows the mode without forward secrecy; it replaces the Diffie–Hellman exchange in Protocol 5.30 with a simple exchange of nonces chosen by A and B.

Shared information: Security parameter L.

A		B
$N_A \in_R [0, 2^L]$ $r_A \in_R \mathbb{Z}_q$	$\xrightarrow{\mathsf{Enc}_B(ID_A, N_A), r_A}$	$N_B \in_R [0, 2^L]$ $r_B \in_R \mathbb{Z}_q$
Verify MAC $K_0 = H(N_A, N_B)$	$\xleftarrow{\mathsf{Enc}_A(N_B), r_B, \mathsf{MAC}_{K_0}(r_A, r_B, ID_B, ID_A)}$ $\xrightarrow{\mathsf{MAC}_{K_0}(r_B, r_A, ID_A, ID_B)}$	$K_0 = H(N_A, N_B)$ Verify MAC

$$\mathbf{K} = \mathsf{MAC}_{K_0}(\mathsf{MAC}_{K_0}(r_B, r_A, ID_A, ID_B))$$

Protocol 5.42: SKEME protocol without forward secrecy

The temporary shared secret K_0 is calculated as $K_0 = H(N_A, N_B)$. The session key is calculated by applying the MAC function to the final message:

$$\mathbf{K} = \mathsf{MAC}_{K_0}(\mathsf{MAC}_{K_0}(r_B, r_A, ID_A, ID_B)).$$

Evidently N_A and N_B, and consequently $\mathbf{K}$, are revealed if the long-term decryption keys of A and B become known, so forward secrecy is not obtained. However, the use of public key encryption prevents key compromise impersonation.

A protocol very similar to Protocol 5.42 was standardised by NIST as KAS2-bilateral-confirmation [578], where RSA is used as the encryption algorithm. The difference is that the NIST protocol uses the ciphertext values as nonces in the MAC, instead of using separate r_A and r_B values. Chatterjee *et al.* [189] provided a formal security proof for KAS2-bilateral-confirmation in an eCK-style model, excluding forward secrecy. Their analysis applies to a more generic version of the protocol with any trapdoor one-way function in place of RSA.

The SKEME rekeying protocol [450] is identical to Protocol 5.42 with the encrypted values in messages 1 and 2 omitted. Instead, the previous K_0 value is reused to find a new $\mathbf{K}$ value based on the newly chosen nonces. There is also a version of the Oakley protocol [596] without Diffie–Hellman, whose structure is very similar to Protocol 5.42.

5.8.2 Boyd–Cliff–González-Nieto–Paterson Protocol

The application of encryption instead of Diffie–Hellman was, as in SKEME, used in a later protocol of Boyd *et al.* [134] with a number of optimisations. In order to achieve a more efficient protocol, *key encapsulation* was used instead of encryption, since the former inherently generates a random value suitable as an input to session key derivation (see Definition 8). In Protocol 5.43, we use the notation $\mathsf{Encap}_U(\cdot)$ to denote encapsulation of a key for a principal U using the public key of U. Encapsulation is a randomised algorithm, but we do not explicitly show the random input. Similarly, $\mathsf{Decap}_U(C)$ denotes decapsulation by principal U of the key encapsulated in C.

A		B
$(C_A, K_A) = \mathsf{Encap}_B(\cdot)$		
	$\xrightarrow{C_A}$	$K_A = \mathsf{Decap}_B(C_A)$
		$(C_B, K_B) = \mathsf{Encap}_A(\cdot)$
$K_A = \mathsf{Decap}_A(C_B)$	$\xleftarrow{C_B}$	
	$\mathbf{Z} = K_A, K_B$	

Protocol 5.43: Protocol of Boyd, Cliff, Gonzaláz-Nieto and Paterson

A stated goal of the protocol design was to achieve a one-round protocol and to be secure without relying on random oracles. In order to achieve a proof in the standard model, Boyd *et al.* [134] employed a randomness extractor [455] rather than applying a hash function to the shared secret $\mathbf{Z}$. They proved security of the protocol in the Canetti–Krawczyk model (without forward secrecy) as long as the key encapsulation mechanism used provided CCA security and that the randomness extractor satisfied a standard pseudo-randomness condition.

Protocol 5.43 is a very simple protocol which achieves basic security goals including resistance to KCI attacks. Essentially the same protocol was standardised by NIST as their KAS2-basic scheme [578] but with the generic encapsulation scheme replaced by RSA encryption of randomly chosen values. In order to include weak forward secrecy, Boyd *et al.* [134] also analysed the same protocol with an independent Diffie–Hellman exchange added in parallel to the protocol messages.

While we have shown Protocol 5.43 in a public key setting (both A and B are assumed to have public keys used in the encapsulation process), Boyd *et al.* [134] included the identity-based setting in their definitions. They also provided detailed efficiency estimates in the identity-based case.

5.8.3 Fujioka–Suzuki–Xagawa–Yoneyama Protocol

Fujioka *et al.* [286] (FSXY) refined and generalised the construction of Protocol 5.43. Their aim was to achieve generic protocols which were secure in strong security models without relying on random oracles. Furthermore, by relying only on KEMs, even to achieve forward secrecy, they were able to instantiate their generic construction with concrete KEMs based on a variety of computational problems.

Protocol 5.44 shows the generic FSXY protocol which makes use of two KEMs and also splits the long-term key of each party into two parts. The first KEM takes the same role as in Protocol 5.43 while the second KEM, which we denote eKEM, is used to achieve (weak) forward secrecy and uses an *ephemeral* public key generated as part of the protocol. We denote the temporary, or ephemeral, public key for eKEM by eK_T and encapsulation using this public key by $\mathsf{eEncap}_T(\cdot)$, with an unspecified random input. The shared secret consists of three parts, K_A, K_B, and K_T, which are combined in a specific key derivation process not shown in Protocol 5.44.

Information known to A: Long-term private key (x_A, x'_A)
Information known to B: Long-term private key (x_B, x'_B)
Shared information: PRF families F, F', two KEMs $(\mathsf{Encap}, \mathsf{Decap})$ and $(\mathsf{eEncap}, \mathsf{eDecap})$

A		B
Generate random r_A, r'_A $(C_A, K_A) =$ $\mathsf{Encap}_B(F_{x_A}(r_A) \oplus F'_{r'_A}(x'_A))$ Generate temporary key eK_T for eKEM		
	$\xrightarrow{C_A, eK_T}$	$K_A = \mathsf{Decap}_B(C_A)$ Generate random r_B, r'_B $(C_B, K_B) =$ $\mathsf{Encap}_A(F_{x_B}(r_B) \oplus F'_{r'_B}(x'_B))$ $(C_T, K_T) = \mathsf{eEncap}_T(\cdot)$
$K_A = \mathsf{Decap}_A(C_B)$ $K_T = \mathsf{eDecap}_T(C_T)$	$\xleftarrow{C_B, C_T}$	
$\mathbf{Z} = K_A, K_B, K_T$		

Protocol 5.44: Protocol of Fujioka, Suzuki, Xagawa and Yoneyama

Fujioka *et al.* [286] proved security of Protocol 5.44 in the HMQV model (see Sect. 2.3.3), which they called the CK+ model. This captures attacks very similar to the eCK model but deals also with leakage of session state; what consitutes session state needs to be defined in the protocol specification. The proofs require that the

KEM is CCA secure and eKEM is CPA secure. The randomness for the KEM is constructed using the *twisted PRF* trick so that the ephemeral and long-term key components appear both as keys and as inputs to the PRFs F and F'. The PRF output is used as the random input to the KEM, and it looks random to an adversary who does not obtain both the long-term and the ephemeral secret keys of the party. This idea is a replacement in the standard model for the NAXOS trick used in the random oracle model (see Sect. 5.4.7).

As mentioned above, weak forward secrecy is achieved through online generation of an ephemeral key pair, and the public key eK_T is sent by A in the first message. In the second message B returns the encapsulation of K_T and this part of the shared secret cannot be recovered once the ephemeral keys have been deleted. This technique can be seen as a generalisation of Diffie–Hellman key exchange, since an ephemeral Diffie–Hellman key can be interpreted as both a public encapsulation key and an encapsulation of the shared Diffie–Hellman key.

Concrete instantiations of Protocol 5.44 can be achieved by choosing any suitable KEMs. The two KEMs could even be the same, although eKEM requires only CPA security. Fujioka *et al.* [286] suggested instantiations based on KEMs relying on integer factorisation, code-based problems and lattice-based problems. A specific advantage of the latter two options is that they can remain secure in the face of quantum computation. However, the need to transmit a new public key during the protocol reduces the practical attractiveness of these instantiations when large public keys are required. Fujioka *et al.* [286] also described a generic ID-based protocol.

A disadvantage of Protocol 5.44 compared with Protocol 5.43 is that the former is not actually a one-round protocol, because it is necessary for B to obtain the newly generated ephemeral key from the first message before completing the returned message. Therefore the two messages cannot be sent simultaneously. Yoneyama [766] proposed a modification of the FSXY protocol which overcomes this limitation, but it relies on the existence of a so-called *KEM with public-key-independent ciphertext* for which efficient constructions are unknown.

Yang [751] designed a one-round protocol related to Protocol 5.44 which avoids the twisted PRF trick. It derives four subkeys. Two of them are from the encapsulation process, like K_A and K_B in Protocol 5.44. The other two come from a passive-secure key exchange (such as basic ephemeral Diffie–Hellman) and from a non-intereractive key exchange. The four keys are combined into the session key. The protocol is secure in the eCK model and the standard model.

5.8.4 Alawatugoda Protocol

Alawatugoda [26] proposed Protocol 5.45 in order to achieve a simple protocol without using the random oracle model. The protocol combines the ephemeral and the static Diffie–Hellman values, just as in the Unified Model (Protocol 5.12). However, Protocol 5.45 also applies encryption to the exchanged ephemeral Diffie–Hellman values.

Alawatugoda showed that Protocol 5.45 is secure in the eCK model for any encryption function which achieves CCA security. Of course, the protocol is not very

Shared information: Static Diffie–Hellman key $S_{AB} = g^{x_A x_B}$. PRF family $F_K(\cdot)$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{E_B(t_A)}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{r_A}, S_{AB}$	$\xleftarrow{E_A(t_B)}$	$\mathbf{Z} = t_A^{r_B}, S_{AB}$

$$Z_1 = g^{r_A r_B}; Z_2 = S_{AB}; T = ID_A, E_B(t_A), ID_B, E_A(t_B)$$
$$\mathbf{K} = F_{Z_1}(T) \oplus F_{Z_2}(T)$$

Protocol 5.45: Alawatugoda key agreement protocol

efficient, since it adds one encryption and one decryption to the three exponentiations required from each principal. Moreover, only weak forward secrecy is achieved, since an active adversary can choose an ephemeral input, send the correct message, and later recover the session key once the long-term decryption keys become known. However, the proof is in the standard model and avoids the NAXOS trick.

5.9 Conclusion

The large number of key agreement protocols that we have examined in this chapter gives some indication of their importance, both in the research literature and in practical applications. Protocols based on Diffie–Hellman key exchange remain the most common, but come in various different types. In particular, we distinguished between those in which the exchanged information consists only of the Diffie–Hellman messages, and those which exchange additional authenticating information. In the first category, the publication of the HMQV protocol generated a lot of research and related constructions. The second category has also seen significant development, with a focus on Internet applications (see also Chap. 6 for key agreement in TLS). Alternative constructions based on KEMs have received increased attention.

One of the major changes in the past 10 years has been the emphasis on stronger security models, particularly eCK-type models incorporating security against leakage of ephemeral keys. This can even be taken a step further by considering partial leakage of keys [27]. In the near future we can expect to see an increasing interest in key agreement protocols designed to remain secure against quantum computers.

6

Transport Layer Security Protocol

6.1 Internet Security Protocols

Authenticated key exchange protocols are at the core of Internet security protocols: they authenticate one or more of the parties communicating, and provide the establishment of a session key that is then used to encrypt application data. There are several protocols in widespread use to secure various applications. The most prominent are the following:

Transport Layer Security (TLS). Formerly known as the *Secure Sockets Layer* (SSL) protocol. The TLS standards are developed and maintained by the Internet Engineering Task Force (IETF) TLS working group. TLS operates over the TCP/IP protocol stack and is used to protect web traffic (using HTTPS), file transfers, email transport, and many other applications. To date there have been two versions of SSL (SSL v2 and SSL v3) and three versions of TLS (TLS 1.0, TLS 1.1, and TLS 1.2); the next version, TLS 1.3, was submitted for standardisation in March 2018. A variant called *Datagram TLS* (DTLS) is used for protection of datagrams transmitted over UDP. See Table 6.1 for a list of versions of TLS and related protocols.

Secure Shell (SSH). SSH operates over the TCP/IP protocol stack and is primarily for used for securing remote command-line logins, replacing the insecure Telnet protocol. SSH can also be used for file transfer (secure copy (scp)), as well as a rudimentary virtual private network. The current version, SSH v2, is incompatible with the prior SSH v1.

Internet Protocol Security (IPsec). The IPsec protocol suite operates at the IP layer of the IETF protocol stack, and is automatically applied to all application data above it. It can be used in two modes: *transport mode* authenticates and encrypts the contents of an IP packet, but leaves the headers unchanged; and *tunnel mode* authenticates and encrypts an entire IP packet, including the headers, and then encapsulates that as the payload of a new IP packet. IPsec is typically run in one of three architectures: host-to-host (directly securing a connection between two computers), host-to-gateway (for example, a remote user

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_6

Table 6.1: Versions of SSL/TLS and Datagram TLS (DTLS)

SSL v2	1995	[355]
SSL v3	1996	[283]
TLS 1.0	1999	[249]
TLS 1.1	2006	[250]
TLS 1.2	2008	[251]
DTLS 1.0	2006	[627]
DTLS 1.2	2012	[628]

connecting to a corporate network via a virtual private network), and gateway-to-gateway (for example, a gateway connecting all computers in a branch office to the head office).

There are several other special-purpose Internet protocols that make use of authenticated key exchange, including the Tor anonymity network and secure instant messaging protocols such as Off-the-Record (OTR) messaging, and the Axolotl ratchet/Signal protocol. Appendix A summarizes many of these.

In this chapter, we examine the Transport Layer Security protocol in detail. TLS is interesting owing to its widespread use, the complexities in its design compared with academic key exchange protocols, and the many weaknesses and flaws found in the protocol and its implementations. TLS also differs from academic key exchange protocols in the sense that it directly combines key exchange and authenticated encryption to establish a secure channel.

6.2 Background on TLS

The Secure Sockets Layer protocol was developed by Netscape in the mid-1990s. The first publicly released protocol was SSL v2 in February 1995 [355]. A redesign, aiming to fix several security flaws, was published in November 1996 (and published as a historical RFC in August 2011 [283]). In January 1999, the Internet Engineering Task Force published the TLS 1.0 protocol [249], which made minor changes to SSL v3. In April 2006, the IETF published TLS 1.1 [250], primarily making changes in how the CBC block cipher encryption mode worked. TLS 1.2 was published in August 2008 [251], incorporating changes to the PRF, support for authenticated encryption with additional data, and more precise negotiation of algorithms. There are more than 45 additional RFCs that describe additional behaviour or functionality; notable ones include the specification of elliptic curve cryptography [106] and pre-shared keys [269], the addition of new ciphers such as AES [213], extensions to the protocol specification [112], and deprecation of old algorithms [55, 619].

TLS is used to protect many applications. It is most familiar to many when it is used to protect web traffic transmitted over the Hypertext Transport Protocol (HTTP). In this context, called HTTPS, an SSL/TLS connection is established (typically on TCP port 443, different from port 80 for the unsecured website), and then

HTTP data is transmitted across the secured connection. TLS can also be used to protect email transport protocols (IMAP and POP, for clients to download messages from mail servers, and SMTP, for delivery of outgoing messages), as well as file transfer (FTP). In these contexts, the unsecured connection is 'upgraded' to a secure connection: first, the normal unsecured connection is established, and then a special command (such as 'STARTTLS') is used to activate TLS, at which point a TLS connection will be established, and all subsequent data will be transmitted over the TLS connection.

6.3 Protocol Structure

The TLS protocol consists of several subprotocols; for cryptographic purposes, the two most important subprotocols are the *handshake protocol* and the *record layer protocol*. In the handshake protocol, the client and server agree on a set of cryptographic parameters, called a *ciphersuite*, exchange authentication credentials, establish a shared secret, perform explicit authentication, and derive keys for bulk encryption and message authentication. The record layer protocol provides delivery of all messages in TLS, including handshake protocol messages and application data, but in particular the record layer protocol optionally protects messages using authentication and encryption. There is an additional *alert protocol*, which is used to notify peers about errors or to close the connection. The stacking of these layers is shown in Fig. 6.1. The exact cryptographic algorithms used in both the handshake and the record layer protocol depend on which ciphersuite the parties have negotiated.

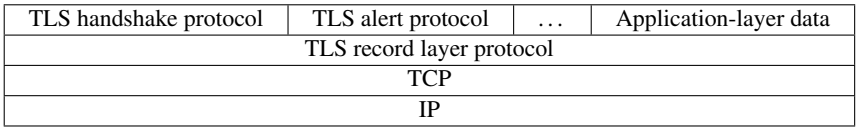


Fig. 6.1: Layering of TLS subprotocols and the TCP/IP stack

TLS can provide entity authentication using long-term public keys using the X.509 public key infrastructure [223]. Authentication can be either mutual or only server-to-client. Parties generate long-term secret key/public key pairs, then submit their public key and a proof of possession (usually a signature using the key) via a certificate-signing request to certification authority (CA). The CA verifies the certificate-signing request and the identity of the requesting party, then issues a certificate containing the party's identity (in the case of servers, this is the server's fully qualified domain name), and public key, as well as additional fields such as the period of validity and the purposes for which the certificate can be used; the certificate is signed by the CA using its long-term key. Web browsers typically have 150 or more certificates for 80 or more commercial and governmental root CAs installed by default; however, many root CAs allow independent subordinate CAs to also issue

certificates, so the exact number of certificate issuers that are trusted by default by browsers is unknown even to browser vendors. Taking into account these subordinate issuers, over 650 certificate issuers trusted by default by major browsers have been observed in Internet-wide surveys of TLS server certificates [266].

6.3.1 Handshake Protocol

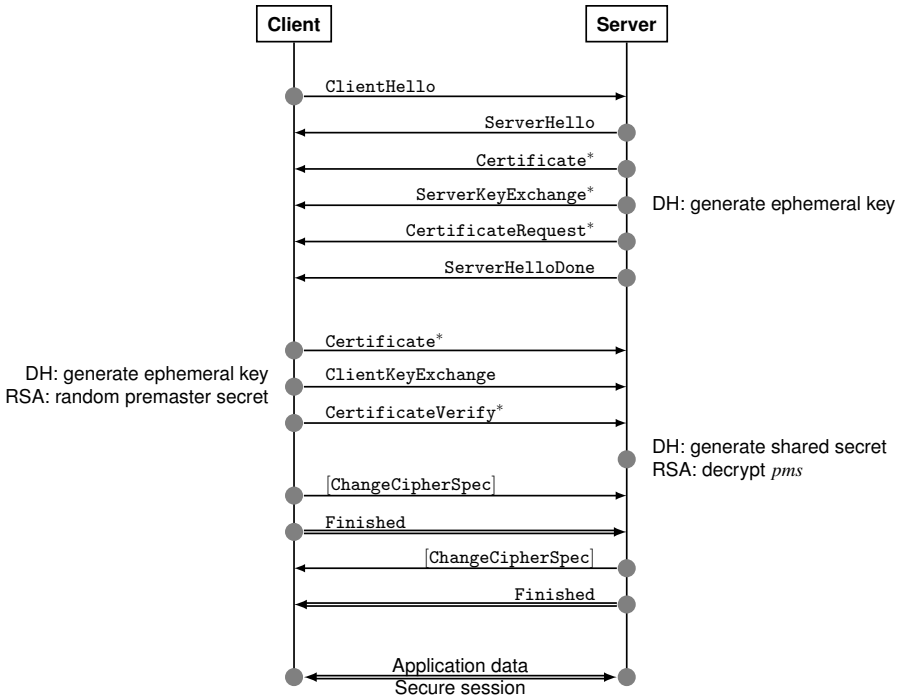
To first establish a TLS connection, a client and a server run the full TLS handshake protocol. As shown in Protocol 6.1, the TLS handshake protocol proceeds as follows.

1. The client initiates the connection by sending a `ClientHello` message, which contains a nonce as well as the client's list of preferred ciphersuites.
2. The server responds with several packets together:
   - `ServerHello` message, containing a nonce and the chosen ciphersuite;
   - `Certificate` message, if server authentication is being used;
   - `ServerKeyExchange` message containing the server's ephemeral Diffie–Hellman value, if the ciphersuite calls for it;
   - `CertificateRequest` message, if the server is asking for client authentication.
3. The client verifies the server's certificate, then responds with:
   - `Certificate` message, if client authentication is being used;
   - `ClientKeyExchange` message, which the parties use to compute the session key;
   - `CertificateVerify` message, if client authentication is being used.

   At this point, the client computes a *premaster secret*, which is the raw shared secret. Using the premaster secret and the client and server nonces, the client then computes the *master* secret, from which it derives four record-layer session keys: two keys for bulk encryption (client-to-server and server-to-client), and two keys for message authentication (client-to-server and server-to-client).
4. The client sends a `ChangeCipherSpec` message, which indicates that all further messages will be sent encrypted using the record layer protocol. Finally, the client sends (encrypted by the record layer) a `Finished` message, containing a key confirmation value.
5. The server computes the premaster secret and master secret, then derives the session keys. It verifies the client authentication, if any, then verifies the `Finished` message it has received from the client.
6. The server sends a `ChangeCipherSpec` message, which similarly indicates that all further messages will be sent encrypted using the record layer protocol. Finally, the server sends (encrypted by the record layer) its own `Finished` message for key confirmation.

This concludes the handshake protocol, and application data can now be sent using the record layer protocol.

If the client and server have previously established a session, they can perform an abbreviated handshake for *session resumption*. In this case, the message flow is as



* denotes messages that may not be present in all ciphersuites.
[...] denotes messages that are sent over the TLS alert protocol.
Single arrows denote plaintext flows; double arrows denote encrypted flows.

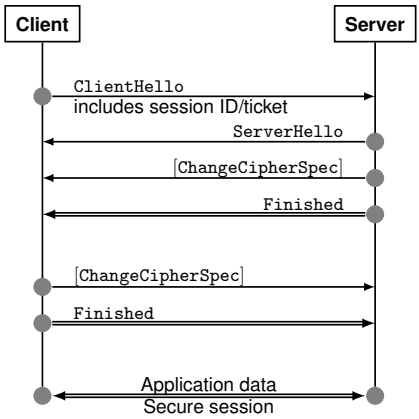
Protocol 6.1: TLS $\leq$ 1.2 handshake protocol – full handshake

shown in Protocol 6.2. The abbreviated handshake allows re-establishment of a TLS record layer in a single round trip, rather than two round trips as in the full handshake protocol; it also avoids some expensive public key operations.

We now examine each message of the handshake protocol in detail.

`ClientHello`. With this message, the client initiates the connection. The primary purpose of the message is to convey the client's parameter preferences. The message includes:

- a protocol version field, indicating the maximum supported version of SSL or TLS;
- the `client_random` value, a 32-byte nonce;
- the `session_id` of a previous session, if session resumption is being used;
- a list of the client's supported cryptographic parameters in order of preference;
- a list of the client's supported compression methods in order of preference; and



Protocol 6.2: TLS $\leq$ 1.2 handshake protocol – abbreviated handshake

- any optional extensions, indicating additional information.

TLS 1.0 and TLS 1.1 were updated to support extensions. Frequently used extensions include the server name indication extension, which helps a TLS server hosting multiple domains to pick the appropriate certificate to send to the client; the elliptic curves extension, indicating which elliptic curves the client supports; and the renegotiation indication extension, which help protects against the renegotiation attack (see Sect. 6.11.2).

`ServerHello`. With this message, the server indicates which parameters have been chosen. The message includes:

- a protocol version field, indicating which protocol version will be used for this connection;
- the `server_random` value, a 32-byte nonce;
- the `session_id` of this session, for the purposes of future session resumption;
- the chosen ciphersuite;
- the chosen compression method; and
- any optional extensions.

The `client_random` and `server_random` nonces serve to provide freshness or liveness guarantees and prevent replay attacks.

`Certificate` (server). For all server-authenticated ciphersuites which involve certificates, the server also sends a message containing its certificate. This message is structured as a chain of X.509 certificates: the first certificate is the server's certificate, and each subsequent certificate in the chain is the certificate of the certification authority that signed the preceding certificate. The last (root) certificate may be omitted, as the client must have that certificate installed already in order to trust it. Upon receiving this message, the client validates the certificate chain by (a) checking that the subject of the server's certificate matches the domain name of the server with which it is communicating, (b) verifying the sig-

nature of each certificate under the public key of the next certificate in the chain, and (c) checking the validity period of the certificate. Optionally, the client may check the revocation status of the certificate using either *certificate revocation lists* (CRLs) or the *online certificate status protocol* (OCSP). Servers can include a recent OCSP response in the handshake in a process known as *OCSP stapling*.

`ServerKeyExchange`. This message is sent if the chosen ciphersuite calls for the server to send an ephemeral public key, which is the case for ephemeral Diffie–Hellman ciphersuites, but not for static Diffie–Hellman or RSA key transport ciphersuites. For signed-Diffie–Hellman ciphersuites, the structure of this message is:

- the server Diffie–Hellman parameters and ephemeral public key:
  - for finite-field Diffie–Hellman, this contains the prime modulus p, the generator g, and the server's ephemeral public key $Y \equiv g^y \mod p$;
  - for elliptic curve Diffie–Hellman, this contains the chosen elliptic curve, either as a named curve or as the explicit parameters (field, curve, generator, order, cofactor), and the server's ephemeral public key $Y = yP$; and
- the server's signature over the `client_random`, `server_random`, and Diffie–Hellman parameters.

In SSL v3, the `ServerKeyExchange` message would be sent for RSA key transport ciphersuites if the server's long-term key was solely a signature key. In TLS 1.0, this was removed, except for export ciphersuites where the server's long-term key was not sufficiently long, and as of TLS 1.1 was removed entirely.

`CertificateRequest`. The server sends this message if it requires the client to authenticate itself using a certificate. The message includes:

- a list of supported client certificate types;
- a list of supported signature types supported for certificate verification; and
- a list of acceptable certificate authorities.

`Certificate` (client). If the server has sent a `CertificateRequest` message, then the client responds with its certificate; this message has the same structure as the server's `Certificate` message. Upon receipt of this message, the server verifies the validity of the client's certificate as above.

`ClientKeyExchange`. In the full handshake, this message is always sent by the client to establish a premaster secret. The structure of the message depends on the ciphersuite chosen.

- For RSA-key-transport-based ciphersuites, this message contains the encrypted premaster secret. In particular, the client chooses a random 46-byte value, and, together with the two bytes of `client_version`, these 48 bytes comprise the *premaster secret*. The client encrypts the premaster secret under the server's RSA public key (from the server's certificate) using PKCS#1v1.5 encryption. These ciphersuites do not provide forward secrecy.
- For finite-field or elliptic curve ephemeral Diffie–Hellman ciphersuites, this message contains the client's ephemeral public key. In this case the premaster secret is the Diffie–Hellman shared secret. These ciphersuites provide forward secrecy.

- For static Diffie–Hellman ciphersuites, this message is empty.

Upon receipt of this message, the server derives the premaster secret. In the case of RSA key transport, the server must carefully implement the PKCS#1v1.5 decryption process to avoid a side-channel leak (see Sect. 6.9.1). Both parties derive the master secret from the premaster secret as specified in the subsequent subsection. Finally, encryption and authentication keys are derived from the master secret.

`CertificateVerify`. This message is sent if the client has sent a certificate. The message is computed as the client's RSA, DSA, or ECDSA signature on all handshake messages it has sent and received up to, but not including, this message. Upon receipt of this message, the server verifies the signature.

`ChangeCipherSpec` (client). The client sends this single-byte message to the server, indicating that all future messages it sends will be encrypted and authenticated. Note that, for networking reasons, this message is technically not part of the handshake protocol but a separate subprotocol.

`Finished` (client). This message is sent by the client to verify that the key exchange and entity authentication were successful. Since it includes authentication of the handshake messages, from a cryptographic perspective the `Finished` message allows the other party to verify that its peer had the same view of the handshake and that no downgrade attacks occurred. The message contains

$$PRF\,(ms, label \| H(handshake))\,,$$

where $label =$ "client finished" and *handshake* is the transcript of all handshake messages sent and received by the party; H is the hash function specified by the ciphersuite. Upon receipt of this message, the server compares the received value with its own computed value. If the values match, the server *accepts*.

`ChangeCipherSpec` (server). The server sends this single-byte message to the client indicating that all future messages it sends will be encrypted and authenticated.

`Finished` (server). The server sends a `Finished` message similar to the client's one above, but computed with $label =$ "server finished". Upon receipt of this message, the client compares the received value with its own computed value. If the values match, the client *accepts*.

Cryptographic Computations in the Handshake Protocol

PRF. In TLS 1.0 and 1.1, the pseudo-random function *PRF* is computed as

$$PRF(secret, label, seed) = P_{\text{MD5}}(S_1, label \| seed) \oplus P_{\text{SHA-1}}(S_2, label \| seed),$$

where S_1 and S_2 are the first and last $|secret|/2$ bytes of *secret* (possibly with a shared middle byte), and

$$\begin{aligned} P_H(secret, seed) = {} & \text{HMAC}_H(secret, A(1) \| seed) \\ & \| \ \text{HMAC}_H(secret, A(2) \| seed) \ \| \ \ldots \end{aligned}$$

for $A(0) = seed$, $A(i) = \text{HMAC}_H(secret, A(i-1))$. In TLS 1.2, the PRF is computed as

$$PRF(secret, label, seed) = P_H(secret, label \| seed)$$

where H is a hash function defined by the ciphersuite; for most ciphersuites defined by TLS 1.2, H is SHA-256.

Master secret. In SSL v3, the 48-byte master secret *ms* is computed from the premaster secret *pms* as

$$ms \leftarrow f(\text{“A”}) \| f(\text{“BB”}) \| f(\text{“CCC”}),$$

where

$$f(\ell) \leftarrow \text{MD5}(pms \| \text{SHA-1}(\ell \| pms \| \texttt{client\_random} \| \texttt{server\_random})).$$

In TLS 1.0 and onward, the 48-byte master secret *ms* is computed from the premaster secret *pms* as

$$ms \leftarrow PRF(pms, \text{“master secret”}, \texttt{client\_random} \| \texttt{server\_random}),$$

where *PRF* is defined as above.

Encryption and authentication keys. In TLS 1.0 and higher, the parties derive encryption and authentication keys from the master secret as follows. First, they compute a sufficiently long value

$$k \leftarrow PRF(ms, \text{“key expansion”} \| \texttt{server\_random} \| \texttt{client\_random})$$

and then partition k into the client-to-server MAC write key, the server-to-client MAC write key, the client-to-server encryption key, the server-to-client encryption key, and, if required, the client-to-server and server-to-client encryption initialisation vectors (IVs). Note that keys are derived slightly different in SSL v3, and also for weakened export ciphersuites.

6.3.2 Record Layer Protocol

A plaintext packet in the record layer protocol consists of:

- a *content type*, which specifies what type of data the payload contains; this may be a handshake message, a `ChangeCipherSpec` message, an alert (error) message, or application data;
- the *version* of TLS used;
- the *length* of the payload, not more than 2^{14};
- the payload data.

Once the plaintext packet is assembled, its payload is *compressed* using whichever compression algorithm was negotiated during the handshake. Supported compression algorithms include the null algorithm (i.e. no compression) and the `DEFLATE` algorithm.

The compressed payload is finally encrypted and authenticated based on the ciphersuite in use. There are three different approaches.

Stream-cipher-based ciphersuites. First, an authentication tag is computed using a message authentication code as

$$\mathsf{MAC}(\texttt{msg\_write\_key}, sn \| \texttt{content\_type} \| \texttt{version} \| len \| m),$$

where `msg_write_key` is the client-to-server or server-to-client MAC write key as appropriate, *sn* is the sequence number of the packet, *m* is the (optionally compressed) payload, and *len* is the length of the payload. The payload is concatenated with the MAC tag and then that plaintext is encrypted using the stream cipher in question. RC4 was the most widely used stream cipher in TLS; since it does not use an initialisation vector, the internal RC4 state persists across packets.

Block-cipher-based ciphersuites. TLS specifies the use of the cipher-block chaining (CBC) mode of operation for block ciphers. First, an initialisation vector (IV) is chosen for the packet. The computation of the IV depends on the TLS version: for SSL v3 and TLS 1.0, the IV of the first packet is the IV derived in the handshake protocol, while the IV of each subsequent packet is the last ciphertext block of the previous packet. For TLS 1.1 and higher, the IV is chosen at random. Next, a MAC is computed as described in the stream-cipher item above. The (optionally compressed) payload and MAC are concatenated, then padded with sufficient bytes to bring the length up to a multiple of the block length; all padding bytes should be the same and should be the padding length value (for example, if 12 bytes of padding are needed, then every byte of padding contains the value $\texttt{0x0C} = 12$). This data is then encrypted using the block cipher in question in CBC mode. This sequence of operations is sometimes referred to as 'MAC-then-encode-then-encrypt' (MEE). The most widely supported block cipher is AES; the use of DES is deprecated, and the use of Triple-DES is no longer recommended.

Authenticated-encryption-based ciphersuites. TLS 1.2 added support for *authenticated encryption with associated data* (AEAD). In these modes, a single algorithm is used for encryption and authentication of packets. The 'additional data' field is used to convey unencrypted data unauthentically; in the case of TLS, the additional data is the same as the non-message data in the MAC computation in stream-cipher-based ciphersuites: the sequence number, content type, version, and plaintext length. There are two supported block cipher modes of operation that provide AEAD: *counter mode with CBC-MAC* (CCM) and *Galois/counter mode* (GCM), both generally implemented using AES. Since these modes provide encryption and authentication using a single algorithm, only the encryption keys, not the MAC keys, are required.

6.4 Additional Functionality

In addition to the core cryptography task of mutual entity authentication and establishment of a secure channel, the TLS protocol performs several other operations, which are described in this section.

6.4.1 Compression

The TLS record layer protocol allows application data to be compressed. In the `ClientHello` and `ServerHello` messages, the parties negotiate whether or not to enable the `DEFLATE` compression algorithm [246, 362] (a combination of the LZ77 algorithm and Huffman coding, widely used in the ZIP and gzip formats).

The record layer protocol splits application data up into *fragments* of at most 2^{14} bytes. If compression is enabled, then each plaintext fragment is independently compressed using the `DEFLATE` algorithm before being encrypted. The receiver reverses the process, decrypting and then decompressing.

Because the amount of compression on a plaintext yields information about the amount of redundancy in the plaintext, TLS compression acts as a side channel, leaking some information about the plaintext content. This leads to a successful adaptive chosen plaintext attack against TLS that allows an attacker to recover a secret value (such as an HTTP cookie) by exploiting differences in the amount of compression. Details of the CRIME attack and related attacks appear in Sect. 6.11.3.

6.4.2 Session Resumption

Session resumption allows a client and server to use an *abbreviated handshake* to resume a previously established session. This saves both communication—using 1.5 round trips instead of 2.5—and computation, as generally no expensive public key operations are needed in session resumption.

There are two distinct mechanisms for session resumption.

- *Session IDs* [251, Sect. F.1.4]. When the initial session is established, the server includes in its `ServerHello` message a session identifier. The server stores the session's parameters and master secret in its cache. To resume a session, the client replays that session identifier in its `ClientHello` message. The server looks up the session in its cache. If the session is found, the server immediately sends the `Finished` message calculated using the stored master secret.
- *Session tickets* [649]. When the initial session is established, the server includes an additional handshake message just prior to `ChangeCipherSpec`: the `NewSessionTicket` message contains the server's state (including its master secret), encrypted and authenticated under a symmetric key known only to the server. To resume a sesion, the client replays that session ticket in its `ClientHello` message. The server decrypts the encrypted state and verifies its integrity; if it passes, the server immediately sends the `Finished` message, calculated using the master secret from the session ticket.

The main difference between session IDs and session tickets is about who stores state: with session IDs, the server stores state for each connection; with session tickets, the client stores the server's state for it. Session tickets reduce storage requirements for the server; as well, a collection of load-balancing servers can easily resume each others' sessions simply by sharing session ticket encryption keys, rather than needing to pool session ID states.

In 2014, Bhargavan *et al.* [96] discovered the *triple handshake attack*, in which an attacker performs a man-in-the-middle attack over three successive handshakes (with various combinations of session resumption and renegotiation), eventually leading to a successful client impersonation attack. Among the causes of the attack is the fact that TLS session resumption does not cryptographically bind the previous session's handshake to the new session's handshake. Details of the attack and countermeasures appear in Sect. 6.11.5.

6.4.3 Renegotiation

Renegotiation allows two parties who are using an existing TLS session to run a new handshake; upon completion of the new handshake, the session uses the newly established encryption and authentication keys for communication. Renegotiation allows parties to either (a) obtain a fresh session key, (b) change cryptographic parameters (such as to negotiate a new ciphersuite), or (c) change authentication credentials. The renegotiated handshake is run *inside* the existing encrypted record layer.

A principal use case of TLS renegotiation is client identity privacy. In the handshake protocol, the client sends its certificate in plaintext. If the client does not wish to reveal its identity over a public channel, it can instead run the first handshake anonymously, then renegotiate using its long-term credential; since the handshake messages for the renegotiation are transmitted within the existing record layer, the transmission of the client certificate is encrypted and the client has privacy of its identity.

Either the client or the server can initiate renegotiation at any time after a session is established. The client triggers renegotiation simply by sending a new `ClientHello` message. The server triggers renegotiation by sending a `HelloRequest` message which asks the client to send a new `ClientHello` message. Renegotiation is supported in SSL v3 and higher.

In 2009, Ray and Dispensa [623] described an attack against some applications supporting TLS renegotiation. As a consequence of the attack, two countermeasures were standardised. Details of the attack and countermeasures appear in Sect. 6.11.2.

6.5 Variants

Versions. There are currently six distinct versions of SSL/TLS. The Secure Sockets Layer (SSL) protocol version 2 was published by Hickman at Netscape in February 1995 (version 1.0 was never released publicly). Owing to security flaws in SSL v2, the protocol was completely reworked, and Netscape released SSL v3 in 1996 [283]. The Internet Engineering Task Force (IETF) made minor changes to SSL v3 and standardised it as the Transport Layer Security (TLS) protocol 1.0 in 1999. This protocol was revised in 2006 to TLS 1.1, particularly with changes in the use of initialisation vectors and padding in CBC mode encryption to protect against attacks identified by Möller [564] and Bard [53]. TLS 1.2 was published in 2008, and replaced the ad hoc MD-5 + SHA-1 pseudo-random

function with a ciphersuite-negotiated hash function, usually SHA-256, added new encryption modes for AES, namely Galois/counter mode (GCM) and CBC in counter mode (CCM), and incorporated functionality from various additional RFCs into the core standard. From 2014-2018, the next version of TLS was under development by the IETF, and was standardised at TLS 1.3 in August 2018. TLS 1.3 constitutes a major revision of the TLS protocol; see Sect. 6.14 for more information.

Ciphersuites. The IETF has standardised more than 320 ciphersuites,[1] each of which specifies a valid combination of the following cryptographic algorithms. Note that up to and including TLS 1.2, cryptographic algorithms cannot be negotiated independently, so not all combinations of the following algorithms are possible. (TLS 1.3 provides à la carte negotiation of each component individually.)

- Key exchange:
  - RSA key transport;
  - ephemeral finite-field or elliptic curve Diffie–Hellman;
  - none (null).
- Entity authentication:
  - RSA key transport;
  - RSA digital signatures;
  - finite-field (DSA) or elliptic curve (ECDSA) digital signatures;
  - static finite-field or elliptic curve Diffie–Hellman key exchange;
  - pre-shared keys;
  - password authentication using Secure Remote Password (SRP) protocol;
  - none (null).
- Bulk encryption:
  - RC4 or ChaCha20;
  - RC2, DES, Triple-DES, IDEA, or SEED in CBC mode;
  - AES-128 or AES-256 in CBC, CCM, or GCM mode;
  - ARIA or CAMELLIA in CBC or GCM mode;
  - none (null).
- Message authentication:
  - HMAC-MD5, HMAC-SHA1, HMAC-SHA256, or HMAC-SHA384;
  - AES-128, AES-256, ARIA, or CAMELLIA in GCM mode;
  - Poly1305;
  - none (null).

SSL v2 and v3, and TLS 1.0 also contained *export* ciphersuites: US export regulations in the 1990s restricted the export of cryptographic software or hardware with keys larger than 40 bits (for symmetric cryptography) or 512 bits (for RSA and finite-field Diffie–Hellman). As a result, 'export' ciphersuites were standardised in which the final encryption keys were derived from subkeys that were

[1] https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

only 40 (or 512) bits long. Regulations were relaxed starting in 1996, simplifying the export of commercial and open-source cryptography software. Export ciphersuites were removed from TLS in version 1.1, although some software still supports operations involving export-sized keys either directly or indirectly; see Sect. 6.9.5 for resulting attacks.

Datagram TLS. SSL/TLS are designed to work over a stream-oriented network protocol, namely the Transmission Control Protocol (TCP) which provides reliable, in-order delivery of packets. Datagram TLS (DTLS) is designed to work over datagram protcools, such as the User Datagram Protocol (UDP) and others which do not guarantee delivery or ordering of packets, and are used in a variety of applications, including simple query–response protocols such as domain name lookups (DNS) and media-streaming protocols such as the Real-time Transport Protocol (RTP). DTLS provides confidentiality and protection against message tampering and forgery for such applications. One notable distinction between TLS and DTLS is that DTLS generally does not terminate the session upon an error. There are currently two versions of DTLS: DTLS 1.0 [627] (based on TLS 1.1) and DTLS 1.2 [628] (based on TLS 1.2), as well as several standards which specify how to use DTLS to protect various higher-level protocols.

Extensions. After the publication of TLS 1.0 in 1999, the TLS extension mechanism was published [111, 264] which allows the client and server to send extensions on the `ClientHello` and `ServerHello` messages for various purposes. Extensions were incorporated into the main specification in TLS 1.2. Some of the functionality provided by extensions includes the client telling the server which of several domains it is trying to connect to, using the *server name indication* (SNI) extension; negotiation of elliptic curves and point formats; a list of certificate authorities trusted by the client; and a *heartbeat* extension which provides a keep-alive functionality.

Keying-material exporters. RFC 5705 [624] provides a mechanism to obtain keying material derived from the master secret that can be used for any purpose by an application. (Here the term 'exporters' is not related to 'export'-grade ciphersuites.) This mechanism uses the TLS PRF with distinct labels to derive export keys that are computationally independent from the TLS record-layer encryption and MAC keys. Among other purposes, this formalises how the Extensible Authentication Protocol (EAP) obtains keying material from TLS.

Implementation-specific behaviour. Various implementations have bugs that are exhibited in certain configurations. As a result, many TLS implementations have optional code that tries to work around bugs in other implementations.

6.6 Implementations

There are several important SSL/TLS implementations.

OpenSSL (`http://www.openssl.org`) is a leading open source implementation, licensed under a BSD-like licence. OpenSSL is split into two libraries:

`libcrypto`, which contains the core cryptographic algorithms, and `libssl`, which contains an implementation of SSL versions 2 and 3, TLS versions 1.0 to 1.3, and DTLS 1.0 and 1.2. OpenSSL is generally included as a standard library on most Unix-based operating systems. It is used by the Apache (via the `mod_ssl` module) and nginx web servers, and many command-line Unix tools, such as `curl` and `wget`. In 2014, several forks of OpenSSL appeared in response to software vulnerabilities in OpenSSL; prominent forks included LibReSSL (`http://www.libressl.org`) and Google's BoringSSL (`https://boringssl.googlesource.com/`).

GnuTLS (`http://www.gnutls.org`) is an open source implementation of SSL and TLS, licensed under the GNU Lesser General Public License v2.1. GnuTLS depends on the Nettle library (`http://www.lysator.liu.se/~nisse/nettle`) for its cryptographic algorithms. GnuTLS is used by a variety of GNU-licensed software.

Network Security Services (NSS) (`https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS`) is an open source implementation of SSL/TLS, licensed under the Mozilla Public License. Originally part of the Netscape Navigator web browser, NSS is used in the Mozilla Firefox browser and other Mozilla software, Google's Chrome browser, the Opera browser, and commercial software from RedHat, Oracle, and AOL, among others. NSS can be used with Apache via the `mod_nss` module.

Bouncy Castle (`http://www.bouncycastle.org`) and **Java Secure Socket Extension (JSSE)** are prominent open source Java implementations of SSL/TLS.

Microsoft and Apple each have their own implementation of SSL/TLS, named *SChannel* and *SecureTransport*, respectively, which ship with their desktop and mobile operating systems, and are, in particular, used by their web browsers Internet Explorer/Edge and Safari, respectively.

6.7 Security Analyses

The earliest work on the security of SSL, by Wagner and Schneier [724], examined various characteristics of SSL v3 and identified certain weaknesses. Since then, a systematic study of the security of TLS has been carried out in two lines of work: using provable security and using formal methods.

6.7.1 Provable Security

Historically, the TLS handshake and record layer protocols have been analysed separately in the provable-security setting.

As demonstrated throughout this book, there is a vast literature on authenticated key exchange protocols and their security. Chapter 2 details security models for AKE protocols, starting with the Bellare–Rogaway model and continuing with the Canetti–Krawczyk and eCK models. Central to all of those models is that a session

key should be indistinguishable from random: in the security definitions, the task of the adversary is to determine whether it has been given the real session key from a target session, or a random string of the same length. Unfortunately, a technicality of TLS prevents it from being proven secure in standard AKE models: the final messages of the TLS handshake protocol (the `Finished` messages) are sent over the encrypted record layer; an adversary who is given a challenge real-or-random session key can try to decrypt the ciphertexts to see if they decrypt correctly, thereby distinguishing real session keys from random ones.

Initial work on the provable security of the TLS handshake protocol looked at a *truncated* handshake protocol in which the `Finished` messages were either omitted or transmitted unencrypted. Jonsson and Kaliski [401] showed that a truncated version of the TLS handshake using RSA key transport is secure under an RSA-based assumption. Subsequently, Morrissey, Smart, and Warinschi [568] showed that a truncated version of the TLS handshake using either RSA key-transport or signed Diffie–Hellman is secure in the random oracle model; their approach was modular, in the sense that they first showed that the premaster secret agreement is secure in a weaker, one-way sense, then that this implied that the master secret is secure in the traditional AKE sense. Gajek *et al.* [288] showed that TLS is a secure, unauthenticated channels protocol in the universal composability setting, but this analysis provided only confidentiality of messages, not authentication of endpoints.

The record layer protocol was also investigated. Krawczyk [451] analysed a variant of the MAC-then-encode-then-encrypt approach using CBC mode encryption in the TLS record layer and showed that it achieved ciphertext indistinguishability (IND-CPA) as well as a certain form of ciphertext unforgeability, weaker than ciphertext integrity. Paterson, Ristenpart, and Shrimpton [605] identified a distinguishing attack against TLS when variable-length padding and short MACs were used, defined a stronger security notion, called *length-hiding authenticated encryption* (LHAE), and showed that the record layer protocol for TLS 1.1 and TLS 1.2, in CBC mode, satisfies this property.

In 2012, the first provable security analysis of a full, unaltered TLS ciphersuite was presented by Jager, Kohlar, Schäge, and Schwenk [392]. They defined a new security model for authenticated and confidential channel establishment (ACCE) protocols. Instead of simply establishing a key, an ACCE protocol establishes a secure channel which can then be used for communication. Formally, the model effectively combines the Bellare–Rogaway model for authenticated key exchange with the Paterson *et al.* model for length-hiding authenticated encryption; this overcomes the aforementioned problem involving the encrypted `Finished` messages that prevented TLS from being proven a secure authenticated key exchange protocol. Originally the model only addressed mutual authentication, but it has subsequently been extended to include the server-only authentication mode that is more widely used in practice.

The original ACCE paper of Jager *et al.* [392] showed that signed finite-field Diffie–Hellman TLS ciphersuites are secure ACCE protocols, assuming that the signature scheme is existentially unforgeable under chosen message attack, certain Diffie–Hellman problems are hard, and the bulk encryption scheme is a secure stateful length-hiding authenticated encryption scheme. Subsequently, several works

have analysed a variety of other ciphersuites, including RSA key transport and static Diffie–Hellman [442, 456], and pre-shared keys [489].

The ACCE model has also been extended to consider additional functionality. Giesen, Kohlar, and Stebila [304] extended the ACCE model to formalise the notion of renegotiation security in light of the renegotiation attack (described in Sect. 6.11.2). Bergsma *et al.* [89] considered multi-ciphersuite ACCE security as a result of the cross-protocol attack (described in Sect. 6.10.2).

There are also several alternative approaches to the ACCE model for proving security of TLS. Brzuska *et al.* [164] used a modular approach in which the key exchange is composed with the authenticated encryption scheme, under an additional constraint where the key from the key exchange has been shown to be 'suitable for' this type of composition.

Bhargavan *et al.* [98] implemented several TLS ciphersuites using a programming language that allows for formal verification that the code of the implementation meets provable security properties, and were thus able to derive security results for ciphersuites based on RSA key transport, signed Diffie–Hellman, and static Diffie–Hellman. Related work by several of the same authors [99] examined the TLS handshake in detail, deriving provable security results (supported by formal verification using the F# functional programming language and the EasyCrypt proof verification tool) for handshakes involving RSA key transport, signed Diffie–Hellman, and static Diffie–Hellman using a compositional approach that provides for security even with limited agility of long-term signing keys, i.e. when the same long-term keys are used in several ciphersuites.

A modular analysis of TLS using the recent *constructive cryptography* formalism [528] (related to *abstract cryptography*) has also been given [443].

6.7.2 Formal Methods

SSL and TLS have also been evaluated using formal methods, which use a variety of approaches in logic and theoretical computer science to give precise specifications of desirable protocol behaviour, and either demonstrate that the protocol achieves these goals or demonstrate a flaw. This approach often, but not always, involves automated tools. Three main classes of automated tools are model checkers (a.k.a. finite state analysis), which enumerate all reachable states in an execution to see if an undesirable state is reached; verifiers, which check some human-specified argument (e.g. a proof) against a formal specification; and theorem provers, which construct and verify a proof of a certain property, or alternatively sometimes find a counterexample.

Model Checking. Mitchell *et al.* [562] performed finite state analysis (model checking) using the Murφ tool. They applied the tool to a sequence of protocols, starting from an approximation of the SSL v2 handshake, and culminating in an approximation of the SSL v3 handshake; they simplified, among other things, the derivation of keys from the premaster secret. The tool successfully identified the main weaknesses in SSL v2 that motivated SSL v3. For (the approximation to) SSL v3, the tool found two downgrade attacks and a weakness in resumption,

but no other flaws. The model checking was run with some constraints, namely that it involved two clients, one server, no more than two simultaneous open sessions per server, and no more than one resumption per session.

Theorem Provers. Paulson [606] gave machine-checked proofs of TLS handshake authentication and session key security in both full and abbreviated handshakes using the Isabelle theorem prover, assuming idealised cryptographic functions; this 'inductive' approach reasons about an inductively defined set of protocol traces built from the protocol specification and adversary actions. Ogata and Futatsugi [588] gave an analysis of the TLS handshake using the OTS/CafeOBJ tool for equation reasoning, showing in the Dolev–Yao model (which assumes perfect cryptography and abstracts away the details of cryptographic operations) that the premaster secret is secure and that TLS handshake messages are authenticated.

Logic-Based Proofs. In 2005, He *et al.* [353] gave an analysis of the IEEE 802.11i protocol (including an analysis of TLS as a subprotocol) using a protocol composition logic (PCL). For TLS, their arguments showed that the TLS handshake messages are authenticated and that the premaster secret is secure in the face of a symbolic adversary. Kamil and Lowe [410] analysed the TLS handshake and record layer protocols in the strand spaces model, which uses the Dolev–Yao model. Their results included findings that the premaster secret key is secure, the handshake is authenticated, the record layer provides two authenticated streams, and those streams also have confidentiality.

Formal Methods Applied to Concrete Implementations. Formal verification tools have also been applied to concrete implementations of SSL/TLS. Jürjens [405] verified a Java implementation of SSL using an automated theorem prover for first-order logic, showing authentication of the handshake and secrecy of the session key in the Dolev–Yao model; the analysis works on the control-flow graph of the protocol execution. Chaki and Datta [186] applied the Aspier framework to perform a symbolic evaluation of the handshake authentication and session key secrecy of the TLS handshake as implemented in the OpenSSL library, in configurations of up to three clients and servers. Bhargavan *et al.* [97] gave an implementation of a simple yet compatible subset of TLS in the F# programming language, and used the CryptoVerif tool to provide symbolic and computational cryptographic security results. This was the first in a series of results by this set of authors; their later results implement additional levels of complexity and move from symbolic to computational results, and are noted in the section on provable security of TLS above. These results are part of the miTLS project (`http://www.mitls.org/`).

6.8 Attacks: Overview

Because of its importance, TLS has been subject to much study and analysis, and many attacks have been found on the protocol and on TLS implementations. A few

attacks date from the early days of SSL/TLS in the late 1990s (such as Bleichenbacher's attack and version downgrade attacks). In the early 2000s, some theoretical weaknesses were identified, but no realised attacks were publicised. Starting in 2009, many major attacks on various aspects of TLS were revealed, necessitating revisions to standards and coordinated release of patches. Many of these recent practical attacks are actually realisations of ideas from Wagner and Schneier's seminal analysis of SSL v3 [724], or other theoretical work from the same era.

Table 6.2 lists attacks against various aspects of SSL/TLS since its inception. The table is organised into several sections, depending on what aspect of SSL/TLS the attack is against. These include weaknesses in core cryptographic algorithms, attacks arising from how cryptographic components are used in TLS ciphersuites, attacks on non-cryptographic functionality in TLS, attacks on libraries implementing TLS, attacks on HTTP-based applications using TLS, and attacks on protocols using TLS.

RFC 7457 [666] summarises some of these attacks, as do Wikipedia's page on TLS and surveys by Meyer and Schwenk [552] and Clark and van Oorschot [218]. The Trustworthy Internet Movement's monthly SSL Pulse survey [621] reports statistics on the TLS configuration of popular websites, such as ciphersuite usage and vulnerability to known attacks.

The remainder of this chapter describes many of these attacks in more detail.

6.9 Attacks: Core Cryptography

The first class of attacks that we look at exploit properties of the cryptographic algorithms used in TLS. This includes both public-key primitives used in the TLS handshake and symmetric-key primitives used in the record layer.

6.9.1 Bleichenbacher's Attack on PKCS#1v1.5 RSA Key Transport

TLS ciphersuites using RSA key transport do not use 'schoolbook' RSA encryption; instead, they rely on the PKCS#1v1.5 standard for RSA encryption [408]. Bleichenbacher [118] discovered an attack in which access to a certain type of error information in PKCS#1 decryption can enable recovery of the secret key. It should be noted that Bleichenbacher's attack applies only to PKCS#1v1.5, not later versions of the PKCS#1 standard. However, RSA key transport in SSL/TLS from SSL v3 to TLS 1.2 uses PKCS#1v1.5.

For RSA encryption, the receiver generates a public key/secret key pair by picking two large, distinct secret primes p and q, computing the RSA modulus $n \leftarrow pq$, and picking values e, d such that $ed \equiv 1 \mod \phi(n)$, where $\phi(n) = (p-1)(q-1)$; the receiver's public key is (n, e) and the private key is d.

In 'schoolbook' RSA encryption, the sender represents a message M as an integer $m \mod n$, then computes the ciphertext as $c \leftarrow m^e \mod n$. The receiver can decrypt this by computing $m \leftarrow c^d \mod n$ and converting the integer m back into the message M. While this does indeed provide confidentiality, it does not protect against malleability. For example, $c^2 \mod n$ is the ciphertext corresponding to the

Table 6.2: Known attacks on SSL/TLS. * denotes a theoretical basis for a later practical attack

Target	Attack name	Year	References
Core cryptography			
RSA PKCS#1v1.5 decryption	Side channel – Bleichenbacher	1998*, 2014	[118]*, [553]
DES	Weakness – brute force	1998	[265]
MD5	Weakness – collisions	2005	[482]
RC4	Weakness – biases	2000*, 2013	[280, 515]*, [28]
RSA export keys	FREAK	2015	[92]
Diffie–Hellman export keys	Logjam	2015	[21]
RSA-MD5 signatures	SLOTH	2016	[101]
Triple-DES	Sweet32	2011*, 2016	[635]*, [100]
Crypto usage in ciphersuites			
CBC mode encryption	BEAST	2002*, 2011	[564]*, [260]
Diffie–Hellman parameters	Cross-protocol attack	1996*, 2012	[724]*, [529]
MAC–encode–encrypt padding	Lucky 13	2013	[29]
CBC mode encryption + padding	POODLE	2014	[565]
TLS protocol functionality			
Support for old versions	Jager *et al.*, DROWN	2015, 2016	[46, 393]
Negotiation	Downgrade to weak crypto	1996, 2015	[21, 92, 724]
Termination	Truncation attack	2007, 2013	[88, 684]
Renegotiation	Renegotiation attack	2009	[623]
Compression	CRIME, BREACH, HEIST	2002*, 2012,16	[422]*, [620, 632, 720]
Session resumption	Triple-handshake attack	2014	[96]
Implementation – libraries			
OpenSSL – RSA	Side channel	2005, 2007	[20, 608]
Debian OpenSSL	Weak RNG	2008	[710]
OpenSSL – elliptic curve	Side channel	2011–14	[161, 162, 756]
Apple – certificate validation	goto fail;	2014	[476]
OpenSSL – Heartbeat extension	Heartbleed	2014	[220]
Multiple – certificate validation	Frankencerts	2014	[160]
NSS – RSA PKCS#1v1.5 signatures	BERserk (Bleichenbacher)	2006*, 2014	[276]*, [473]
Multiple – state machine	CCS injection, SMACK	2014, 2015	[92, 483]
Implementation – HTTP-based applications			
Netscape	Weak RNG	1996	[309]
Multiple – certificate validation	'The most dangerous code...'	2012	[302]
Application-level protocols			
HTTP	SSL stripping	2009	[524]
HTTP server virtual hosts	Virtual host confusion	2014	[239]
IMAP/POP/FTP	STARTTLS command injection	2011	[722]

message m^2. One mechanism for preventing ciphertext malleability is to impose formatting and padding requirements on the message; the homomorphic property of RSA encryption would not a priori preserve that formatting requirement.

PKCS#1v1.5 encryption pads the message M as follows. Let k be the length in bytes of the RSA modulus n. The padded format of M is

$$m = 00||02||PS||00||M,$$

where 00 and 02 are single bytes, and PS is a padding string composed of $k - 3 - |M| \geq 8$ non-zero bytes. Once M is converted into m as above, the ciphertext is $m^e \mod n$. During decryption of a ciphertext c, the receiver again computes $m \leftarrow c^d \mod n$, then checks to see if m is *PKCS-conforming*, meaning that (a) the first byte $m_1 = 00$, (b) the second byte $m_2 = 02$, (c) bytes m_3 to m_{10} are all non-zero, and (d) at least one of the bytes from m_{11} to m_k is 00. If m is PKCS-conforming, then M is extracted. A ciphertext is said to be PKCS-conforming if its decryption is.

In Bleichenbacher's attack [118], we assume that an attacker has access to an oracle which indicates whether a given ciphertext is PKCS-conforming. An attacker can use such an oracle to decrypt a target ciphertext in a relatively small number of oracle queries; for a 1024-bit RSA modulus, around 2^{20} queries to the oracle suffice. The attack proceeds as follows. Let c_0 be the challenge ciphertext, which corresponds to PKCS#1v1.5 plaintext m_0 (and message M_0). The attacker chooses a small value s and computes $c' \leftarrow c_0 s^e \mod n$, then submits c' to the PKCS-conformance-checking oracle. If the ciphertext is PKCS-conforming, then the attacker knows that the first two bytes of $m_0 s \mod n$ are 00 and 02. Mathematically, this means that $2B \leq m_0 s \mod n < 3B$, where $B = 2^{8(k-2)}$. The attacker repeats this many times with different s values, recording each s such that $m_0 s \mod n$ is in the required range. The adversary can then use this information to narrow the range of values for m_0. For a detailed description of the range-narrowing process and an evaluation of the success probability, see [118], and improvements by Bardou *et al.* [54].

Straightforward implementations of SSL v3 enabled the attacker to obtain a PKCS-conformance-checking oracle using a timing attack. In particular, the pseudocode of early implementations was as follows.

1. Compute $m \leftarrow c^d \mod n$.
2. If m is not PKCS-conforming, then reject.
3. Otherwise, do additional cryptographic processing based on m (e.g. strong authentication).
4. If authentication fails, then reject; otherwise accept.

Since the strong-authentication operations in step 3 above require some non-trivial amount of time to perform, there is a timing mismatch between a reject in step 2 and a reject in step 4, allowing an adversary to learn whether the ciphertext was PKCS-conforming or not.

SSL v3 actually imposes some additional padding constraints of its own, with the effect that the 46-byte premaster secret PMS is padded to

$$m = 00||02||PS||00||03||00||PMS,$$

and hence the padding string *PS* is always of the same length (for the same RSA key size k). Imposing this additional 'SSL-conformance' check that the bytes 03||00 always appear in the correct place increases the number of oracle queries to around 2^{40}, making the attack harder but still feasible. Klíma *et al.* [434] proposed a variant of the attack and tested the attack in the wild, finding that nearly two-thirds of tested web servers were vulnerable at the time.

TLS versions 1.0–1.2 still use PKCS#1v1.5 encryption for RSA key transport, but impose additional requirements on implementations [251, Sect. 7.4.7.1]. In particular, before RSA decryption, implementations generate a random 48-byte string. If the decryption does not conform to the PKCS and SSL formatting and version requirements, processing continues, but using the randomly generated 48-byte string as the premaster secret rather than the decrypted value. In other words, the handshake protocol continues regardless of whether the ciphertext was PKCS- and SSL-conforming or not. Rejection only happens when authentication fails in the processing of the `Finished` message; since the attacker has forged a ciphertext, it expects that authentication will fail, and thus learns nothing about the PKCS conformance of the forged ciphertext.

Bleichenbacher's attack continues to plague TLS (and other) implementations. Specifically related to TLS implementations, Meyer *et al.* [553] successfully applied Bleichenbacher's attack against timing side channels in OpenSSL, JSSE, and Cavium, as well as an error message side channel in JSSE. Jager *et al.* [393] showed that even though TLS 1.3 (draft 10) does not include RSA key transport, and thus does not rely on PKCS#1v1.5 encryption, Bleichenbacher's attack on older versions of TLS at servers which use the same RSA certificate for TLS 1.3 can potentially lead to impersonation attacks on TLS 1.3 servers.

In 2016, Aviram *et al.* [46] presented the DROWN (Decrypting RSA using Obsolete and Weakened eNcryption) attack, which works against servers that use the same RSA key with SSL 2 and modern versions of TLS, up to TLS 1.2. They identified a Bleichenbacher RSA padding oracle in the SSL 2 protocol, which can then be used to decrypt TLS 1.2 ciphertexts. To decrypt at 2048-bit RSA TLS 1.2 ciphertext, their attack requires an attacker to observe 1,000 TLS handshakes, initiate 40,000 SSL 2 connections, and do 2^{50} offline work. They found that some 33% of publicly accessible HTTPS servers were vulnerable to this attack.

6.9.2 Bleichenbacher's Attack on PKCS#1v1.5 RSA Signature Verification

In 2006, Bleichenbacher [276] identified another vulnerability involving PKCS# v1.5, this time in how implementations parse data during signature verification. Bleichenbacher observed that the OpenPGP implementation of signature verification did not correctly parse the padding of the hash value after exponentiation with the public exponent. Bleichenbacher's 2006 attack required that the RSA public exponent be 3 and that the RSA modulus be quite large (e.g. 3072 bits); a subsequent improvement by Kühn *et al.* [460] allowed for smaller moduli and demonstrated how to use the attack against CA, intermediate, or server certificates used by TLS web browsers,

including a specific attack against the NSS library. This technique was again applicable in a 2014 attack, called the 'BERserk' attack, on the ASN.1 parsing of padding in PKCS#1v1.5 RSA signature verification in the NSS library, independently discovered by Delignat-Lavaud and Intel Security [473].

6.9.3 Weaknesses in DES, Triple-DES, MD5, and SHA-1

DES. The Data Encryption Standard (DES) was supported for encryption in CBC mode in SSL v2, SSL v3, TLS 1.0, and TLS 1.1. DES has an effective key size of 56 bits, far below the level needed for security against attackers today. Indeed, in 1998, the Electronic Frontier Foundation (EFF) built a DES cracker for under US$250,000 that could perform an exhaustive key search in less than 3 days [265], and in 2006 Kumar *et al.* [461] built an FPGA-based machine for under US$10,000 that performs an exhaustive key search in less than 9 days. DES ciphersuites were not included in TLS 1.2, and their use in earlier versions was recommended to be discontinued [268].

Triple-DES. TLS versions up to 1.2 still support Triple-DES (three-key encrypt–decrypt–encrypt) in CBC mode, which has an effective security of 112 bits. No significant weaknesses are known in the core Triple-DES function, and, in guidance dated January 2012, NIST continued to allow the use of Triple-DES for encryption of 'sensitive unclassified' data. However, CBC mode is known to suffer from collision attacks [635] which require a number of ciphertext blocks up to the birthday bound on the block size, so block ciphers with small block sizes are at risk. Triple-DES has the same block size as DES: 64 bits. Bhargavan and Leurent [100] observed that the amount of ciphertext required to carry out such an attack against Triple-DES is only 32 GB, and showed how to use the attack to recover secret session cookies from HTTPS traffic encrypted using Triple-DES; this was called the 'Sweet32' attack. The guidance following the attack was to disable Triple-DES ciphersuites.

MD5. The MD5 hash function is used in several places in various versions of TLS. In TLS 1.0 and TLS 1.1, the following apply.

1. Ciphersuites using RSA signatures sign the concatenation of the MD5 and SHA-1 hashes of the `ServerKeyExchange` message.
2. The TLS PRF combines outputs from HMAC-MD5 and HMAC-SHA-1.
3. The message authentication code can be HMAC-MD5 or HMAC-SHA-1.
4. X.509 certificates can be signed using RSA with MD5 hashes or SHA-1 hashes.

In TLS 1.2, the following apply.

1. The MAC can be HMAC-MD5 or HMAC-SHA-1.
2. X.509 certificates can be signed using RSA with MD5 hashes or SHA-1 hashes.

The collision resistance of MD5 is completely broken. Following Wang *et al.*'s initial discovery of a collision in MD5 [730], Lenstra and de Weger [482] constructed two different X.509 certificates containing identical signatures using a

'meaningful' MD5 collision; Stevens, Lenstra, and de Weger [696] later constructed colliding X.509 certificates, one of which was a normal certificate (that they got signed by a commercial CA) and one of which included flags for acting as a certificate authority, thus giving them the ability to issue fraudulent certificates that would be browser-trusted. This means that use of 4 in the list above in TLS 1.0 and TLS 1.1 and use of 2 for TLS 1.2 are insecure.

Conventional wisdom would be that the other uses of MD5 in TLS mentioned above do not immediately become insecure owing to the failure of collision resistance: signatures involving the concatenation of the MD5 and SHA-1 hashes remain secure if SHA-1 is collision resistant (theoretical attacks are known, but no collision has been published to date); and HMAC can remain secure both for message authentication and as a PRF under weaker conditions than collision resistance [68]. Despite this, these particular uses of MD5 in TLS are still problematic. In 2016, Bhargavan and Leurent [101] identified a class of "transcript collision" attacks (which they called the SLOTH attack). Their immediately practical attacks (48 core hours of parallelisable computation) included impersonation of clients or servers using RSA-MD5 certificates for authentication, and the breaking of `tls-unique` channel binding (which relies on HMAC-MD5 truncated to 96 bits); they also projected the complexity of finding a collision in the concatenated MD5 and SHA-1 hashes of the handshake transcript. At the time of the SLOTH disclosure, many software packages, and 32% of TLS servers, accepted RSA-MD5 signatures, although commercial CAs have not been issuing for RSA-MD5 certificates for many years.

SHA-1. The SHA-1 hash algorithm is also used in various places in TLS. The projected complexity of finding a SHA-1 hash collision is between $2^{60.3}$ and $2^{65.3}$ operations [695], no collisions were publicly known as of October 2015. Major browser vendors (including Google, Microsoft, and Mozilla) and CAs are transitioning to SHA-256; CAs have been required to not issue SHA-1 certificates since January 2016, and browsers have been treating SHA-1 certificates as insecure since January 2017.

6.9.4 RC4 Biases

In many TLS ciphersuites, the RC4 stream cipher is used for bulk encryption. RC4 was for many years viewed as preferred compared with DES (RC4 having larger keys), Triple-DES (RC4 being faster and easier to implement) and initial implementations of AES (RC4 again being faster than early software-only implementations of AES). In December 2018, Qualys SSL Labs' SSL Pulse [621] reported that 15.6% of popular web servers had RC4 support enabled, and 1.6% would negotiate an RC4-based ciphersuite even with modern browsers.

It has long been known that the RC4 stream cipher is less than ideal in a variety of ways. Sen Gupta *et al.* [662] provided an excellent survey of the history of RC4 cryptanalysis over the years, including the existence of weak keys, the partial recovery of the initial key from the internal state and from the keystream, and observed biases in the keystream.

In 2013, AlFardan *et al.* [28] demonstrated two nearly practical plaintext recovery attacks on RC4 as used in TLS based on prior known biases and some novel improvements. In their attack scenario, a fixed plaintext is encrypted using RC4 many times in succession, using the same or independent RC4 keystreams.

Their first attack, based on single-byte bias, is against the initial 256 bytes of the RC4 keystream: if the same plaintext is encrypted many times under distinct RC4 keys, then plaintext bytes can be recovered. In TLS, the keystream is used to encrypt the last handshake message (the 36-byte `Finished` message) which changes each session, but the next 220 bytes are application plaintext: a careful attacker may be able to cause a client to send the same request many times. In HTTP, cookies may appear in the first 220 bytes (though browsers often send more HTTP headers before the cookies); in the IMAP protocol for email collection, the user's password typically appears in the first 220 bytes. The attack works based on statistically averaging many ciphertexts, considering known biases in the first 256 bytes of RC4 keystreams. In AlFardan *et al.*'s experiments, the first 40 bytes of TLS application data were each recovered with a success rate of over 50% per byte using 2^{26} sessions, and all 220 bytes of application data with a success rate of over 96% per byte using 2^{32} sessions.

These attacks on RC4 have subsequently been improved. In March 2015, Garman *et al.* [292] demonstrated proof-of-concept password recovery attacks on certain application-layer protocols (BasicAuth and IMAP) when using RC4 ciphersuites in TLS. Their attacks achieved a significant success rate using 2^{26} ciphertexts. The main advantage comes from assuming the secret password is not distributed uniformly, instead using knowledge of typical password distributions, for example based on leaked corpuses of passwords [126].

It has been previously suggested [555] that the first few hundred bytes of the RC4 keystream should be dropped owing to known biases; however, the TLS standards do not provide a mechanism for doing so. AlFardan *et al.* gave a second attack, based on double-byte biases, that allows recovery of plaintext bytes anywhere in the ciphertext, not just the first 256 bytes. It is based on biases in pairs of bytes in the RC4 keystream. In their experiments, 16 consecutive bytes could be recovered with a 100% success rate using statistical analysis of 13×2^{30} ciphertexts. Vanhoef and Piessens [721] identified new biases and exploited them to be able to recover a 16-byte cookie with a 94% success rate using a novel statistical analysis of 9×2^{27} ciphertexts.

After the BEAST attack [260] against CBC mode in TLS in 2011, it was recommended that RC4 be adopted as the preferred ciphersuite. After the attacks by AlFardan *et al.*, it was recommended that RC4 be avoided, and that CBC mode with a certain countermeasure be used or, preferably, that the Galois/counter mode of authenticated encryption in TLS 1.2 be used where supported. In February 2015, the IETF TLS working group approved an RFC to prohibit the use of RC4 in all versions of TLS [619].

6.9.5 Weak RSA and Diffie–Hellman: FREAK and Logjam Attacks

As noted above, early versions of SSL included support for *export ciphersuites* which used shorter keys, as required by US export regulations. For RSA encryption and finite-field Diffie–Hellman key exchange, this meant the use of 512-bit (or shorter) keys. This key size would not be considered secure by modern standards, as RSA keys of this size could be factored in 1999. With the lapsing of these particular export regulations, modern TLS clients do not offer export ciphersuites. Nonetheless, many TLS implementations still include code supporting either export ciphersuites directly, or smaller keys indirectly. This old code led to the discovery of two major vulnerabilities in 2015.

Beurdouche *et al.* [92] discovered the so-called FREAK ("Factoring RSA Export Keys") attack, which exploits a state machine vulnerability in OpenSSL, Microsoft's SChannel library, and Apple's Secure Transport library, to trick a client and server into using an export-grade RSA key as long as the server supports RSA export ciphersuites, even if the client does not.

The FREAK attack works as follows. It requires a vulnerable client and a server that supports export-grade RSA key transport ciphersuites. First, the client sends a `ClientHello` message with a (non-export) RSA ciphersuite supported. The man-in-the-middle (MITM) replaces the supported ciphersuites with an export RSA ciphersuite and forwards it to the server. When the server responds with a `ServerHello` message for the RSA export ciphersuite, the MITM replaces it with a standard RSA ciphersuite and forwards it to the client. The server will also send its RSA certificate (which may be, for example, 2048 bits) as well as a `ServerKeyExchange` message containing an export-grade (e.g. 512-bit) ephemeral RSA public key. The MITM forwards these along to the client. The client *should* reject at this point, since no honest server running a non-export RSA ciphersuite would send a `ServerKeyExchange` message, but, owing to state machine bugs, several client implementations did not reject, and instead proceeded to use the ephemeral RSA public key for key transport. In particular, these clients would respond with a `ClientKeyExchange` message which contained the random premaster secret encrypted under the export-grade ephemeral RSA public key. The MITM needs to factor the RSA public key, derive all secrets, and then use these secrets to forge `Finished` message MAC tags to trick the client and server into accepting the altered transcripts. While factoring a 512-bit RSA key is possible, it would still take weeks with 2015 technology; however, many implementations will cache ephemeral RSA public keys, since RSA key generation is costly, so it may be possible to carry out the attack using sufficient pre-computation. Affected implementations were patched prior to the release of the paper.

Later in 2015, Adrian *et al.* [21] discovered the Logjam attack, which similarly allows a MITM attacker to downgrade the cryptography used in TLS, this time downgrading ciphersuites using finite-field Diffie–Hellman for forward secrecy to export-grade (or lower!) keys. The attack applied against all major browsers (Internet Explorer, Chrome, Firefox, Opera, and Safari). (Interestingly, Safari accepted finite-field Diffie–Hellman groups as small as 16 bits.) Using Internet-wide scan-

ning tools, the researchers identified that 8.4% of the Alexa Top 1 Million HTTPS domains allowed export-grade ephemeral Diffie–Hellman key exchange.

The Logjam attack works as follows. It requires a vulnerable client and a server which supports export-grade finite-field ephemeral Diffie–Hellman ('`DHE_EXPORT`') ciphersuites. First, the client sends a `ClientHello` message with a (non-export) finite-field Diffie–Hellman ciphersuite supported. The MITM replaces the supported ciphersuites with an export-grade finite-field Diffie–Hellman ciphersuite and forwards it to the server. When the server responds with a `ServerHello` message for the export DH ciphersuite, the MITM replaces it with a standard Diffie–Hellman ciphersuite and forwards it to the client. The server also sends its export-grade ephemeral Diffie–Hellman key in a `ServerKeyExchange` message, which the MITM forwards to the client. A vulnerable client implementation will accept the server's export-grade ephemeral Diffie–Hellman public key even though the client and server did not explicitly negotiate an export ciphersuite, and respond with its ephemeral Diffie–Hellman public key. The MITM needs to compute the shared Diffie–Hellman secret, derive all secrets, and then use these secrets to forge `Finished` message MAC tags to trick the client and server into accepting the altered transcripts.

Two implementation details make it feasible for the attacker to complete the attack. First, 92% of web servers supporting `DHE_EXPORT` ciphersuites use one of two standardised 512-bit finite-field groups. While each server will use a distinct ephemeral public key from this group, the number field sieve discrete logarithm algorithm can be structured to make use of pre-computation for the group that is independent of the specific discrete logarithm being computed. Second, many web servers, including Apache, nginx, and Microsoft's SChannel, reuse server ephemeral Diffie–Hellman keys. On Adrian *et al.*'s computing cluster, with 1 week of pre-computation for each group, computing individual 512-bit discrete logarithms took on average 70 seconds. The paper includes a discussion about the potential of state-level attackers using greater computing power for 768-bit and even 1024-bit finite-field Diffie–Hellman key exchange, and suggests that, based on the Snowden documents, the National Security Agency may be employing this technique to decrypt some TLS connections as well as virtual private networking connections established using IPsec's Internet Key Exchange (IKE) protocol with a specific 1024-bit finite-field Diffie–Hellman group.

One notable characteristic of both the FREAK and the Logjam attacks is that they exploit the fact that in SSL and TLS up to version 1.2, authentication of the transcript is done using a MAC rather than a signature, and the MAC is computed using (a key derived from) the master secret, *the security of which is itself negotiated inside the protocol*. (Note that the server's signature is only over the nonces and the raw ephemeral public key, and excludes the negotiation in the `ClientHello`/`ServerHello` messages.) In other words, the key used to authenticate the transcript and show that a downgrade attack has not occurred (i.e. that the parties agree on the `ClientHello`/`ServerHello` messages) is being downgraded prior to authentication. TLS 1.3 aims to avoid this type of attack by using the long-term public key to sign the entire transcript.

6.10 Attacks: Crypto Usage in Ciphersuites

We next examine attacks in which the cryptographic algorithms in the TLS ciphersuite interact in an unfortunate way with the other protocol components.

6.10.1 BEAST Adaptive Chosen Plaintext Attack and POODLE

The use of block ciphers (DES, Triple-DES, AES) in cipher block chaining (CBC) mode in SSL v3 and TLS 1.0 is vulnerable to an adaptive chosen plaintext attack owing to the manner in which initialisation vectors for different requests in the same SSL/TLS connection are set. This vulnerability was observed by Möller in 2002 [564] and Bard in 2004 [53], but neither was able to demonstrate an actual attack. In 2011, Rizzo and Duong demonstrated a working attack, which they called the BEAST attack (a 'backronym' for 'Browser Exploit Against SSL/TLS') that allowed them to recover short secret strings in known locations in HTTP plaintexts [260, 625, 650].

CBC mode is one of several block cipher modes that allow an arbitrary-length message m to be split into blocks $m_1 \| m_2 \| m_3 \| \ldots \| m_n$ and encrypted by a fixed-length cipher. In CBC mode, the first block of plaintext, m_1, is XORed with an initialisation vector iv, then encrypted using the key k:

$$c_1 \leftarrow \{m_1 \oplus iv\}_k.$$

Subsequent plaintext blocks are encrypted in a similar way, except that the initialisation vector is replaced with the previous ciphertext block:

$$c_i \leftarrow \{m_i \oplus c_{i-1}\}_k.$$

In SSL and TLS, the initialisation vector is derived from the master secret using the PRF. In many applications, including HTTPS, the same SSL/TLS connection is used for multiple requests. For example, in a web browser, the SSL/TLS connection will be established for the first request to a server, and then subsequent requests (including page resources such as images, JavaScript, applets, and later HTML pages) may all be sent over the same connection. SSL v3 and TLS 1.0 do *not* choose a new initialisation vector for each subsequent request within the same TLS connection: instead, they simply continue cipher-block chaining, using the last ciphertext block of the previous request as the initialisation vector for the next request.

The attack allows an adversary to test whether a particular guess at a plaintext block has a particular value. Suppose the adversary has observed the transmission of ciphertext $c_1 \| c_2 \| \ldots \| c_n$ and wishes to determine whether plaintext block m_j equals some guessed plaintext m^*. Now, the adversary knows that the next plaintext block sent will be encrypted with initialisation vector c_n. The adversary directs the user to send the next plaintext, block $m_{n+1} = c_{j-1} \oplus c_n \oplus m^*$; the user will compute the ciphertext

$$c_{n+1} = \{m_{n+1} \oplus c_n\}_k = \{c_{j-1} \oplus c_n \oplus m^* \oplus c_n\}_k = \{c_{j-1} \oplus m^*\}_k.$$

If $m_j = m^*$, then $c_{n+1} = c_j$, allowing the adversary to verify whether or not its guess of m^* for plaintext block j was correct.

The above attack allows the adversary to verify guesses of a single block one at a time. In AES, for example, a single block is 128 bits long; an adversary trying to guess a whole secret block could require up to 2^{128} operations, as much work as a brute force key recovery attack. However, Rizzo and Duong showed how an attacker who can inject plaintext near the beginning of a request can adjust how the plaintext is broken across block boundaries, isolating just a single unknown byte at a time. For example, suppose the attacker can cause the victim to make an HTTP request to a given URL, and that the cookie (which contains the secret value the attacker is targeting) comes immediately after, in a request like this:

$$\underbrace{\texttt{GET /↩Cookie: s=}}_{16}\underbrace{\texttt{1234567890123456}}_{16}$$

If the plaintext is encrypted in blocks of 16 bytes (128 bits), notice that the first block of 16 bytes is entirely known to the adversary, but the second block of 16 bytes is entirely unknown. If the adversary can change the URL that the victim requests, it can control where the block boundary falls. For example, if the adversary causes the victim to request a 15-character URL such as `abcdeabcdeabcde`, then the block boundaries fall like this:

$$\underbrace{\texttt{GET /abcdeabcdea}}_{16}\underbrace{\texttt{bcde↩Cookie: s=1}}_{16}\underbrace{\texttt{234567890123456}}_{15}$$

The second block (`bcde↩Cookie: s=1`) now contains only a single unknown byte, and thus can be found by carrying out the above attack using only 256 plaintext guesses. Once that byte is found, the adversary can shift the boundary again for the next unknown byte:

$$\underbrace{\texttt{GET /abcdeabcdeabcd}}_{16}\underbrace{\texttt{↩Cookie: s=12}}_{16}\underbrace{\texttt{34567890123456}}_{14}$$

and so on, allowing it to recover a k-byte secret with just $256k$ guesses.

To carry out this attack, the adversary needs to be able to observe ciphertexts, know the format of requests from the victim and be able to cause the victim to make multiple requests with the same secret, while being able to inject plaintext to control where the block boundary falls and to test guesses. Modern web browsers have a variety of communication technologies that have the potential to allow the adversary such powers, such as asynchronous Javascript requests and HTML5 WebSockets, as well as APIs in plugin technologies such as Java's URLConnection API and Silverlight's WebClient API. In most cases, the browser enforces a same-origin policy—preventing scripts from one domain sending data to another domain—that make it difficult to carry out the attack. Rizzo and Duong made use of a zero-day exploit in Java that bypassed the same-origin policy restrictions; this exploit has subsequently been patched, and to date no other means of carrying out the BEAST attack has been exhibited.

One of the major changes introduced in TLS 1.1 was the use of explicit IVs in CBC mode ciphers to prevent this attack. This suffices as a countermeasure to the BEAST attack. However, deployment of web browsers and servers supporting TLS 1.1 was slow; in early 2012, just after the BEAST attack, less than 1% of SSL-enabled websites surveyed by SSL Pulse supported TLS 1.1 or TLS 1.2. As a result, the immediate recommendation at the time of the BEAST attack was to switch to ciphersuites using the RC4 stream cipher; this recommendation has since been rescinded owing to subsequent attacks on RC4 described below. Browsers have since been updated to do so-called $1/n-1$ record splitting: the first block in each new message contains only a single byte of plaintext, the second block contains the next $n-1$ bytes (here n is the block size in bytes), and then the remaining blocks are full size. This has the effect of randomizing the IV in a (relatively) backward-compatible way [475].

In 2014, Möller, Duong, and Kotowicz [565] described a more powerful form of the BEAST attack against CBC-mode encryption SSL v3 in an attack which they called the POODLE (Padding Oracle On Downgraded Legacy Encryption) attack. In CBC-mode encryption in SSL v3, plaintext is concatenated with the MAC over the plaintext, then padding is applied to pad it out to a multiple of the block length L. The padding must be between 1 and L bytes, and the last byte of the last block is the length of the padding. There are no requirements on the content of the padding, and it is not covered by the MAC, so the integrity of the padding is not verified when decrypting.

The POODLE attack allows an attacker to decrypt a secret value inside the plaintext as follows. The attack is simpler if we assume the full last block is padding, which can be achieved easily by an attacker who has partial control over the plaintext. The attacker takes the ciphertext block it is trying to decrypt and uses it as the final ciphertext block. The decryption algorithm then XORs this with the previous block of ciphertext, which is known to the adversary. If the last byte of the result is the same as the expected padding length, then the block will decrypt successfully, otherwise an error will occur. Thus, with a 1-in-256 chance, the attacker can learn the last byte of one block of plaintext. Once one byte of the plaintext has been learnt, an attacker with partial chosen plaintext abilities can request ciphertexts with an additional byte, shifting the next byte of the target secret into the last byte of a block, and repeat.

While the POODLE attack can be defeated with a clever record-splitting technique similar to the $1/n-1$ record-splitting technique as described for the BEAST attack above, the preferred technique is to simply avoid use of SSL v3. Since some servers still support only SSL v3, some clients continue to offer SSL v3 support, to which an attacker could try to cause clients to downgrade. A recent new feature of TLS, the `TLS_FALLBACK_SCSV` signalling-ciphersuite value, can prevent downgrade attacks and thus protect clients from the POODLE attack [474].

6.10.2 Cross-Protocol Attack on Diffie–Hellman Parameters

As previously identified, TLS supports many different ciphersuites, and most deployments also support several ciphersuites. For example, a standard server might support ciphersuites using RSA key transport, RSA-signed finite-field ephemeral Diffie–Hellman, and RSA-signed elliptic curve ephemeral Diffie–Hellman, using AES encryption in CBC or Galois/counter mode, and with either SHA-1 or SHA-256 hash functions. However, in almost all installations, each server has only a single RSA long-term key, which is reused across all supported ciphersuites. In fact, popular web servers such as Apache only allow the server administrator to install a single key for each long-term key algorithm. Reuse of keying material—also known as *key agility*—can sometimes result in a vulnerability, either because secrets used in one algorithm may leak information when used in another algorithm, or because data encrypted/authenticated by one protocol may be unintentionally useful in another protocol.

Recall from Sect. 6.3.1 that, in ciphersuites where authentication is based on digital signatures, the server signs the `ServerKeyExchange` message. The contents of the `ServerKeyExchange` message depend on the type of key exchange method and are shown in Fig. 6.2. For ephemeral Diffie–Hellman key exchange (either finite-field or elliptic curve), the `ServerKeyExchange` message includes the description of the group as well as the server's ephemeral public key. In RSA key transport ciphersuites in SSL v3 and the export ciphersuites of TLS 1.0, the server may also send an RSA public key for encryption.

Wagner and Schneier [724] identified potential cross-protocol attacks in SSL v3 in 1996. They observed that the data structure to be signed—either `ServerRSAParams` or `ServerDHParams`—does not explicitly indicate what the type of the data structure is; instead, the data type is inferred by each party based on the negotiated ciphersuite. Wagner and Schneier hypothesised that it may be possible for an attacker to convince a client to misinterpret one type of data structure as another.

In 2012, Mavrogiannopoulos *et al.* [529] noted that, while the exact attack of Wagner and Schneier would not work owing to a subtlety in how TLS packets were processed, a similar attack could be executed. In 2006, support for elliptic curve Diffie–Hellman key exchange had been added to TLS [106]; in the relevant ciphersuites, the `ServerKeyExchange` message contains the description of the elliptic curve parameters and the server's ephemeral public key point. The specification allows curves to be specified either as one of several predefined *named curves* using a single index value, or as an *explicit curve* by specifying the prime or irreducible polynomial, curve equation coefficients, base point, and group order. When explicit curves are used, Mavrogiannopoulos *et al.* observed that it is possible to construct a `ServerECDHParams` data structure that is also a valid `ServerDHParams` data structure. Moreover, with careful choices of the curve parameters, a non-trival proportion (around 1 in 2^{40}) of elliptic curve ephemeral public keys will, when the `ServerECDHParams` data structure is interpreted as a `ServerDHParams` structure, result in a modulus that is sufficiently smooth to allow efficient factoring, enabling the attacker to compute the premaster secret and impersonate the server in the con-

```
struct {
  select (KeyExchangeAlgorithm):
    case rsa:
      ServerRSAParams params;
      Signature signed_params;
    case dhe_dss, dhe_rsa:
      ServerDHParams params;
      Signature signed_params;
    case ec_diffie_hellman:
      ServerECDHParams params;
      Signature signed_params;
} ServerKeyExchange
```

Note that `case rsa` only applies in some scenarios of SSL v3 and TLS 1.0.

```
struct {
  opaque rsa_modulus;
  opaque rsa_exponent;
} ServerRSAParams;

struct {
  opaque dh_p<1..2^16-1>;
  opaque dh_g<1..2^16-1>;
  opaque dh_Ys<1..2^16-1>;
} ServerDHParams;

struct {
  ECCurveType curve_type = 1;
  opaque      prime_p <1..2^8-1>;
  ECCurve     curve;
  ECPoint     base;
  opaque      order <1..2^8-1>;
  opaque      cofactor <1..2^8-1>;
  opaque      point <1..2^8-1>;
} ServerECDHParams;
```

Fig. 6.2: `ServerKeyExchange` data structures for signed ciphersuites in TLS

nection. Fortunately, as of this writing no major TLS server implementations support explicit elliptic curve parameters, so the attack remains theoretical. Motivated by this attack, Bergsma *et al.* [89] extended the ACCE security notion to consider multi-ciphersuite security and characterised the security of certain subsets of TLS ciphersuites with long-term key reuse across ciphersuites.

6.10.3 Lucky 13 Attack on MAC-Then-Encode-Then-Encrypt

The use of CBC mode for block ciphers in the TLS record layer follows a MAC-then-encode-then-encrypt pattern: a MAC is computed over the plaintext, then the plaintext and MAC tag are padded to a multiple of the block length, then the plaintext+MAC+padding is encrypted. The padding must have a specified format. In order to prevent one type of timing attack, the TLS standards require that a MAC check still be performed even if the padding is invalid. However, it becomes unclear as to which data the MAC should be computed over, so the RFC recommends processing as if there was a zero-length pad. In contrast, no processing of a validly padded and encrypted message would ever be performed with a zero-length pad. This difference opens up the possibility of a small timing side channel in the form of a padding oracle attack. The RFCs indicated it was "not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal" [251, Sect. 6.2.3.2].

AlFardan and Paterson [29] demonstrated how to exploit this small timing side channel, including practical attacks on open source implementations. The name 'Lucky 13' comes from what they describe as a "fortuitous alignment of various factors such as the size of MAC tags, the block cipher's block size, and the number of header bytes". Ciphertexts with invalid padding will result in an error message appearing at a slightly different time than for ciphertexts with valid padding but an invalid MAC tag.

In their most general attack on TLS in OpenSSL, an attacker on the same LAN segment is able to recover a full plaintext block using roughly 2^{23} sessions, provided that the same plaintext is sent in multiple sessions. More specific variants are more effective. It is possible to use the attack technique to distinguish the encryptions of two chosen messages in just a few sessions. Partial plaintext recovery attacks against TLS and DTLS are possible using fewer sessions. Attacks against DTLS are more effective: since DTLS is non-reliable, errors in DTLS are not fatal to the session, so it is much more practical to carry out repeated queries against the session. All major open source TLS implementations were vulnerable to attack.

One countermeasure proposed was to remove the timing side channel using a careful implementation, although the authors of the attack cautioned that it would be difficult to remove all related timing side channels, especially in DTLS implementations. Adding random timing delays was noted to be relatively ineffective, adding only a small increase in statistical uncertainty that could be averaged out with additional samples. The alternatives proposed were to switch away from CBC mode, either to RC4 or to authenticated encryption modes. The recommendation to switch to RC4 was with the caveat that RC4 was known to have some statistical weaknesses, and the same authors (plus a few new co-authors) subsequently demonstrated practical attacks against RC4-based ciphersuites, invalidating that recommendation (see Sect. 6.9). Authenticated encryption modes are only supported in TLS 1.2, leaving lower versions stuck with a difficult choice. An option to switch to pad-then-encrypt-then-MAC was standardised in 2014 [339] but does not seem to be widely implemented as of the time of writing.

6.11 Attacks: Protocol Functionality

This section examines attacks which arise owing to the extensive range of functionality provided in TLS. Unfortunately, flexibility which can be useful to applications can sometimes also open up opportunities for attackers.

6.11.1 Downgrade Attacks

All existing versions of SSL and TLS have compatible packet formats and in most applications run over the same network ports, but have different security properties, so there is the risk that devices which support multiple versions of SSL/TLS may be subject to a downgrade attack in which they are tricked into using a lower version than they both support.

Let us briefly recall the version negotiation mechanism. In the `ClientHello` message, the client sends the highest version it supports. In the `ServerHello` message, the server responds with the highest version it supports that the client also supports.

The risk of downgrade attacks in SSL/TLS was identified as early as Wagner and Schneier's 1996 paper [724]. SSL v2 was vulnerable to a ciphersuite rollback attack: since the ciphersuite negotiation was not authenticated in SSL v2, an active

attacker could replace one party's list of supported ciphersuites with the weakest mutually supported ciphersuite (for example, an export ciphersuite or one with a weak algorithm).

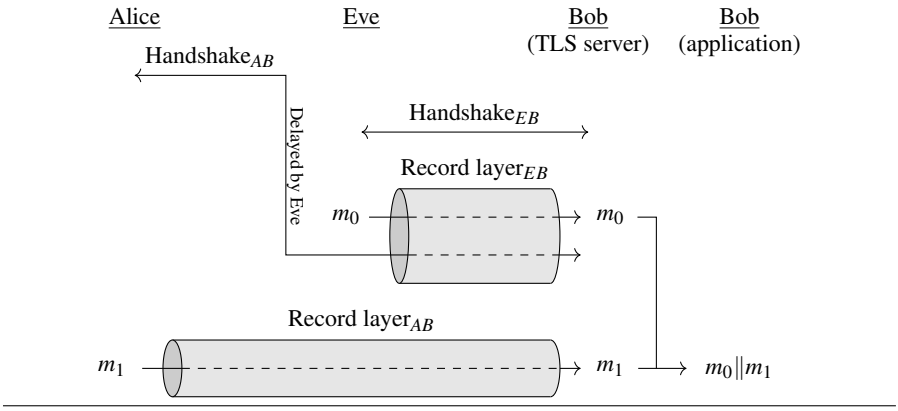
SSL v3 introduced a countermeasure to this attack: the complete handshake transcript was authenticated using a MAC with (a key derived from) the shared session key. This was designed to protect both version negotiation and ciphersuite negotiation. While this provides immediate protection against the naive downgrade attack on SSL v2 noted above, it is still flawed: the security of the version and ciphersuite negotiation mechanism comes from mutual authentication of the transcript, the transcript is authenticated *using the shared session key*, and the algorithm for computing this is determined by the ciphersuite and version being authenticated. This cyclic dependency places negotiation security at risk: a man-in-the-middle attacker who can compute the shared session key in real time can potentially disrupt negotiation undetectably. The FREAK and Logjam attacks, discussed in detail in Sect. 6.9.5, are examples of downgrade attacks that depend on this cyclic dependency of authentication negotiation using a key derived from the negotiated ciphersuite.

If strong ciphersuites are used, defeating attacks like FREAK and Logjam, then authentication of the transcript (which includes the version negotiation) theoretically ensures the parties are not subject to a version downgrade attack. Unfortunately, this is not the case in practice. Many implementations implement a further version negotiation mechanism called *fallback*, sometimes called the *downgrade dance*. Since some flawed TLS server implementations respond with a failure message when confronted with a `ClientHello` containing versions higher than they support, TLS clients will retry the handshake with the next lower version, sidestepping the in-protocol version negotiation mechanism. Unfortunately, this downgrade is easy for an attacker to trigger: the failure message is not authenticated, so a man-in-the-middle attacker can spoof the failure message and trigger a downgrade. Originally, there was no link between the original request and the fallback request. In particular, there was no mechanism for the client to indicate to the server that it was attempting a downgraded handshake owing to a failure message, and thus the client and server would not detect the downgrade attack. In April 2015, the IETF standardised the TLS Fallback Signalling Ciphersuite Value (SCSV) [563]: when trying a lower version during a fallback, the client can send a flag indicating it is doing so (for engineering reasons, this flag is implemented as a special 'dummy' ciphersuite in the list of supported ciphersuites, hence the term 'signalling ciphersuite value'). A server that also supports SCSV can thus detect when a downgrade attack is occurring.

Cryptographic modelling of negotiation and downgrade resistance has been presented by Dowling and Stebila [258] and Bhargavan *et al.* [94].

6.11.2 Renegotiation Attack

Recall from Sect. 6.4.3 that renegotiation allows two parties who are using an existing TLS session to run a new handshake; upon completion of the new handshake, the session switches to the newly established encryption and authentication keys for



Attack 6.1: Ray and Dispensa's attack on TLS renegotiation

communication. In November 2009, Ray and Dispensa [623] described a man-in-the-middle attack that exploits how certain TLS-reliant applications—such as HTTP over TLS—process data across renegotiations. The attack is as shown in Attack 6.1.

The attacker Eve observes Alice attempting to establish a TLS session with Bob. Eve delays Alice's initial `ClientHello` and instead establishes her own TLS session with Bob, then transmits a message m_0 over that record layer. Then Eve passes Alice's initial `ClientHello` to Bob over the Eve–Bob record layer. Bob views this as a valid renegotiation and responds accordingly; Eve relays the handshake messages between Alice and Bob, which serve to establish a new record layer between Alice and Bob to which Eve has no access. Alice then transmits a message m_1 over the Alice–Bob record layer.

This is not, strictly speaking, an attack on TLS but on how some applications process TLS-protected data. It results from some applications, including many implementations of HTTPS [623] and SMTPS, concatenating m_0 and m_1 and treating them as coming from the same party in the same context. For example, if Eve sends the HTTP request

$$m_0 = \text{“GET /orderPizza?deliverTo=123-Fake-St} \hookleftarrow \text{X-Ignore-This: ”}$$

and Alice sends the HTTP request

$$m_1 = \text{“GET /orderPizza?deliverTo=456-Real-St} \hookleftarrow \text{Cookie: Account=111A2B”}$$

(where $\hookleftarrow$ denotes a new-line character), then the concatenated request (across multiple lines for readability) is

$$\begin{aligned} m_0 \| m_1 = \text{“GET /orderPizza?deliverTo=123-Fake-St} &\hookleftarrow \\ \text{X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St} &\hookleftarrow \\ \text{Cookie: Account=111A2B”} & \end{aligned}$$

The 'X-Ignore-This:' prefix is an invalid HTTP header. Since this header, without a new-line character, is concatenated with the first line of Alice's request, Bob's application receives a full HTTP header with an unknown header name, so this line is ignored. However, the following line, Alice's account cookie, is still processed. Eve is able to have the pizza delivered to herself but paid for by Alice's account.

It should be noted that Ray and Dispensa's attack works for both the server-only authentication and the mutual authentication modes of TLS: the use of client certificates does not in general prevent the attack.

The immediate recommendation due to this attack was to disable renegotiation except in cases where it was essential. The IETF TLS working group developed RFC 5746 [629] to provide countermeasures to this attack, with the goal of applicability to all versions of TLS and SSL v3. Two countermeasures were standardised:

Renegotiation Information Extension (RIE). This countermeasure essentially provides handshake recognition, confirming that both parties have the same view of the previous handshake when renegotiating. With this countermeasure, each client or server always includes a renegotiation information extension in its respective `ClientHello` or `ServerHello` message. This extension contains one of three values. If the party is not renegotiating, then it includes a fixed 'empty' string, which denotes that the party supports and understands the renegotiation extension, but that party is in fact not renegotiating. If the party is renegotiating, then it includes the handshake/key confirmation value from the previous handshake: the client sends the previous client verification data value while the server sends the concatenation of the previous client verification data and server verification data values. Intuitively, by including the verification data from the previous handshake, the parties can be assured that they have the same view of the previous handshake, and thus the attack in Fig. 6.1 is avoided.

Signalling Ciphersuite Value (SCSV). This countermeasure was designed to avoid interoperability problems with TLS 1.0 and SSL v3 implementations that did not gracefully ignore extension data at the end of `ClientHello` and `ServerHello` messages. Here, the client puts an additional value in its initial handshake—an extra, fixed, distinguished ciphersuite value (byte codes `0x00`, `0xFF`) included in its ciphersuite list—to indicate that it knows how to securely renegotiate. Old servers will ignore this extra value; new servers will recognise that the client supports secure renegotiation, and the server will use the RIE in the remainder of the session. In other words, the only difference between SCSV and RIE is in the `ClientHello` message of the initial handshake: with RIE, the client sends an empty extension, whereas with SCSV the client sends a distinguished value in the list of supported ciphersuites.

These countermeasures were quickly implemented in most TLS libraries, web browsers, and web servers. Giesen *et al.* [304] extended the ACCE model presented in Sect. 2.8.3 to formalise the notion of renegotiation security, and proved that these countermeasures provided renegotiation security for TLS.

6.11.3 Compression-Related Attacks: CRIME, BREACH

As described in Sect. 6.4.1, the TLS record layer can optionally perform compression of all application data. The *Compression Ratio Info-leak Made Easy* (CRIME) attack, presented by Rizzo and Duong in 2012 [632] makes use of a side channel based on the length of the compressed plaintext, first identified by Kelsey in 2002 [422], to extract secret values such as HTTP cookies from encrypted application data.

In 2002, Kelsey [422] described at a general level the various side channels that exist when plaintext is compressed prior to encryption; in particular, the length of the compressed plaintext compared with the original plaintext acts as a side channel, leaking some information about the amount of redundancy in the plaintext. One of the most powerful attacks described by Kelsey was an adaptive chosen input attack. Suppose the adversary is given access to an oracle that behaves as follows. When the adversary inputs a message x, the oracle outputs

$$\mathcal{O}_{a,b,S}(x) = \mathsf{Enc}(\mathsf{Comp}(a\|x\|b\|S)),$$

where Enc denotes encryption, Comp denotes compression, a and b are fixed public strings, and S is the secret the adversary is trying to recover. The basic idea of the attack is that, when the attacker's input x overlaps with S, the compressed string will be slightly shorter than when x does not overlap with S. The attacker works adaptively character by character.

Rizzo and Duong [632] showed how to use Kelsey's attack against TLS compression in the context of HTTP. On the web, we have a web browser making HTTP GET requests to an HTTPS website. In addition to the URL, the browser will send a header containing the authentication cookie; all of this is encrypted using TLS. For example, to visit `https://server.com/index.html`, the browser might send

$$\mathsf{Enc}(\mathsf{Comp}(\text{“}\texttt{GET /index.html}\hookleftarrow\texttt{Cookie: secret=31415926}\text{”})).$$

If the attacker can cause the user to visit arbitrary URLs on the target website `server.com`, then the browser acts as an oracle in Kelsey's adaptive chosen input attack above. In particular, if the attacker can cause the browser to visit the URL `https://server.com/`*url*, then the browser will send

$$\mathsf{Enc}(\mathsf{Comp}(\text{“}\texttt{GET /}\mathit{url}\hookleftarrow\texttt{Cookie: secret=31415926}\text{”})).$$

The attacker now iterates through different values of *url* (`secret=1`, `secret=2`, ...):

$$\mathsf{Enc}(\mathsf{Comp}(\text{“}\texttt{GET /secret=1}\hookleftarrow\texttt{Cookie: secret=31415926}\text{”})),$$
$$\mathsf{Enc}(\mathsf{Comp}(\text{“}\texttt{GET /secret=2}\hookleftarrow\texttt{Cookie: secret=31415926}\text{”})),$$
$$\mathsf{Enc}(\mathsf{Comp}(\text{“}\texttt{GET /secret=3}\hookleftarrow\texttt{Cookie: secret=31415926}\text{”})).$$

The third value will compress slightly more than the others, because of the greater overlap between `secret=3` and the cookie `secret=31415926`. The attacker

now knows that the first character of the secret is `3`. The attacker now adaptively queries for the next character:

Enc(Comp("`GET /secret=31↩Cookie: secret=31415926`"))
Enc(Comp("`GET /secret=32↩Cookie: secret=31415926`"))

and so on. The attacker can rapidly recover the secret. In particular, assuming no noise in the repeated requests, for a secret of n characters chosen from a c-character alphabet, the attacker can recover the secret with at most nc adaptive queries. Since the compressed plaintext is encrypted, some care is required when block encryption schemes are used to ensure that different-length compressed plaintexts lead to different-length ciphertexts, but this is possible using padding like in the BEAST attack.

The CRIME attack was quite effective: it worked regardless of the cipher used (AES or RC4) and only required the victim to visit once a URL controlled by the attacker, although it required the attacker be able to observe the size of the responses transmitted over the network. Both the Chrome and the Firefox browsers were vulnerable (since they supported TLS compression), but Internet Explorer was not (since it did not support compression). The attack was also effective against the SPDY protocol [86, 323], a modification of the HTTP protocol to improve performance. The only effective countermeasure against CRIME is to disable TLS compression. The TIME attack [67] extends the CRIME attack by using a timing side channel via JavaScript, relieving the attacker from the need to observe the responses as they are transmitted over the network. The HEIST attack [720] is another variant that removes the need for a man-in-the-middle network observer.

In 2013, Prado *et al.* [620] announced the BREACH (Browser Reconnaissance and Exfiltration via Aadaptive Compression of Hypertext) attack, which used a similar adaptive chosen input attack, this time against HTTP compression, allowing an attacker to recover secrets in the HTTP body (such as anti-cross-site-request-forgery tokens). The BREACH attack works independently of TLS compression. Despite the attack, HTTP compression remains widespread.

6.11.4 Termination Attack

TLS includes an alert protocol which allows a party to signal a communication error to its peer, or to signal that it is terminating the connection. Each party is required to send a `close_notify` alert immediately before closing the write side of its connection, and the receiving party should respond with a `close_notify` alert and then terminate the connection. This functionality was introduced in SSL v3. Originally, a connection that had been terminated using `close_notify` could not be resumed using session resumption, but this behaviour was permitted starting in TLS 1.1.

The purpose of the `close_notify` alert is to ensure that each party's application agrees on the data sent and received, and in particular to avoid *truncation attacks*, in which an attacker drops some data as well as the `close_notify` alert. Unfortunately, several attacks have been identified involving how data is passed from SSL/TLS implementations to relying applications in the context of termination/truncation.

Berbecaru and Lioy [88] showed how various SSL v3 and TLS 1.0 implementations deal inconsistently with truncated data. For example, consider a web server that is transmitting an image using HTTPS to a web browser (Mozilla Firefox v1.5.0.7 in their tests), and a man-in-the-middle who cancels various parts of the transmission. They observed that if the MITM cancelled part of the data record encrypting the image, but allowed subsequent records including the `close_notify` to be transmitted, the browser would report it could not display the image because of errors, the expected behaviour. However, if the MITM cancelled part of the data record encrypting the image and *all* subsequent records, including the `close_notify` alert, the web browser would display the truncated image to the user. The SSL library did not transmit a warning to the application data that the received data was incomplete, and correspondingly the web browser did not abort in error, despite receiving an HTTP response that was shorter than indicated in the HTTP `Content-Length` header.

Smyth and Pironti [684] showed how to exploit this type of flaw in more complex modern web applications. For example, a user browsing a variety of Google properties (Gmail, Search, etc.) has sessions established with each service, even though a single sign-on is used. When the user clicks 'Log out', the user's session must be logged out with each service. Smyth and Pironti observed that this is sometimes achieved using parallel logout requests to each service, followed by display of a success page. In their study, Smyth and Pironti showed how an attacker who can truncate/drop selected TLS packets can cause some sessions to remain active even after the user has clicked log out and received the success page.

6.11.5 Triple Handshake Attack

As noted in Sect. 6.4, TLS allows parties to create new sessions from existing ones. *Session resumption* uses an abbreviated handshake to start a new session from a prior session. *Renegotiation* performs a new, full handshake inside an existing session, allowing parties to change ciphersuites or authentication credentials, or obtain fresh keying material.

Bhargavan *et al.* [96] discovered a *triple handshake attack*, which allows an attacker to confuse a client into thinking it is connected to a different server; the attack takes advantage of flaws in session resumption and renegotiation. The attack proceeds in three steps. In the first step of the attack, the client establishes a TLS session using RSA key transport with the attacker; the attacker establishes a session with the target server using the same nonces and premaster secret. In the second step, the client uses session resumption with the attacker to resume its session; the attacker relays the session resumption handshake messages to the target server. Since the relevant cryptographic values are the same (specifically, the premaster secret), the session resumption succeeds, and the client now has a (resumed) session, but with the target server. The attacker injects a message to the target server, which it can do because it still knows the session key of the resumed session. In the final step, a renegotiation takes place between the client and the server, the result of which is a renegotiated session between the client and the target server which the adversary is

not able to access. However, some applications concatenate the data sent *by the attacker before the final step* and the data sent *by the client after the final step* as coming from the same party (this is similar to the renegotiation attack in Sect. 6.11.2).

The primary reason that the triple handshake attack is successful is that session resumption depends only on the previous session's master secret, rather than the whole handshake. The principal proposed countermeasure is to bind the master secret to the full handshake, rather than just the nonces and premaster secret, and was standardised in RFC 7627 [102].

6.12 Attacks: Implementations

Software libraries implementing TLS, like other pieces of software, unfortunately contain bugs which often result in vulnerabilities. While we do not aim to provide an extensive overview of software-related TLS vulnerabilities, we will briefly mention a few notable ones. These subsections examine attacks resulting from bad random number generators and bad certificate validation code in software libraries.

Software vulnerabilities, by their nature, usually affect only a particular implementation. OpenSSL, being a widely used open source TLS implementation, is a frequent target of vulnerability research, but all TLS implementations have had flaws.

6.12.1 Side Channel Attacks

Several flaws in OpenSSL have been identified related to side channels in processor microarchitectures. Percival [608] successfully recovered an RSA private key from OpenSSL owing to a cache-timing side channel in its implementation of the Chinese Remainder Theorem. Acıiçmez *et al.* [20] identified a further vulnerability in RSA private key operations in OpenSSL due to a simple branch prediction analysis attack. Yarom and Benger [756] subsequently identified a cache-timing side channel that also allowed for recovery of signing keys when ECDSA over binary fields was used.

Brumley and Tuveri [162] discovered a timing side channel in OpenSSL's implementation of elliptic curve arithmetic over binary fields. The exploit allows an attacker to recover the ECDSA private key from a TLS server; the authors demonstrated the exploit over a loopback network interface.

Brumley *et al.* [161] discovered how to exploit a fault/error side-channel in OpenSSL's implementation of elliptic curve point multiplication for prime fields. Using an adaptive attack against repeated use of a NIST-P256 prime field elliptic curve private key, the attack recovers the private key in just 663 queries. The attack requires reuse of the private key, which is the case with static ECDH TLS ciphersuites and with ephemeral ECDH TLS ciphersuites where the server re-uses the ephemeral key as an optimisation technique. While reuse of ephemeral keys is optional, it is apparently the default behaviour of OpenSSL.

Most of the above attacks led to updated versions of OpenSSL and reports in the Common Vulnerabilities and Exposures (CVE) database. Although supported by OpenSSL, few applications actually make use of elliptic curves over binary fields,

and, in particular, no commercial certificate authority to date has issued X.509 certificates for ECDSA public keys over binary fields, so the practical impact of the binary ECDSA vulnerabilities was minimal.

6.12.2 TLS-Specific Implementation Flaws

Researchers at Google and Codenomicon [220] independently identified a buffer over-read vulnerability in OpenSSL's implementation of the TLS heartbeat extension [661], allowing an attacker to read up to 64 KB of additional memory from the server, memory which could potentially contain private keys, passwords, or other sensitive information. The bug was given the moniker 'Heartbleed' and even got its own website and logo (`http://heartbleed.com`). Unlike the other attacks above, this bug requires no cryptographic expertise to discover or exploit. Turnkey exploits were readily developed and deployed, and researchers were successfully able to remotely recover signing private keys from test servers. Using the Zmap tool [262], researchers identified that approximately 17% (around half a million) of TLS-enabled websites were vulnerable at the time of discovery; 2 months later, 1.5% of the 800,000 most popular TLS-enabled websites remained vulnerable [484]. The Heartbleed vulnerability attracted widespread public attention.

Another specific class of implementation errors is related to processing messages in the correct order. Kikuchi [483] discovered a flaw in OpenSSL where it would accept `ChangeCipherSpec` messages at any point in time, rather than only immediately prior to the transmission of the `Finished` messages. If the `ChangeCipherSpec` message is accepted earlier, the server will use an empty master secret to compute the session keys. It is unclear how to exploit this weakness to defeat cryptographic protections, but it is still an implementation flaw. In the same vein, Beurdouche *et al.* [92] identified several more state machine attacks (which they named 'SMACK'). They discovered that many TLS implementations do not correctly implement the TLS state machine: the various TLS versions, extensions, authentication modes, and key exchange methods may lead to different sequences of messages and processing, and failure to use the correct sequence in every context can lead to vulnerabilities.

6.12.3 Certificate Validation

A notable class of implementation errors is related to certificate validation, either in TLS implementations directly or in applications or higher-level libraries that make use of TLS implementations.

OpenSSL releases prior to 0.9.8j failed to correctly validate some types of malformed signatures in a certificate chain, allowing an attacker to bypass validation [595]. Apple's implementation of SSL/TLS in its Secure Transport library for Mac OS X and iOS had a bug in which certificate validation always succeeded owing to a spurious `goto fail` statement in the code, giving the bug its name [476].

Georgiev *et al.* [302] reported SSL certificate validation errors in a variety of non-browser applications and libraries, typically related to incorrect parameters used in libraries. For example, Amazon provided third-party PHP developers with

a Flexible Payments Service library allowing them to receive payments. The library made use of the command-line program cURL for HTTPS connections. The library attempted to enable hostname verification in cURL by setting an option `CURLOPT_SSL_VERIFYHOST = true`. However, the correct value should have been 2 rather than `true`, and the effect was that certificate validation was disabled, allowing an attacker to carry out an impersonation attack. Similar mistakes involving hostname verification, certificate revocation, and certificate chain validation were observed by these authors in a variety of libraries using a variety of programming languages and applications.

In 2014, Brubaker *et al.* [160] reported on the results of testing the certificate validation logic of a variety of client and server TLS implementation using *frankencerts*, described as "synthetic certificates that randomly mutated from parts of real certificates and thus include unusual combinations of extensions and constraints". When one implementation accepted a frankencert while another did not, this identified a discrepancy in certificate validation logic meriting further investigation. Overall, Brubaker *et al.* found 208 discrepancies between popular implementations, many of which led to significant vulnerabilities, such as the ability to act as a rogue CA, to create certificates not signed by a trusted CA, to accept certificates not intended for authentication, or to trigger user interface indicators that provide inaccurate warnings.

6.12.4 Bad Random Number Generators

The security of any TLS implementation depends crucially on the quality of the random number generator and seed used for the picking of long-term private keys and per-session private keys. Several prominent SSL/TLS implementations have had flaws in how their random number generators are seeded, leading to exploitable attacks.

In 1996, Goldberg and Wagner [309] determined, by reverse engineering, that the pseudo-random number generator (PRNG) in one of the earliest web browsers, Netscape Navigator v1, was poorly seeded. They found that the seed to the PRNG in Netscape was constructed from the process ID (pid), parent process ID (ppid), and the time in seconds and microseconds. The pid and ppid can be easily determined by another user running on the same system; for a remote attacker, there are at most 2^{27} possible pid/ppid value pairs, and often the pid and ppid values are considerably smaller. A local or network attacker can determine the time in seconds based on network observations, leaving at most 1,000,000 (approximately 2^{20}) values to test for the microseconds. Thus, for an attacker, there are at most 47 bits of entropy in the seed of the PRNG and hence at most 47 bits of randomness in the secret keys. Netscape Navigator version 1.22 and higher improved the seeding of the PRNG to avoid this problem, but the flaw still serves as a significant real-world example of the challenges of random number generation.

In 2008, Bello [710] discovered a flaw in the random number generator used by the OpenSSL library package distributed on the Debian operating system. The flaw was introduced in 2006 when Debian maintainers commented out a couple of lines

of code in OpenSSL when updating the package for Debian; the flaw was not present in the original OpenSSL source code. The effect of the code change was to remove almost all of the entropy added to the PRNG seed, leaving the sole randomness as the process ID. Taking into account the effects of different processor architectures, there are just 294,912 keys that can possibly be generated by the vulnerable versions of Debian OpenSSL [761]. Because the OpenSSL library is used in many applications, this flaw affected not only TLS, but also SSH and OpenVPN (for virtual private networking). In a survey of 50,000 SSL servers [761] on the day of vulnerability disclosure in 2009, approximately 700 (1.4%) were using a weak key; 6 months later, 30% of those Debian weak keys were still in use. In a later, large-scale survey of 12.8 million TLS servers and 10.2 million SSH servers published in 2012 [354], Heninger *et al.* found a very small percentage (0.03%) of TLS servers using Debian weak keys, though a non-trivial percentage (0.52%) of SSH servers were still using Debian weak keys.

6.13 Attacks: Other

Finally, there are a few other attacks which do not fit into the categories explored above.

6.13.1 Application-Level Protocols

The most common use of TLS is in combination with the Hypertext Transport Protocol (HTTP). Because much web traffic still runs over HTTP, rather than HTTPS, security vulnerabilities can arise owing to how HTTP and HTTPS interact.

One common flow on the web is for users to initially browse a website using plaintext (HTTP) communication. At some point, such as when they click 'login' or try to purchase something, they are redirected to a secure site, using an HTTPS address specified in the web page's source code. In 2009, Marlinspike [524] demonstrated an HTTPS 'stripping' attack using a new tool called 'sslstrip'. A man-in-the-middle attacker observing an unencrypted HTTP connection alters the source code of responses from the server to replace HTTPS URLs with HTTP URLs: thus, when the user clicks the 'login' button and enters her password, the password is transmitted unencrypted.

To prevent SSL stripping, the HTTP Strict Transport Security (HSTS) mechanism was created [360]. HSTS allows a web server to tell a client to automatically use HTTPS for all pages on a certain domain for a certain period of time, even if the URL is given as HTTP. This is a trust-on-first-use mechanism, so it requires that the client make at least one non-adversary-controlled connection to the server. Despite HSTS, HTTPS stripping can still be achieved by an attacker who rewrites links in plaintext pages to alternative domain names that are not covered by an HSTS policy [523].

HTTP servers that serve multiple sites—such as content distribution networks—have also been vulnerable to 'virtual host confusion' attacks [239]. These attacks

depend on a complex interaction between TLS and application characteristics, so we omit the details; the root cause of the attacks is that servers have many different 'identities' at different levels of the network stack (IP, TCP, TLS, HTTP), and they do not always match the identity that is cryptographically authenticated by TLS.

Other application-level protocols also use TLS, but in a different way from HTTPS. Whereas HTTPS runs on a separate TCP port from HTTP and immediately begins with a TLS handshake, other applications, such as IMAP, POP3, XMPP, NNTP, IRC, and FTP, run the secure and plaintext versions on the same port: the parties start with a plaintext communication, then use a protocol-specific command, often called 'STARTTLS', to initiate a TLS handshake for secure communication. Venema and Orlando [722] discovered vulnerabilities in many applications' use of STARTTLS that allowed an attacker to inject commands in the plaintext portion of the protocol that would be executed during the encrypted portion of the protocol.

6.13.2 Certificate Authority Breaches and Related Flaws

Since authentication in TLS uses certificates, users' security can be undermined by mistakes or by attacks on certificate authorities. In some applications, TLS clients may be configured to accept certificates from a single CA or a very small number of CAs (see Sect. 6.12.3 for cases when this should—but does not—happen).

In many settings, however, TLS clients are configured to accept certificates from many CAs. On the public web, mainstream web browsers ship with 150–200 root CA certificates that are trusted by default, issued by some 80+ organizations. Some of these root CAs issue web server certificates directly or via an intermediate CA run by the same company, but many root CAs also issue certificates for subordinate CAs run by other organizations, who then also issue web server certificates directly or via intermediate CAs. For example, at the time of writing, an HTTPS connection to `eprint.iacr.org` returned a certificate issued by 'RapidSSL SHA256 CA - G3', whose certificate was issued by 'GeoTrust Global CA'. The subordinate CA, RapidSSL, is a separate organization from the root CA, GeoTrust.

An Internet-wide scan by the Electronic Frontier Foundation's SSL Observatory project in 2010 found more than 650 organizations that were trusted by default by major browsers as CAs [266]. In general, any one of these organizations has the ability to issue a browser-trusted certificate for any domain name, without geographic or other restrictions. As a result, there are many potential points of failure in the CA ecosystem. The CA/Browser Forum is a consortium of CAs and browser vendors who work together to agree on guidelines for the operation of CAs and the criteria for root CA inclusion in browsers and operating systems.

There have been several incidents involving CAs mistakenly issuing certificates. Among the most dramatic was the DigiNotar breach of 2011 [282]. DigiNotar was a Dutch CA that issued certificates under two main CAs, one for public websites ('DigiNotar Root CA') and one for the Dutch government. The certificate for 'DigiNotar Root CA' was built in to all major browsers. In July 2011, DigiNotar mistakenly issued a wildcard certificate for `*.google.com`. This certificate was used in Iran in August 2011 to conduct man-in-the-middle attacks against users of Google

services. Subsequently, it was discovered that DigiNotar had issued more than 500 other fraudulent certificates, including ones for Yahoo!, Mozilla, WordPress, and the Tor project. All major browser vendors issued updates that removed 'DigiNotar Root CA' from their browser's list of trusted CAs. DigiNotar declared bankruptcy in September 2011.

The DigiNotar breach was detected owing to Google Chrome's use of *certificate pinning*. At the time, Google Chrome had hard-coded the fact that Google services would use only certificates from certain CAs, so Chrome users were in fact protected against the man-in-the-middle attack on Google services, while other users would not have been. This ad hoc approach has been standardised as HTTP Public Key Pinning [272], which allows web servers to specify that future connections to that same domain will be secured using a certain certificate or CA. This limits the ability of an attacker to make use of a fraudulently issued certificate in the event of a CA breach, though HTTP Public Key Pinning is a trust-on-first-use mechanism, meaning it only provides security if the *initial* connection between the client and the server does not involve a fraudulently issued certificate.

System administrators and local software can also modify the browser or operating system's list of trusted root CAs. This can be done for legitimate reasons, to support a company's internal PKI or to enable an HTTPS proxy to inspect downloads for viruses, but can also be done maliciously. Some malware is known to do this to enable further attacks. In 2015, it was discovered that Lenovo computers were shipping with a piece of advertising software called Superfish which added a root CA certificate to computers so that it could act as an HTTP/HTTPS proxy and inject advertising into web pages. Unfortunately, the software used the same root CA certificate on all computers, and, moreover, the corresponding private key was also included in the software, so anyone who could reverse engineer the software could recover the root CA key and impersonate any web server to any user with the Superfish software installed [40]. Interestingly, public key pinning would not protect against this attack, as browsers are configured to allow locally installed certificates to override pins.

6.14 TLS Version 1.3

In 2014, the Internet Engineering Task Force began a multi-year process to develop the next version of TLS, now called TLS version 1.3. In August 2018, TLS 1.3 was standardised in RFC 8446 [626]. There were several motivations for the development of TLS 1.3:

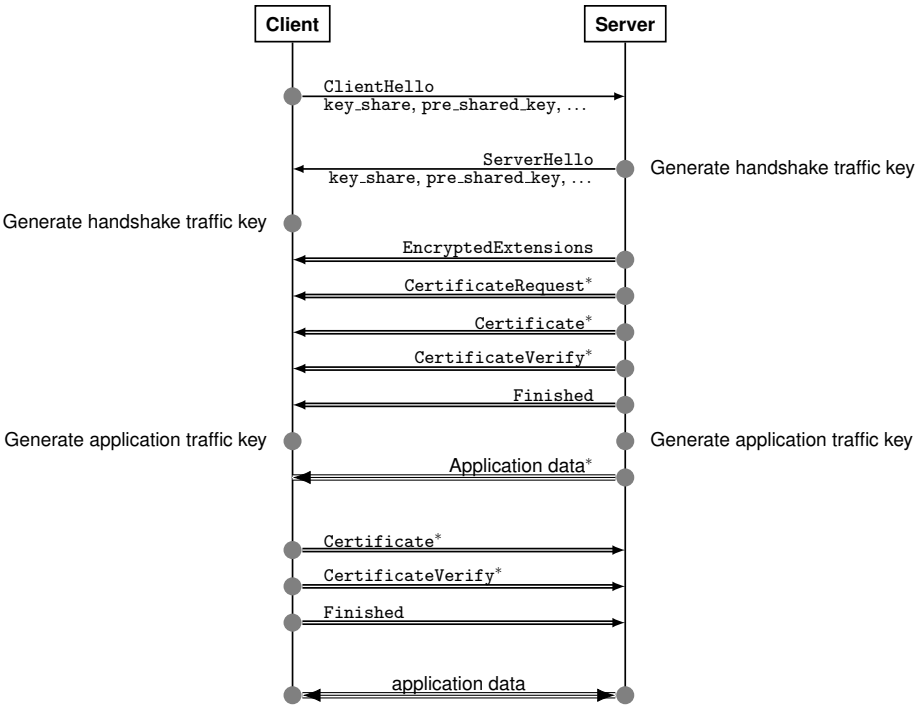
1. to deprecate old cryptographic algorithms and switch to modern algorithms and modes of operation;
2. to encrypt parts of the handshake to enhance privacy, in part as a response to concerns about mass surveillance;
3. to reduce the latency of connection establishment by providing modes with fewer round trips;

4. and to make general changes to cryptographic and other operations of the protocol, including simplifying protocol logic.

To achieve the first goal, many cryptographic algorithms that had been in use in TLS 1.2 and earlier versions were removed from TLS 1.3. The symmetric encryption and integrity algorithms in the record layer protocol had been the cause of several vulnerabilities; TLS 1.3 removed the MAC-then-encode-then-encrypt mode of operation, and focused on provably secure authenticated encryption schemes, mandating AES in Galois/counter mode, and optionally ChaCha20 with Poly1305. In terms of public key cryptography, static Diffie–Hellman ciphersuites were removed, as was key exchange based on RSA key transport; RSA-based digital signatures within TLS must use the PSS algorithm, rather than PKCS #1 v1.5 (although certificates may still be signed using PKCS #1 v1.5). All asymmetric key exchange algorithms in TLS 1.3 are based on ephemeral Diffie–Hellman to provide forward secrecy. New elliptic curve algorithms are available, including the use of Curve25519 in key exchange (X25519) and signatures (Ed25519), and some finite-field and elliptic curve groups have been removed.

To achieve the second goal of enhancing privacy, the flow of messages in the handshake algorithm was substantially altered. Most notably, unauthenticated ephemeral key exchange happens in the first two flows of the protocol, after which the parties establish a temporary 'handshake traffic key' which is used to encrypt the remainder of the handshake, including the transmission of certificates, before finally switching to the 'application traffic key' which is used to encrypt application data in the record layer protocol. Protocol 6.3 shows the handshake protocol for a full handshake in TLS 1.3. The client sends one or more ephemeral keys in the `key_share` extension of the `ClientHello` message. Since the parties have at this point not yet negotiated algorithms, the client is simply guessing which ephemeral key exchange algorithms might be acceptable to the server; if the server does not support any of the algorithms offered by the client, the server can send a `HelloRetryRequest` message with a list of algorithms to ask the client to retry.

To reduce latency of connection establishment, a new zero-round-trip (0-RTT) mode of operation is available when pre-shared keys are being used for session establishment (which includes session resumption). This allows the client to send application data immediately on its first flow to the server, rather than having to wait until receiving a response from the server. In 0-RTT mode, the client derives an early application traffic key from the pre-shared key, and uses this to encrypt its first application flow, which it sends along with the `ClientHello`. The client and server can still optionally negotiate a forward secure ephemeral shared secret in this mode, but that ephemeral secret will only apply to subsequent application data flows, not the first one from the client to the server. Protocol 6.4 shows the 0-RTT handshake mode. One concern regarding 0-RTT mode is that the first client-to-server flow can be replayed; the TLS 1.3 document discusses some potential mitigations for servers to prevent replay attacks, such as keeping state and using application-level protections. TLS 1.3 has no explicit session resumption mode; instead, a 'resumption master se-



* denotes messages that may not be present in all ciphersuites.
Single lines denote plaintext flows; double lines denote encrypted flows using the handshake traffic key; triple lines denote encrypted flows using the application traffic key.

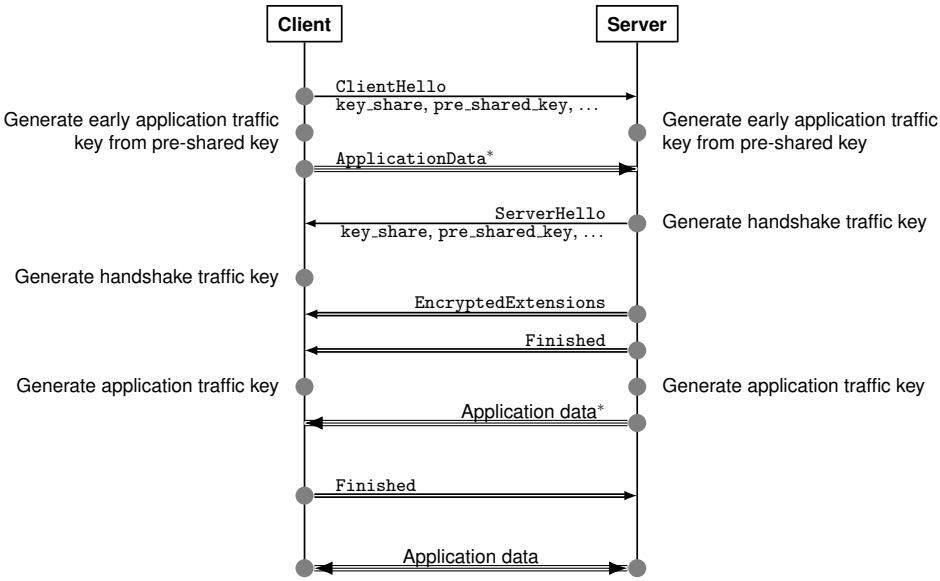
Protocol 6.3: TLS 1.3 handshake protocol – full handshake

cret' can be exported from any session, and this value can be used as a pre-shared key in a subsequent PSK or 0-RTT handshake.

Finally, many other changes have been made throughout the protocol, some cryptographic, some not. The key derivation function is now HKDF [454], and there is a new key derivation schedule that incorporates each new piece of keying material and generates all necessary traffic keys. The handshake state machine has been reorganized, and the `ChangeCipherSpec` messages have been removed. Digital signatures in the `CertificateVerify` message now cover the entire handshake transcript. Ciphersuite negotiation has been made modular: each cryptographic component (authenticated encryption algorithm, digital signature algorithm, key exchange algorithm) is negotiated separately.

All major browser and library vendors have committed to support TLS 1.3, and as of December 2018, support is available in the Chrome and Firefox browsers, and the OpenSSL and NSS libraries.

One notable aspect of the development of TLS 1.3 has been the early and ongoing involvement of the academic community. The core cryptographic design of TLS 1.3



* denotes messages that may not be present in all ciphersuites.
Single lines denote plaintext flows; double lines denote encrypted flows using the handshake traffic key; triple lines denote encrypted flows using the early or regular application traffic key.

Protocol 6.4: TLS 1.3 handshake protocol – pre-shared key handshake with early application data ('zero-round-trip')

was heavily influenced by the OPTLS protocol of Krawczyk and Wee [457]. There were a variety of academic papers published analyzing and commenting on various aspects of drafts of TLS 1.3, including results using provable security [257, 457, 487], constructive cryptography [444], and formal methods [230], and the miTLS team's efforts of using formal methods combined with a verified implementation [92, 95]. A workshop called 'TLS 1.3: Ready or Not? (TRON)' was held in February 2016 that brought together academic researchers, industry professionals, and members of the IETF TLS working group to exchange knowledge. In late 2016, Paterson and van der Merwe [604] published an account of the TLS 1.3 standardization effort.

7

Identity-Based Key Agreement

7.1 Introduction

Identity-based public key cryptography was first proposed by Shamir in 1984 [665]. The idea is to avoid the need for public key certificates by making the public key publicly computable from the identification information of the owner. The identification information can include any desired fields such as real name, physical description or identification numbers. Identity-based cryptography avoids the difficulty of having to distribute public keys and thus avoids the need for a public key infrastructure, although parties still need to obtain and manage private keys.

In 1984 Shamir proposed an algorithm for identity-based signatures but was unable to obtain an identity-based encryption algorithm. In 1987 Okamoto [589, 590] published the first identity-based key agreement protocol, using the same format of key pairs as in Shamir's original identity-based signatures. For over a decade there was limited activity in the area of identity-based cryptography, until in 2000 the first practical identity-based encryption schemes were proposed [123, 647]. These schemes exploit the bilinearity property of elliptic curve pairings. Following this discovery, there was an explosion of interest in identity-based cryptography based on pairings. This included a variety of different cryptographic primitives and protocols, including scores of key agreement protocols.

There are many similarities between identity-based key agreement and key agreement using standard public key cryptography. Indeed, many key exchange protocols use generic building blocks, such as encryption or signatures, and can be instantiated with either identity-based or conventional public key versions of these building blocks. Essentially, the aim in designing a good identity-based key agreement protocol is to achieve all the properties of the best conventional key agreement protocols but without the need for certified public keys, and at the same time trying to maximise efficiency.

We continue to use the notation ID_I to denote the identifying string of entity I. In many identity-based protocols it is not acceptable for an arbitrary string, which may be chosen by the adversary, to be used directly as input to the private key generation process. This can allow construction of new private keys corresponding to algebraic

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_7

combinations of other identities. Therefore ID_I is typically the output of a one-way hash function applied to the identifying data.

7.1.1 Security Model for Identity-Based Cryptosystems

In an identity-based cryptosystem, the public key of any entity is determined by that entity's identity string and any public parameters of the system. This is a very attractive way of obtaining keying material, but unfortunately there is a significant drawback. No principal can be allowed to generate its own private key. If this were possible then *any* entity could do the same; this would mean that any entity could masquerade as any other. Therefore, in all identity-based schemes the private key of each principal must be generated by a trusted third party. Such a degree of trust may not always be reasonable, although it is probably acceptable in a corporate environment. This trusted party is usually known as the *key generation centre* (KGC).

In order to generate private keys, the KGC must have some secret information that depends on the public parameters of the system. (All parties must obtain authentic copies of the public parameters.) This secret information is known as the *master secret* of the identity-based cryptosystem. We usually assume that the public system parameters and the master secret are generated by the KGC when the system is initialised. The private keys for an entity I can be generated as needed by the KGC as a function of the identity information ID_I and the master secret. An interesting property of identity-based cryptosystems is that the public key can be used before the private key has even been generated. In identity-based key agreement this can lead to the situation where one party A possesses a key that is implicitly shared with a partner B who is unable to compute that shared key until B contacts the KGC to obtain its private key.

Generation of private keys is often known as *key extraction* in the literature. In formal security models, we normally expect the adversary to have the ability to extract private keys for any parties which are not the target of its attack. This may seem a strong assumption but it reflects the reality that the adversary may be able to obtain private keys for many different entities.

Desirable security properties of identity-based key agreement include all those discussed earlier for key agreement based on certified public keys. An additional property that is desirable is an extended version of forward secrecy with respect to the KGC. Although knowledge of the master secret will allow the adversary to masquerade as any entity, there is no reason that it should also allow previously used session keys to become compromised. Since the KGC private key can be used to obtain any user private key, this is arguably even more important than forward secrecy for conventional key agreement.

Definition 34. *An identity-based key agreement protocol provides* KGC forward secrecy *if compromise of the KGC's master secret does not compromise the session keys established in previous protocol runs.*

The necessity of a KGC to generate user keys is a limitation of identity-based cryptography. It is often referred to as the *escrow* problem, since the KGC can at

any time generate a spare user private key. Ways to mitigate the escrow problem have been suggested in the literature. These generally involve a compromise between truly identity-based schemes and ordinary public key schemes using certificates. In Sect. 7.5.2, we will look at Girault's classification and scheme for dealing with the problem. Another possibility, discussed in Sect. 7.5.3, is to use *certificateless* cryptography, for which the KGC generates only a part of the user private key.

7.1.2 Elliptic Curve Pairings

Pairings are functions which take as input two elliptic curve points and *pair* them to form an output in a subgroup of a finite field. Different pairing functions have been used for pairing-based cryptography, but they all have the critical feature of being *bilinear*, that is they are linear in both of the input components.

The first cryptographic application of elliptic curve pairings was in cryptanalysis of elliptic curve cryptosystems. Menezes, Okamoto and Vanstone [544] used the Weil pairing to map elliptic curve discrete logarithms into a finite field, which, for certain curve types, results in a much easier way of solving the problem. This is often called the *MOV attack* on elliptic curve cryptosystems. It was only later that it was realised that pairings can be used constructively to design cryptosystems with properties previously unavailable. The Weil pairing was modified by Boneh and Franklin [123] as a concrete construction for their identity-based encryption scheme. Later, other pairings have been proposed for cryptographic purposes, particularly the Tate pairing and variants thereof.

A full explanation of the construction of elliptic curve pairings is beyond the scope of this book and the reader is referred elsewhere for the mathematical details [221]. In general, the two points that are paired come from different elliptic curve groups, which we denote $\mathbb{G}_1$ and $\mathbb{G}_2$. The output of the pairing is a finite-field point from a subgroup which we denote $\mathbb{G}_T$, sometimes called the *target group*. There is potential for confusion about the notation because the early literature on identity-based cryptography used additive notation in groups $\mathbb{G}_1$ and $\mathbb{G}_2$ as is traditional for elliptic curve groups. However, more recently it has become normal to use multiplicative notation for all three groups so we adopt this convention in this chapter.

Using the multiplicative notation, we let $\mathbb{G}_1$ be a group of prime order q and $\mathbb{G}_2$ be a group of the same order q. We assume the existence of the pairing map $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. The mapping $\hat{e}$ must be efficiently computable and have the following properties.

Bilinear: for $w, x \in \mathbb{G}_1$ and $y, z \in \mathbb{G}_2$, both

$$\hat{e}(w, yz) = \hat{e}(w, y) \cdot \hat{e}(w, z) \quad \text{and} \quad \hat{e}(wx, z) = \hat{e}(w, z) \cdot \hat{e}(x, z).$$

Non-degenerate: for some elements $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, we have $\hat{e}(g, h) \neq 1$.

When $a \in \mathbb{Z}_q$ and $w \in \mathbb{G}_1$, we write w^a for exponentiation in $\mathbb{G}_2$ (traditionally called elliptic curve scalar multiplication). Owing to bilinearity, for any $w \in \mathbb{G}_1$, $y \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_q$ we have

$$\hat{e}(w^a, y^b) = \hat{e}(w, y)^{ab} = \hat{e}(w^{ab}, y) = \hat{e}(w, y^{ab}).$$

For some types of elliptic curves (specifically, supersingular curves), it is possible to assume that the two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same. Such pairings are often called *symmetric pairings*. However, such a choice restricts the efficiency of the resulting protocols, so in general it is preferable to avoid this assumption. The details of the properties of different pairings are complex for the non-specialist to appreciate, and so Galbraith *et al.* [289] summarised the important properties and classified pairing groups into three types. The relevant issues include:

- the availability of an efficient homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$;
- the possibility to hash efficiently into $\mathbb{G}_2$;
- the efficiency of exponentiation in each group;
- the size of element representation in each group.

Chen *et al.* [192] have considered in detail the effect of different pairing types on the efficiency of a wide variety of identity-based key agreement schemes. They also introduced a fourth pairing type in addition to the three considered by Galbraith *et al.* [289]. Table 7.1 summarises the defining properties of the different pairing types. Note that although Type 1 looks the most favourable in this table, its efficiency is limited, especially at higher security levels.

Table 7.1: Pairing types of Galbraith *et al.* [289] and Chen *et al.* [192]

	Symmetric	Homomorphism $\mathbb{G}_2 \to \mathbb{G}_1$	Hash to $\mathbb{G}_2$
Type 1	✓	✓	✓
Type 2	✘	✓	✘
Type 3	✘	✘	✓
Type 4	✘	✓	✓

A pairing-based key agreement scheme usually combines long-term identity-based keys with ephemeral keys (both private and public). When setting up such a scheme, a decision has to be made regarding which group each key will lie in. Extraction of private keys from identities normally entails hashing, so this may not be possible for certain pairing types if long-term keys are to lie in $\mathbb{G}_2$. However, pairings require one input to be from $\mathbb{G}_1$ and the other from $\mathbb{G}_2$ so that, depending on the way that the values are combined, it may be necessary to make use of a homomorphism to take keys from $\mathbb{G}_2$ into corresponding values in $\mathbb{G}_1$. Again, this does not exist for all pairing types.

Making a precise comparison between protocols turns out to be very difficult, since it depends ultimately on the cost of operations on elliptic curves and finite fields, whose optimisation may depend on the details of the specific computing platform. Therefore, in this chapter we will limit ourselves to general observations about the relative efficiency of protocols. We will also explain most protocols using symmetric pairings for ease of exposition, but will also remark on the possibility of using

other pairing types. An exception is our treatment of the SOK protocol below, which we use to illustrate the effect of using asymmetric pairings.

For security of key agreement protocols in finite fields we typically need to assume that the Diffie–Hellman problem (and hence also the discrete logarithm problem) is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. In pairing-based key exchange a natural problem to base security upon is the bilinear version of the Diffie–Hellman problem.

Definition 35 (*Bilinear Diffie–Hellman (BDH) problem*). *Given $\mathbb{G}_1$, $\mathbb{G}_2$ and $\hat{e}$ as above, the BDH problem is to compute $\hat{e}(g_1,g_2)^{xyz} \in \mathbb{G}_T$ given $\langle g_1, g_2, g_1^x, g_2^y, g_1^z, g_2^z \rangle$ with $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $x,y,z \in \mathbb{Z}_q$.*

This computational assumption is just one of many which have been used in security proofs for different identity-based cryptographic primitives [144]. Sometimes it is possible to relate such assumptions to each other or to existing accepted assumptions. For example, the BDH problem is no harder to solve than solving the Diffie–Hellman problem either in $\mathbb{G}_1$ or in $\mathbb{G}_2$. However, as usual, we do not have any absolute guarantee of the difficulty of any of these problems.

7.1.3 Sakai–Ohgishi–Kasahara Protocol

The Sakai–Ohgishi–Kasahara (SOK) protocol [647] is a fundamental building block in many identity-based key agreement protocols. It is a *non-interactive protocol* and can be regarded as an identity-based analogue of static Diffie–Hellman using traditional public key certificates. In an identity-based infrastructure, it allows any two parties to establish a shared secret without exchange of any messages. The security of SOK relies on the difficulty of the BDH problem.

The SOK protocol makes use of a pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The master secret is a value $s \in \mathbb{Z}_q$ chosen randomly by the KGC. In order to extract private keys we need to hash identity strings onto points in $\mathbb{G}_1$ or $\mathbb{G}_2$, so we define two functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \{0,1\}^* \to \mathbb{G}_2$. Note that defining such an H_2 is not possible for all pairing types so we need to be careful about how to make the protocol work. We consider three variants.

- First, suppose that we are going to use a Type 1 pairing. This means that we can assume $\mathbb{G}_1 = \mathbb{G}_2$, and we only need to have one hash function, H_1. We denote the public keys of A and B as $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$. The private keys of parties A and B will then be the values $d_A = q_A^s$ and $d_B = q_B^s$, each of which can only be computed by the KGC, which is in possession of the master secret s. With the above parameters, any two principals A and B with identities ID_A, ID_B can efficiently calculate a shared secret as:

$$F_{AB} = \hat{e}(q_A, q_B)^s = \hat{e}(d_A, q_B) = \hat{e}(q_A, d_B).$$

  This variant of the SOK protocol only works with a Type 1 symmetric pairing, since any principal's public/private key pair needs to be defined in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

- Assume now that $\mathbb{G}_1 \neq \mathbb{G}_2$. In order to allow pairing between the keys of any pair of users, we can define the keys of all users to lie in $\mathbb{G}_2$ and use the homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ to move one key into $\mathbb{G}_1$ when the protocol is run. Now we denote the public keys of A and B as $q'_A = H_2(ID_A) \in \mathbb{G}_2$ and $q'_B = H_2(ID_B) \in \mathbb{G}_2$. The corresponding private keys are the values $d'_A = (q'_A)^s \in \mathbb{G}_2$ and $d'_B = (q'_B)^s \in \mathbb{G}_2$. We also need some way of deciding whether party A or party B's key should be moved into $\mathbb{G}_1$. One easy way of doing this is to use any natural ordering on the strings q'_A and q'_B and say that A's key will be moved into $\mathbb{G}_1$ if and only if $q'_A < q'_B$. With the above parameters, any two principals A and B with identities ID_A, ID_B can efficiently calculate a shared key as

$$F_{AB} = \hat{e}(\psi(q'_A), q'_B)^s = \hat{e}(\psi(d'_A), q'_B) = \hat{e}(\psi(q'_A), d'_B).$$

  This SOK variant requires pairings where the homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ exists as well as the hash function H_2. Therefore it can be seen from Table 7.1 that only pairings of Type 1 and 4 are possible here.
- We can broaden the usable pairing types by making each party's identity-based key consist of two values. We now denote the public key of A as $(q_A, q'_A) = (H_1(ID_A), H_2(ID_A))$ and similarly for party B. The private key of party A with identity string ID_A will then be the pair $(d_A, d'_A) = (q_A^s, {q'_A}^s)$. With the above parameters, any two principals A and B with identities ID_A, ID_B can efficiently calculate the shared key as

$$F_{AB} = \hat{e}(q_A, q'_B)^s \cdot \hat{e}(q_B, q'_A)^s = \hat{e}(d_A, q'_B) \cdot \hat{e}(q_B, d'_A) = \hat{e}(q_A, d'_B) \cdot \hat{e}(d_B, q'_A).$$

  This variant can be used with pairings of Type 1, 3, and 4, but Type 2 is still ruled out owing to the need for hashing to $\mathbb{G}_2$. Dupont and Enge [261] analysed the security of this variant.

These variants illustrate the typical problems that arise when one tries to apply different pairing types to key agreement protocols. For Types 1 and 4, we can usually make any protocol work. Some protocols can use all pairing types by avoiding pairing between user long-term keys. Chen *et al.* [192] illustrated how this works for a number of protocols. In order to simplify the presentation, we will normally show symmetric pairings in the rest of this chapter.

7.2 Identity-Based Protocols without Pairings

In recent years almost all research in identity-based key establishment has made use of bilinear pairings. However, older protocols are worth studying too, and not just because of their historical interest. For one thing, the older protocols are based on better-established computational assumptions.

Several of the older protocols work in groups with a composite modulus. It is worth emphasising that this does *not* mean that the parameter sizes for such protocols need to be larger than those used in elliptic curve pairings. Although the discrete

logarithm problem in elliptic curve groups can remain hard with much smaller parameter sizes than those used in $\mathbb{Z}_p^*$, this does not apply to pairing groups. This is because the possibility of the well-known MOV attack [544] requires that the target group $\mathbb{G}_T$ in the pairing is large enough to resist finite-field discrete logarithm attacks.

7.2.1 Okamoto's Scheme

Okamoto's scheme [589, 590] was the first published identity-based key agreement protocol. It uses a composite modulus $n = pq$ whose factorisation is known only to the KGC. The KGC chooses values e and d as in the RSA algorithm, so that $ed \bmod \phi(n) = 1$, and an element g that is a primitive root in both the integers mod p and the integers mod q. The values g and e are made public.

Before engaging in the key agreement protocol, each user must register with the authority to obtain a private key. User I's identification string, ID_I, is treated as an integer modulo n. The authority calculates the value $s_I = ID_I^{-d} \bmod n$ and distributes s_I securely to user I.

Protocol 7.1 shows the key agreement message flows. The shared secret is defined as $\mathbf{Z} = g^{er_Ar_B}$. On the assumption that it is necessary to know either s_A or s_B in order to find $\mathbf{Z}$, the scheme provides implicit key authentication.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: s_A such that $s_A^e = ID_A^{-1} \bmod n$.
Information known to B: s_B such that $s_B^e = ID_B^{-1} \bmod n$.

A		B
$r_A \in_R \mathbb{Z}_n$ $t_A = g^{r_A}$		
	$\xrightarrow{s_At_A}$	$r_B \in_R \mathbb{Z}_n$ $t_B = g^{r_B}$
	$\xleftarrow{s_Bt_B}$	
$\mathbf{Z} = ((s_Bt_B)^e ID_B)^{r_A}$		$\mathbf{Z} = ((s_At_A)^e ID_A)^{r_B}$

Protocol 7.1: Okamoto's identity-based protocol

Mambo and Shizuya [514] conducted a computational proof of security of Protocol 7.1 against a passive eavesdropper. They showed that any efficient algorithm that can find the shared secret can also break the Diffie–Hellman problem in $\mathbb{Z}_n^*$. Later, this analysis was extended by Kim *et al.* [430] to show a security proof against active attacks. These authors provided a reduction of attacks on the protocol to the Diffie–Hellman problem or to the RSA problem. However, success of the adversary requires

that the complete key is returned, rather than being defined in terms of any partial information about the key being recovered. Later, Gennaro *et al.* [294, 295] provided a security proof assuming only the difficulty of the RSA problem and in a model that requires the adversary only to distinguish the session key from a random value. In this model, ephemeral keys cannot be obtained by the adversary. Gennaro *et al.* did, however, make two small modifications to the original Okamoto protocol. One was that the identities needed to be hashed with a function which they modeled as a random oracle. The other modification was that the computation of the key included an additional squaring operation so that the shared key became $\mathbf{Z} = g^{2er_A r_B}$. Gennaro *et al.* further included a proof that Protocol 7.1 provides full forward secrecy (not just weak forward secrecy) but only by making a stronger computational assumption (the modified knowledge-of-exponent assumption).

In addition to full forward secrecy, Protocol 7.1 can also be shown to provide KCI resistance (Gennaro *et al.* [294] stated that this can be formally proven).[1] However, a significant weakness of the protocol is that it is very sensitive to compromise of ephemeral keys. If a single ephemeral value, say r_A, becomes available to the adversary then A's long-term secret is also known to the adversary. Thus, although the protocol provides full forward secrecy, it is insecure when any ephemeral key of the victim party is revealed. This means that the protocol is not secure in some models, such as eCK.

Protocol 7.2 is a variant of Protocol 7.1, proposed by Okamoto and Tanaka [590], which includes a hashed value to allow explicit authentication. Timestamps T_A and T_B, are included inside the hashes c_A and c_B, respectively, to ensure freshness. The shared secret is again $\mathbf{Z} = g^{er_A r_B}$, but this value is calculated in a different way in this variant.

There does not seem to have been any formal analysis carried out on Protocol 7.2. It seems reasonable to assume that the properties of Protocol 7.1 would carry over to Protocol 7.2 once the modifications of Gennaro *et al.* [294, 295] discussed above are incorporated. In addition, mutual explicit authentication is claimed to be achieved as long as synchronised timestamps are available. Using timestamps instead of nonces allows mutual authentication to be completed with only two messages.

A similar variant was later published by Shieh *et al.* [667] with claimed computational advantages, but Yen [757] showed that explicit authentication fails. A further variation is to provide one-pass key establishment (see also Sect. 7.5.5) suitable for applications such as electronic mail. A scheme originally proposed by Okamoto and Tanaka [590] was shown by Tsai and Hwang [714] to be vulnerable to attacks by insiders, and Tsai and Hwang proposed new schemes of their own. Later, Tanaka and Okamoto [707] designed another scheme to prevent the KGC from obtaining session keys (although the KGC can always masquerade as any user).

[1] In the first edition of this book we erroneously stated that Protocol 7.1 is vulnerable to a KCI attack.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: s_A such that $s_A^e = ID_A^{-1} \bmod n$.
Information known to B: s_B such that $s_B^e = ID_B^{-1} \bmod n$.

A		B
$r_A \in_R \mathbb{Z}_n, u_A = g^{er_A}$		
$c_A = H(u_A, ID_A, ID_B, T_A)$		
$v_A = s_A g^{c_A r_A}$	$\xrightarrow{u_A, v_A}$	$c_A = H(u_A, ID_A, ID_B, T_A)$
		$ID_A \stackrel{?}{=} u_A^{c_A} / v_A^e$
		$r_B \in_R \mathbb{Z}_n, u_B = g^{er_B}$
		$c_B = H(u_B, ID_B, ID_A, T_B)$
	$\xleftarrow{u_B, v_B}$	$v_B = s_A g^{c_B r_B}$
$c_B = H(u_B, ID_B, ID_A, T_B)$		
$ID_B \stackrel{?}{=} u_B^{c_B} / v_B^e$		
$\mathbf{Z} = u_B^{r_A}$		$\mathbf{Z} = u_A^{r_B}$

Protocol 7.2: Okamoto–Tanaka identity-based protocol

7.2.2 Günther's Scheme

Günther [336] proposed an identity-based scheme in the familiar setting of $\mathbb{Z}_p^*$. The KGC has private and public key pair x_S and $y_S = g^{x_S}$. In the registration phase, user I obtains an ElGamal signature (u_i, v_i) generated by the KGC by choosing k_i randomly in $\mathbb{Z}_p^*$ (but coprime to $p-1$) and setting $u_i = g^{k_i}$, $v_i = (ID_I - x_S u_i)/k_i \bmod p-1$. The signature pair (u_i, v_i) is given to I. The verification equation of the ElGamal signature scheme can be written as

$$u_i^{v_i} = g^{ID_I} y_S^{-u_i}.$$

Although any party can verify this equation, the idea is not to reveal v_i, but rather to reveal that I is the only party who is in possession of the discrete logarithm of $g^{ID_I} y_S^{-u_i}$ to the base u_i. Protocol 7.3 shows a successful protocol run. The shared secret is $\mathbf{Z} = u_A^{r_B v_A} u_B^{v_B r_A}$. In the description of Protocol 7.3 (and also for Protocol 7.4), we have omitted the calculation of the value ID_A by B and the value ID_B by A from the basic identifying information, which was explicitly included in the original description.

It can be seen that compromise of the long-term secrets v_A and v_B enables the adversary to find old session keys, so that forward secrecy is not provided. Therefore Günther also proposed Protocol 7.4 as an extension of the basic protocol that provides forward secrecy at the cost of an extra exponentiation on each side. A and B choose additional random values r'_A and r'_B, respectively. This time the shared secret is the same as in the basic protocol but multiplied by the ephemeral Diffie–Hellman key: $\mathbf{Z} = u_A^{r_B v_A} u_B^{v_B r_A} g^{r'_A r'_B}$. This idea of incorporating an unauthenticated

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: ElGamal signature of KGC on ID_A, (u_A, v_A), so that $g^{ID_A} = y_S^{u_A} u_A^{v_A}$.
Information known to B: ElGamal signature of KGC on ID_B, (u_B, v_B), so that $g^{ID_B} = y_S^{u_B} u_B^{v_B}$.

A		B
	$\xrightarrow{ID_A, u_A}$	
	$\xleftarrow{ID_B, u_B}$	$X_B = g^{ID_A} y_S^{-u_A}$
$X_A = g^{ID_B} y_S^{-u_B}$		
$r_A \in_R \mathbb{Z}_q$		
$w_A = u_B^{r_A}$	$\xrightarrow{w_A}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{w_B}$	$w_B = u_A^{r_B}$
$\mathbf{Z} = w_B^{v_A} X_A^{r_A}$		$\mathbf{Z} = w_A^{v_B} X_B^{r_B}$

Protocol 7.3: Günther's key agreement protocol

Diffie–Hellman value into the shared secret may seem strange at first but has often been used in other protocol designs.

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: ElGamal signature of KGC on ID_A, (u_A, v_A), so that $g^{ID_A} = y_S^{u_A} u_A^{v_A}$.
Information known to B: ElGamal signature of KGC on ID_B, (u_B, v_B), so that $g^{ID_B} = y_S^{u_B} u_B^{v_B}$.

A		B
	$\xrightarrow{ID_A, u_A}$	
	$\xleftarrow{ID_B, u_B}$	$X_B = g^{ID_A} y_S^{-u_A}$
$X_A = g^{ID_B} y_S^{-u_B}$		
$r_A, r'_A \in_R \mathbb{Z}_q$		
$w_A = u_B^{r_A}$		
$t_A = g^{r'_A}$	$\xrightarrow{w_A, t_A}$	$r_B, r'_B \in_R \mathbb{Z}_q$
		$w_B = u_A^{r_B}$
	$\xleftarrow{w_B, t_B}$	$t_B = g^{r'_B}$
$\mathbf{Z} = w_B^{v_A} X_A^{r_A} t_B^{r'_A}$		$\mathbf{Z} = w_A^{v_B} X_B^{r_B} t_A^{r'_B}$

Protocol 7.4: Günther's extended key agreement protocol

There is no security proof, or even formal security property claimed, for the original protocol of Günther, but later Fiore and Gennaro [277] did provide a proof for the extended version (Protocol 7.4), with a slight modification to the signature algorithm and where a key derivation function is applied. Informally, we may observe a similarity with the MTI protocol A(0) and, consequently, it seems reasonable to assume that one of v_A or r_B, and one of v_B or r_A, are required to find $\mathbf{Z}$. With this assumption, it follows that resistance to key compromise impersonation is provided. Indeed, resistance to key compromise impersonation and weak forward secrecy were later proven by Fiore and Gennaro [277] for their adapted version of Protocol 7.4.

Fiore and Gennaro [277] discovered a reflection attack, applicable to both Protocol 7.4 and Protocol 7.3, which allows an adversary to impersonate A to herself and obtain the correct session key. The protocols should therefore not be used in a scenario where A and B may be the same entity (such as where A shares the same key across different devices). As with Okamoto's protocol, if users cannot choose their own identities then the unknown key-share attacks on the MTI protocols do not carry over. Burmester [167] showed that his triangle attack is applicable to both versions of the protocol.

A potential disadvantage of Protocols 7.4 and 7.3 is that four messages are required, although it is worth noting that messages 2 and 4 can be combined in both protocol versions. Saeednia [640] proposed Protocol 7.5 as a variant of Günther's scheme that reduces the number of messages to only two. The idea is to replace the equation for the user's secret v_i with $v_i = ID_I k_i + x_S u_i \bmod (p-1)$ while, as before, the public key is $u_i = g^{k_i}$. Consequently, A can generate her random value $t_A = g^{r_A}$ for the protocol without waiting to receive B's public value u_B. The shared secret becomes $\mathbf{Z} = g^{v_A r_B + v_B r_A}$. The change effectively replaces the ElGamal signature in Günther's protocol with a variant signature (in fact, it is variant 3 in Table 11.5 in the *Handbook of Applied Cryptography* [550]). Saeednia also showed how to add forward secrecy to Protocol 7.5 without further message exchanges.

Fiore and Gennaro [277] provided a security proof for Protocol 7.5 after making modifications to the signature algorithm and a modification where a key derivation function is applied, similarly to how they obtained a proof for Protocol 7.4. Their security analysis includes a proof of weak forward secrecy, and they also argued for security against key compromise impersonation. Unlike Protocol 7.4, Protocol 7.5 is also secure against reflections attacks, according to the analysis of Fiore and Gennaro.

7.2.3 Fiore–Gennaro Scheme

Fiore and Gennaro [277, 278] proposed a protocol which can be viewed as an improvement of Protocol 7.5. The major difference is that a Schnorr signature is used to form the private keys of users. This allows for a more efficient protocol. Moreover, Fiore and Gennaro provided a proof of security in the Canetti–Krawczyk model. The proof covers (weak) forward secrecy and KCI resistance in addition to basic protocol security. The messages exchanged are shown in Protocol 7.6.

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: Signature of KGC on ID_A, (u_A, v_A), so that $g^{v_A} = y_S^{u_A} u_A^{ID_A}$.
Information known to B: Signature of KGC on ID_B, (u_B, v_B), so that $g^{v_B} = y_S^{u_B} u_B^{ID_B}$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		
	$\xrightarrow{ID_A, u_A, t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{ID_B, u_B, t_B}$	
$X_A = u_B^{ID_B} y_S^{u_B}$ $\mathbf{Z} = t_B^{v_A} X_A^{r_A}$		$X_B = g^{ID_A} y_S^{u_A}$ $\mathbf{Z} = t_A^{v_B} X_B^{r_B}$

Protocol 7.5: Saeednia's variant of Günther's key agreement protocol

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: Signature of KGC on ID_A, (u_A, v_A), so that $g^{v_A} = u_A y_S^{H_1(ID_A, u_A)}$.
Information known to B: Signature of KGC on ID_B, (u_B, v_B), so that $g^{v_B} = u_B y_S^{H_1(ID_B, u_B)}$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		
	$\xrightarrow{ID_A, u_A, t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{ID_B, u_B, t_B}$	
$z_1 = (t_B u_B y_S^{H_1(ID_B, u_B)})^{(r_A + v_A)}$ $z_2 = t_B^{r_A}$ $\mathbf{Z} = H_2(z_1, z_2)$		$z_1 = (t_A u_A y_S^{H_1(ID_A, u_A)})^{(r_B + v_B)}$ $z_2 = t_A^{r_B}$ $\mathbf{Z} = H_2(z_1, z_2)$

Protocol 7.6: Fiore–Gennaro key agreement protocol

The security proof for Protocol 7.6 relies on a computational assumption known as the *strong Diffie–Hellman* assumption. It also models the hash functions H_1 and H_2 as random oracles. Cheng and Ma [198] pointed out that Protocol 7.6 is not secure if ephemeral keys may be leaked, as assumed in the eCK model, but this was not allowed in the model used by Fiore and Gennaro.

7.2.4 Comparison

Table 7.2 summarises the protocols we have examined in this section, comparing their efficiency and security properties. Most of these protocols were originally published a long while back when security properties and modelling of protocols had not been extensively developed. It is therefore not surprising that originally many of these protocols did not carry security proofs. Recent work has 'modernised' some of these protocols, for example with the proof of Okamoto's protocol by Gennaro *et al.* [294, 295] and the Fiore–Gennaro version of Saeednia's protocol. An asterisk in the table indicates that a property may not hold for the original protocol; consult the section about that protocol for details.

Table 7.2: Summary of ID-based protocols without pairings. FS: forward secrecy; KCIR: key compromise impersonation resistance; ROM: random oracle model

Protocol	Message passes	Modulus type	FS	KCIR	Proof	Exponentiations on/offline
Okamoto (7.1)	2	Composite	Full*	Yes*	ROM*	1/1
OT (7.2)	2	Composite	Yes	Yes	No	2/2
Günther (7.3)	4	Prime	No	Yes	No	3/0
Günther (7.4)	4	Prime	Weak	Yes	ROM*	4/0
Saeednia (7.5)	2	Prime	Weak	Yes	ROM*	2/1
Fiore–Gennaro (7.6)	2	Prime	Weak	Yes	ROM	2/1

Most of the protocols in this section provide some form of forward secrecy, but usually this is only weak forward secrecy. An important feature of the Okamoto protocol in its refinement by Gennaro *et al.* [294, 295] is that it provides full forward secrecy. As noted in Sect. 7.2.1 this comes at the cost of fragile security in the face of compromise of ephemeral keys.

Although their origins are from quite some time ago, the protocols in this section are still competitive with the more modern pairing-based protocols examined in the next section. The computational requirements shown in Table 7.2 are divided into two parts, online and offline. The offline computations are those that can be computed before the protocol run starts. We have counted as offline those computations that require knowledge of the identity of the peer. Multi-exponentiations are counted the same as a single exponentiation in Table 7.2, while simpler computations are ignored altogether. Overall, the computational comparison should only be regarded as indicative.

7.3 Pairing-Based Key Agreement with Basic Message Format

We now turn to the popular case of identity-based key agreement using elliptic curve pairings. It is reasonable to ask what advantage there is in identity-based key agreement based on pairings in comparison with the older identity-based protocols considered in Section 7.2 above. Generally, the answer might be expected to be the same advantages as that of using elliptic curves over older public key technology, namely a saving in computation and key size. This may be true with regard to savings in bandwidth since message exchanges can be considerably shorter. However, it is not necessarily the case in terms of computation, because the pairing operation can be quite costly. Research still continues into deciding how to implement pairings most efficiently. In Sect. 7.3.9 we compare the efficiency of many pairing-based key agreement protocols. Another possible reason for choosing pairing-based key agreement is to exploit the infrastructure for identity-based cryptography, with its many other benefits.

In this section we survey a number of protocols, focusing on those which have two message passes, one in each direction between principals A and B, and which do not provide explicit authentication. (Protocols which include explicit authentication are discussed in Sect. 7.4.) There are three ingredients defining most of these protocols: the format of the key pair, the format of the exchanged messages, and the construction of the session key. We consider each of these in turn.

Key pair. Most protocols use the key construction from the first protocol of Sakai *et al.* [647], which was discussed in Sect. 7.1.3. We call this type of key the *SOK type*. There also a few examples of protocols using an alternative key type first suggested by Sakai and Kasahara [646], which we call the *SK type*.

- SOK-type keys make use of a hash function, H_1, which outputs members of the elliptic curve group $\mathbb{G}_1$. Boneh and Franklin [123] suggested an explicit H_1 function for a particular elliptic curve which costs one exponentiation in the underlying field. Then the SOK-type public key for entity A is $q_A = H_1(ID_A) \in \mathbb{G}_1$ and the private key is $d_A = q_A^s$.
- In contrast, SK-type private keys use a hash function $\hat{H}_1$ whose output is a scalar in $\mathbb{Z}_q$. In this case any regular hash function can be used for $\hat{H}_1$; the output bit string can be mapped to a number in $\mathbb{Z}_q$ in the natural way. The public key for entity A is then $q'_A = g^{s+\hat{H}_1(ID_A)}$, which can be calculated as $g^s \cdot g^{q_A}$. so that it depends on the master public key, g^s, as well as on ID_A. The SK-type private key is $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$. Note that this construction implies that $\hat{e}(d'_A, q'_A) = \hat{e}(g, g)$.

Message structure. In order to obtain the best efficiency, it is desirable to minimise the length of messages. Many protocols send only one message element typically consisting of an elliptic curve point, which can be viewed as an ephemeral key. This section is limited to protocols with only two messages. In Sect. 7.4 we look at protocols which include an authentication value, which is checked by the recipient before the session key is accepted.

Session key construction. There are many different ways in which the exchanged messages can be combined in order to derive the session key. Each party uses the received message together with its private long-term key and its short-term random input.

In the following protocol descriptions, we will assume that all users have access to the public parameters for the identity-based system. A random value $s \in \mathbb{Z}_q$ plays the role of the *master secret* of the KGC. The published values include descriptions of the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing $\hat{e}$, a point g that generates $\mathbb{G}_1$, and a master public key $h = g^s$. The KGC distributes to each party P_i with identity ID_i a long-term key pair of either SOK or SK type. We will usually assume that the pairing is a symmetric (Type 1) pairing so that $\mathbb{G}_1 = \mathbb{G}_2$. As with the key agreement protocols examined in Chap. 5 using public key certificates, we assume here that parties agree on a shared secret $\mathbf{Z}$ from which the session key will be derived using an appropriate key derivation function. Usually we do not mention the KDF explicitly but sometimes protocol designers have had a specific KDF in mind, which may have an effect on the security properties. For example, including the protocol messages in the KDF can prevent some kinds of attack. Table 7.3 summarises the notation.

Table 7.3: Notation and terminology for pairing-based schemes

ID_I	Identity string for entity I
KGC	Key generation centre
s	KGC master secret
h	KGC master public key: $h = g^s$.
$\hat{e}$	Elliptic curve pairing: $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
g	Generator of group $\mathbb{G}_1$
q_A	Public key of user A for SOK type: $q_A = H_1(ID_A)$
q'_A	Public key of user A for SK type: $q'_A = g^{s+\hat{H}_1(ID_A)}$
d_A	Private key of user A for SOK type: $d_A = q_A^s$
d'_A	Private key of user A for SK type: $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$
$\mathbf{Z}$	Shared secret

7.3.1 Smart's Protocol

Smart [679] seems to have been the first to propose, in 2002, an identity-based authenticated key agreement protocol based on pairings. The exchanged messages are no different from ordinary ephemeral Diffie–Hellman keys on elliptic curves. The pairing is used to combine identity-specific information about the parties so that only the participants should be able to obtain the shared secret $\mathbf{Z}$. Smart's protocol and its variants all use SOK-type keys, so $d_A = q_A^s$.

The messages and key computation are shown in Protocol 7.7. When both principals follow the protocol without interference, the shared secret is

Shared information: Master public key $h = g^s$ for KGC private key s. $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$\mathbf{Z} = \hat{e}(q_B^{r_A}, h) \cdot \hat{e}(d_A, t_B)$		$\mathbf{Z} = \hat{e}(q_A^{r_B}, h) \cdot \hat{e}(d_B, t_A)$

Protocol 7.7: Smart's identity-based key agreement protocol

$$\mathbf{Z} = \hat{e}(q_B^{r_A}, h) \cdot \hat{e}(q_A^{r_B}, h) = \hat{e}(q_B^{r_A} q_A^{r_B}, h).$$

Smart's protocol comes with no proof of security but it has been the basis of a number of later protocols which have been proven secure, and this can give confidence in the basic security properties of the protocol. However, the protocol does not provide forward secrecy. An adversary who obtains the two long-term private keys d_A and d_B can compute the shared key of an observed protocol run as $\mathbf{Z} = \hat{e}(d_B, t_A) \cdot \hat{e}(d_A, t_B)$. Since the KGC can generate d_A and d_B from knowledge of s, this also means that KGC forward secrecy is not provided either.

In order to provide a high level of security, it is essential that A and B check that the received values t_B and t_A lie in the group generated by g. The cost of this check is relatively cheap in the case that the protocol is implemented using a Type 1 pairing or, in general, that g lies in $\mathbb{G}_1$ when $\mathbb{G}_1$ and $\mathbb{G}_2$ are different. Chen *et al.* [192] pointed out a simple certificational attack in which the adversary simply multiplies one of the exchanged messages by a low-order value outside the group. Since this low-order element will disappear during the exponentiation with a reasonably high probability, A and B will not have matching conversations, so the protocol is broken in a strong security model.

In the original paper [679], Smart's protocol was defined only for Type 1 symmetric pairings. As pointed out by Chen *et al.* [192], it can be run using any pairing type as long as the long-term private keys are generated in $\mathbb{G}_1$.

7.3.2 Variants of Smart's Protocol

Noticing the lack of forward secrecy in Smart's protocol, Chen and Kudla [194] proposed a simple change in 2003. In addition to computing the original shared secret, they proposed to compute a straightforward ephemeral Diffie–Hellman key using the

exchanged values t_A and t_B. The messages in Smart's protocol remain unchanged, but the Diffie–Hellman key $g^{r_A r_B}$ is included in the shared secret value. The secret value therefore becomes

$$\mathbf{Z} = \hat{e}(q_B^{r_A} q_A^{r_B}, h), g^{r_A r_B}.$$

Subsequently, Chen *et al.* [192] provided a security proof for this extended protocol using a key derivation function which combines $\mathbf{Z}$, the protocol messages, and the identities of the participants. This proof shows that the protocol provides KGC forward secrecy as well as resistance to key compromise impersonation, on the assumption that the BDH problem is hard.

Later, Choie *et al.* [207] in 2005 proposed another variant of Smart's protocol which has the same basic idea of incorporating the ephemeral Diffie–Hellman value using the exchanged values. The difference in their protocol is that a hash value $f = H(g^{r_A r_B})$ is computed by both parties, where $H : \mathbb{G}_1 \to \mathbb{Z}_q$ is a hash function. The value f is then included as an exponent and the shared secret becomes

$$\mathbf{Z} = \hat{e}(q_B^{r_A} q_A^{r_B}, h)^f,$$

which is computed by A as $\mathbf{Z} = \hat{e}(q_B^{f r_A}, h) \cdot \hat{e}(d_A, t_B^f)$ and by B in a symmetrical fashion. Choie *et al.* [207] provided no security proof but claimed that the protocol provides KGC forward secrecy as well as key compromise impersonation resistance. Boyd and Choo [133] pointed out an attack on the protocol in which an adversary can obtain the session key by querying a non-matching session. However, this attack is not due to the basic structure of the protocol but due to the lack of session-specific information in the key derivation function. The attack can be avoided by including a session identifier consisting of the concatenation of the protocol messages inside the key derivation function.

7.3.3 Ryu–Yoon–Yoo Protocol

The protocol due to Ryu, Yoon and Yoo [639] has a simple and elegant structure. Again this protocol uses SOK-type keys, so $d_A = q_A^s$. Protocol 7.8 describes the protocol.

At the end of the protocol execution, both A and B will compute the shared secret $\mathbf{Z} = g^{r_B r_A}, \hat{e}(q_A, q_B)^s$. This is simply the concatenation of the ephemeral Diffie–Hellman key using the exchanged values and the SOK non-interactive key examined in Sect. 7.1.3. Since the SOK protocol can be regarded as analogous to static Diffie–Hellman, we can say that there is an analogy between Protocol 7.8 and the Unified Model protocol (Protocol 5.12). It is therefore not surprising that the properties of these two protocols are similar.

Ryu *et al.* [639] claimed that the protocol provides KCI resistance, but this is not the case [133, 727]. It is easy to see that computing the key for the SOK protocol requires knowledge of only one of d_A and d_B. Therefore, in a KCI attack, an adversary who knows d_A can compute the SOK key for any party claiming to share a key with A. Boyd and Choo [133] also described a 'key replicating attack' on Protocol 7.8. However, like the similar attack on the protocol of Choie *et al.* [207] mentioned in

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$	
$\mathbf{Z} = t_B^{r_A}, \hat{e}(d_A, q_B)$		$\mathbf{Z} = t_A^{r_B}, \hat{e}(d_B, q_A)$

Protocol 7.8: Ryu–Yoon–Yoo protocol

Sect. 7.3.2, this can be avoided by including a suitably defined session identifier in the key derivation function.

Wang *et al.* [728] provided a security proof for Protocol 7.8 when used together with a specific key derivation function. Specifically, the session key $\mathbf{K}$ is defined as

$$\mathbf{K} = H(ID_A, ID_B, \mathbf{Z}, t_A, t_B),$$

where H is a hash function modelled as a random oracle. The computational assumption is that the BDH problem is hard. A proof was also provided for KGC forward secrecy. Because the identity-based keys are used on both sides of the pairing, Protocol 7.8 can only be implemented on Type 1 and Type 4 pairings.

7.3.4 Shim's Protocol

Another early proposal for identity-based key agreement was Shim's protocol, published in 2003 [668]. This protocol uses SOK-type keys, so $d_A = q_A^s$. The original version had some serious problems, but it has later formed the basis of other protocols which have been proven secure.

The messages and key computation are shown in Protocol 7.9. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(t_A d_A, t_B q_B)^s = \hat{e}(g,g)^{sr_A r_B} \cdot \hat{e}(q_A, g)^{sr_B} \cdot \hat{e}(g, q_B)^{sr_A} \cdot \hat{e}(q_A, q_B)^s.$$

Since $\mathbf{Z}$ contains the SOK key $\hat{e}(q_A, q_B)^s$ as well as the bilinear Diffie–Hellman key $\hat{e}(g,g)^{sr_A r_B}$, it might intuitively be expected to be secure. However, Sun and Hsieh [701] found that the protocol is completely insecure, as shown in Attack 7.1.

The adversary plays in the middle between A and B and alters the messages sent between them. Once A and B complete the protocol, they have agreed on keys which can be computed by the adversary C. Note that because C chooses both u and v, C can compute $\mathbf{Z} = \hat{e}(t_A q_A, h^v)$ and $\mathbf{Z}' = \hat{e}(t_B q_B, h^u)$.

Shared information: Master public key $h = g^s$ for KGC private key s. $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
$\mathbf{Z} = \hat{e}(h^{r_A} d_A, t_B q_B)$	$\xleftarrow{t_B}$	$\mathbf{Z} = \hat{e}(h^{r_B} d_B, t_A q_A)$

Protocol 7.9: Shim's protocol

A		C		B
$r_A \in_R \mathbb{Z}_q$				
$t_A = g^{r_A}$				
	$\xrightarrow{t_A}$	$u \in_R \mathbb{Z}_q$		
		$t'_A = g^u / q_A$		
			$\xrightarrow{t'_A}$	$r_B \in_R \mathbb{Z}_q$
		$v \in_R \mathbb{Z}_q$	$\xleftarrow{t_B}$	$t_B = g^{r_B}$
		$t'_B = g^v / q_B$		
	$\xleftarrow{t'_B}$			
$\mathbf{Z} = \hat{e}(h^{r_A} d_A, t'_B q_B)$				$\mathbf{Z}' = \hat{e}(h^{r_B} d_B, t'_A q_A)$
$= \hat{e}(h^{r_A} d_A, g^v)$				$= \hat{e}(h^{r_B} d_B, g^u)$
$= \hat{e}(g^{r_A} q_A, h^v)$				$= \hat{e}(g^{r_B} q_B, h^u)$

Attack 7.1: Sun and Hsieh's attack on Shim's protocol

Later, in 2005, Yuan and Li [769] proposed a simple variation of Shim's protocol in order to avoid Attack 7.1. As in the Chen and Kudla variant of Smart's protocol, the change is simply to add the ephemeral Diffie–Hellman value to the definition of the shared secret, which therefore becomes

$$\mathbf{Z} = \hat{e}(t_A d_A, t_B d_B), g^{r_A r_B}.$$

Yuan and Li did not provide any formal security analysis, but Chen *et al.* [192] later provided a proof of security under the BDH assumption. The proof shows that this

protocol provides all the desirable properties, including KGC forward secrecy and KCI resistance.

Huang and Cao [368] proposed another variant, which is very similar to the Yuan and Li protocol. They make use of the twin Diffie–Hellman construction of Cash *et al.* [184]. The only difference from Yuan and Li's protocol is that twin public keys are constructed and used in a duplicate way in the protocol. The consequence of this is to make the proof of security simpler and more complete.

Because the identity-based keys are used on both sides of the pairing, Protocol 7.9 can only be implemented on Type 1 and Type 4 pairings.

7.3.5 Scott's Protocol

Scott [660] published an early pairing-based protocol in which the user's private key is stored as a combination of a user password and a secret stored on a physical device. In the following description, we ignore the implementation details and use the same assumptions as usual with regard to the ways that keys are constructed.

In contrast to the previous protocols in this section, Scott's protocol uses messages which are dependent on the identity of the recipient. This results in the pairing operation being used before the message is sent, but it is not needed to compute the final shared secret. Protocol 7.10 describes the message exchange.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$p_A = \hat{e}(d_A, q_B)^{r_A}$	$\xrightarrow{p_A}$	$r_B \in_R \mathbb{Z}_q$
		$p_B = \hat{e}(d_B, q_A)^{r_B}$
$\mathbf{Z} = p_B^{r_A}$	$\xleftarrow{p_B}$	$\mathbf{Z} = p_A^{r_B}$

Protocol 7.10: Scott's protocol

At the end of the protocol execution, both A and B will compute the shared secret $\mathbf{Z} = \hat{e}(q_A, q_B)^{sr_Ar_B}$ which is equal to the SOK key raised to the power r_Ar_B. The message exchange can be regarded as analogous to the MTI C(0) protocol and the key computation and protocol properties are rather similar.

Scott [660] presented an argument that the security of Protocol 7.10 can be reduced to the BDH problem, but he did not consider a full formal model. He also argued that the protocol provides forward secrecy, and this also appears to extend to KGC forward secrecy. However, the protocol does not provide resistance to KCI attacks since either of the long-term private keys is sufficient to compute the SOK key that forms part of the basis of the protocol.

Because the identity-based keys are used on both sides of the pairing, Protocol 7.10 can only be implemented on Type 1 and Type 4 pairings.

7.3.6 Chen–Kudla Protocol

Chen and Kudla [194] designed a number of protocols aimed at improving the efficiency and security of Smart's protocol. One of these has already been discussed in Sect. 7.3.2. The protocol described below reduces the number of pairing computations for each party from two to one in comparison with Protocol 7.7. Notice that the messages exchanged are also different. This protocol uses SOK-type public keys.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$w_A = q_A^{r_A}$		
	$\xrightarrow{w_A}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{w_B}$	$w_B = q_B^{r_B}$
$\mathbf{Z} = \hat{e}(d_A, w_B q_B^{r_A})$		$\mathbf{Z} = \hat{e}(d_B, w_A q_A^{r_B})$

Protocol 7.11: Chen–Kudla protocol

The messages and key computation are shown in Protocol 7.11. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(q_A^s, q_B^{r_B+r_A}) = \hat{e}(q_B^s, q_A^{r_A+r_B}) = \hat{e}(q_A, q_B)^{s(r_A+r_B)}.$$

In the original published paper [194], Chen and Kudla claimed a complete security proof for this protocol. However, subsequently [195] they pointed out a flaw in their argument (finding this flaw is attributed to Zhaohui Cheng) and only claimed security when the adversary is prevented from obtaining old session keys via reveal queries.

Protocol 7.11 does not provide forward secrecy, except for partial forward secrecy when only one of the private keys is revealed. It does, however, provide KCI resistance as proven by Chen and Kudla under the BDH assumption [194, Theorem 2], but again only when the adversary is prevented from obtaining old session keys. Chen and Kudla proposed a modification of Protocol 7.11 which adds in a separate unauthenticated Diffie–Hellman exchange, in the same manner as in their modification of Protocol 7.7. Because the identity-based keys are used on both sides of the pairing, Protocol 7.11 can only be implemented on Type 1 and Type 4 pairings.

7.3.7 Wang's Protocol (IDAK)

Wang [731, 732] designed a protocol with the same message exchange as inProtocol 7.11 but with a more complex computation of the shared secret. In compensation for this extra work a higher level of security is achieved – specifically, there is a security proof which allows reveal queries and proves forward secrecy. Wang called this protocol IDAK, to indicate identity-based and authenticated key agreement. This protocol uses SOK-type public keys.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$w_A = q_A^{r_A}$		
	$\xrightarrow{w_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$w_B = q_B^{r_B}$
	$\xleftarrow{w_B}$	
$s_A = H_2(w_A, w_B)$		$s_A = H_2(w_A, w_B)$
$s_B = H_2(w_B, w_A)$		$s_B = H_2(w_B, w_A)$
$\mathbf{Z} = \hat{e}(d_A^{r_A+s_A}, w_B q_B^{s_B})$		$\mathbf{Z} = \hat{e}(d_B^{r_B+s_B}, w_A q_A^{s_A})$

Protocol 7.12: Wang's protocol

The messages and key computation are shown in Protocol 7.12. The protocol makes use of an additional hash function $H_2 : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{Z}_q^*$. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(q_A^{r_A+s_A}, q_B^{r_B+s_B})^s = \hat{e}(q_A, q_B)^{s(r_A+s_A)(r_B+s_B)}.$$

Wang [731, 732] proved the security of Protocol 7.12 based on the decisional BDH problem. The proof is in the random oracle model, assuming that both H_1 and H_2 act as random oracles. The security proof includes forward secrecy and key compromise impersonation resilience, but the protocol does not provide KGC forward secrecy. KGC forward secrecy can be achieved by adding a separate unauthenticated Diffie–Hellman exchange.

Wang made some concrete suggestions for how to implement the function H_2 efficiently. One is to use a hash function with range $\mathbb{Z}_{q/2}^*$. Although this invalidates the full proof, Wang claimed that there was formal evidence that the protocol was still secure. This choice of H_2 allows the protocol to save half an exponentiation, since the size of s_A and s_B will be half the size of q.

Because the identity-based keys are used on both sides of the pairing, Protocol 7.11 can only be implemented on Type 1 and Type 4 pairings.

Protocol 7.13 is a refinement of Protocol 7.12 due to Chow and Choo [212]. The values s_A and s_B in Protocol 7.12 are replaced by values h_A and h_B in Protocol 7.13. The main effect of this change seems to be that it reduces the amount of *online* computation required by each party: A can compute h_A and $d_A^{r_A+h_A}$ before the protocol run starts.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$w_A = q_A^{r_A}$		
	$\xrightarrow{w_A}$	
		$r_B \in_R \mathbb{Z}_q$
		$w_B = q_B^{r_B}$
	$\xleftarrow{w_B}$	
$h_A = H_2(w_B, ID_A)$		$h_A = H_2(w_B, ID_A)$
$h_B = H_2(w_A, ID_B)$		$h_B = H_2(w_A, ID_B)$
$\mathbf{Z} = \hat{e}(d_A^{r_A+h_A}, w_B q_B^{h_B})$		$\mathbf{Z} = \hat{e}(d_B^{r_B+h_B}, w_A q_A^{h_A})$

Protocol 7.13: Chow and Choo's protocol

Chow and Choo provided a proof of security for Protocol 7.13 in the Canetti–Krawczyk model. They also formally defined a protocol variant which includes a separate Diffie–Hellman exchange, and showed that this variant provides weak KGC forward secrecy. In addition to considering the usual protocol properties, Chow and Choo also defined a protocol extension designed to provide anonymous key agreement. This extension uses a ring signature so that the peer of a party involved in the protocol can only know that the party is one out of a set (ring) of parties.

7.3.8 McCullagh–Barreto Protocol

McCullagh and Barreto [531] were the first to propose usage of SK-type keys for identity-based key agreement. One advantage of this type of key is that the hash function used, $\hat{H}_1$, does not have to map to an elliptic curve point as in SOK-type keys. Recall that the SK-type public key is $q'_A = hg^{\hat{H}_1(ID_A)} = g^{s+\hat{H}_1(ID_A)}$, with corresponding private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.

The messages and key computation are shown in Protocol 7.14. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(g,g)^{r_A r_B}.$$

McCullagh and Barreto pointed out that Protocol 7.14 does not provide KGC forward secrecy. To see this note that the KGC can compute $z_A^{(s+\hat{H}_1(ID_B))^{-1}} = g^{r_A}$

Shared information: $q'_A = g^{s+\hat{H}_1(ID_A)}$ and $q'_B = g^{s+\hat{H}_1(ID_B)}$.
Information known to A: Private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.
Information known to B: Private key $d'_B = g^{1/(s+\hat{H}_1(ID_B))}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_A = (q'_B)^{r_A}$	$\xrightarrow{z_A}$	$r_B \in_R \mathbb{Z}_q$
		$z_B = (q'_A)^{r_B}$
$\mathbf{Z} = \hat{e}(z_B, d'_A)^{r_A}$	$\xleftarrow{z_B}$	$\mathbf{Z} = \hat{e}(z_A, d'_B)^{r_B}$

Protocol 7.14: McCullagh–Barreto protocol

and $z_B^{(s+\hat{H}_1(ID_A))^{-1}} = g^{r_B}$ from its knowledge of s and the exchanged messages, and then obtain $\mathbf{Z} = \hat{e}(g^{r_A}, g^{r_B})$. They therefore also proposed a second protocol aimed at providing KGC forward secrecy using an asymmetric pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are different and are generated by unrelated elements g_1 and g_2. Private keys of users are generated in $\mathbb{G}_2$; for example, A's private key becomes $d'_A = g_2^{1/(s+\hat{H}_1(ID_A))}$ but public keys remain in $\mathbb{G}_1$ as before. The protocol messages and key computation can then be identical to Protocol 7.14. Note that user keys are applied only on the right-hand side of the pairing, which means that there is no need for a homomorphism between the pairing groups. Also, the SOK-type private key does not require hashing to the pairing group. Therefore any pairing type can be used to implement Protocol 7.14.

Protocol 7.14 was found to be subject to some weaknesses. Firstly, Xie [744] pointed out that the protocol is not resistant to KCI attacks. As a result of this attack, McCullagh and Barreto proposed a protocol variant (included only in the extended version of their paper [531] on the IACR ePrint Archive). This variant avoids the KCI attack but no longer provides forward secrecy, as subsequently pointed out by Xie [744].[2] Xie [745] also proposed a new version of the protocol with the same exchanged messages but a different key computation and shared key $\mathbf{Z} = \hat{e}(g,g)^{r_A r_B + r_A + r_B}$. Xie claimed that this change ensured forward secrecy and resistance to KCI attacks, but this variant was itself broken by Shim [670] and by Li *et al.* [486]. The latter also provided their own variant of Protocol 7.14, designed to avoid the known attacks but without any formal analysis provided.

Meanwhile, Choo [210] had shown another attack on the original version of Protocol 7.14. This attack enables an adversary to recover the session key, given the usual adversary capabilities assumed in the Bellare–Rogaway model. In order to prevent this attack, it is necessary to forbid the adversary from obtaining session keys

[2] Note that there are multiple versions of both the paper of McCullagh and Barreto [531] and the paper of Xie [744], which were updated as understanding developed. These different versions can all be obtained from the IACR ePrint Archive.

from other sessions between the same participants. In the specification of Protocol 7.14, the session key is defined to be a hash of the secret value $\mathbf{Z}$. The final IACR ePrint Archive version of the McCullagh and Barreto paper [531] claims security only when the adversary is restricted from making any reveal queries.

Later, Cheng and Chen [199] showed that all of the existing proofs of Protocol 7.14 and its variants could not be correct, owing to a technical problem. They also provided their own proof of a modified protocol which consists of McCullagh and Barreto's own variant but with an explicit key derivation function, as shown in Protocol 7.15.

Shared information: $q'_A = g^{s+\hat{H}_1(ID_A)}$ and $q'_B = g^{s+\hat{H}_1(ID_B)}$.
Information known to A: Private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.
Information known to B: Private key $d'_B = g^{1/(s+\hat{H}_1(ID_B))}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_A = (q'_B)^{r_A}$	$\xrightarrow{z_A}$	$r_B \in_R \mathbb{Z}_q$
		$z_B = (q'_A)^{r_B}$
$\mathbf{Z} = \hat{e}(z_B, d'_A)\hat{e}(g,g)^{r_A}$	$\xleftarrow{z_B}$	$\mathbf{Z} = \hat{e}(z_A, d'_B)\hat{e}(g,g)^{r_B}$

$$\mathbf{K} = H_2(ID_A, ID_B, z_A, z_B, \mathbf{Z})$$

Protocol 7.15: Modified McCullagh–Barreto protocol of Cheng and Chen

Protocol 7.15 shows the modified protocol. The messages exchanged are the same as in Protocol 7.14 but the shared secret is computed differently, to obtain $\mathbf{Z} = \hat{e}(g,g)^{r_A+r_B}$. Cheng and Chen provided a proof of security for Protocol 7.15 based on a rather complex computational assumption. However, this protocol still does not provide forward secrecy.

7.3.9 Comparison

Table 7.4 summarises the properties of the protocols which have been described in this section. Earlier surveys by Chen, Cheng and Smart [192] and by Boyd and Choo [133] have their own tables of comparison. The present table includes only the protocols which we have explicitly listed – many other protocols are known, some of which have been mentioned in the text. Recall that the protocols in this section use unauthenticated messages, and private keys are not used in their construction. The asterisks next to the properties for the Shim protocol indicate that these properties refer to the modified version discussed in Sect. 7.3.4.

As discussed in the text, many protocols have evolved over time and sometimes variants have been proposed by others. In the table an asterisk indicates that a prop-

Table 7.4: Summary of implicitly authenticated two-message ID-based protocols. FS: forward secrecy; KCIR: key compromise impersonation resistance; ROM: random oracle model

Protocol	Private key	Message type	FS	KCIR	Proof	Computation on/offline	Pairing types
Smart [679] (7.7)	SOK	g^{r_A}	No	Yes	No	$1P/2E+1P$	All
RYY [639] (7.8)	SOK	g^{r_A}	No	No	ROM [728]	$1E/1E+1P$	1,4
Shim [668] (7.9)	SOK	g^{r_A}	Yes*	Yes*	ROM [192]	$1P/2E$	1,4
Scott (7.10)	SOK	$\hat{e}(d_A,q_B)^{r_A}$	KGC	No	No	$1E/1E+1P$	1,4
CK [194] (7.11)	SOK	$q_A^{r_A}$	No	Yes	Restricted	$1P/2E$	1,4
Wang [731] (7.12)	SOK	$q_A^{r_A}$	Yes	Yes	ROM	$2E+1P/1E$	1,4
CC [212] (7.13)	SOK	$q_A^{r_A}$	KGC	Yes	ROM	$1E+1P/2E$	1,4
MB [531] (7.14)	SK	$(q'_B)^{r_A}$	Yes	No	Restricted	$1E+1P/1E$	All
MBCC [199] (7.15)	SK	$(q'_B)^{r_A}$	No	Yes	ROM	$1E+1P/1E$	All

erty may not hold for the original protocol – consult the section about that protocol for details.

There are some interesting comparisons possible between the protocols seen in Table 7.4 and various protocols using conventional Diffie–Hellman in finite fields. For example, the RYY protocol has strong similarities to the Unified Model protocol. Also, the CK protocol is closely related to the MTI A(0) protocol. Table 7.4 notes whether each protocol provides forward secrecy and key compromise impersonation resistance and has a security proof. In all cases which have forward secrecy, only weak forward secrecy is provided for these two-message protocols.

Table 7.4 also summarises the computation done by each party. We only record pairings (P) from group $\mathbb{G}_1$ to $\mathbb{G}_2$, and exponentiations (E) in either $\mathbb{G}_1$ or $\mathbb{G}_2$. For simplicity, we do not differentiate between exponentiations in $\mathbb{G}_1$ and exponentiations in $\mathbb{G}_2$. Computational requirements are divided into two parts, online and offline. The offline computations are those that can be done before the protocol run starts. We have counted as offline those computations that require knowledge of the identity of the peer. This may not always be realistic. Some computations are also independent of the peer's identity.

The amount of communication required by each protocol can be estimated by looking at the message type sent, as listed in Table 7.4. (Only the message sent from A to B is shown, but all protocols in Table 7.4 are symmetrical in their messages.) Well-known techniques for elliptic curve point compression allow points to be expressed as an element in the underlying field plus a single bit. The message length used is considerably less than for an RSA-based protocol such as Protocol 7.1 if only one point is sent. Protocols that require online pairing computation may be rather inefficient, since a pairing requires several times the computation of an elliptic curve multiplication. However, the exact computation required varies considerably depending on the choice of curve and various implementation details.

Most protocol descriptions ignore the cofactor check that may be required to ensure that the point sent is a member of the prime-order subgroup. Such a check

may be important for security reasons (to avoid small subgroup attacks such as those by Lim and Lee [492]). However, when the received point is used in a pairing, the effort required to check that the point is in $\mathbb{G}_1$ is only a small part of the overall computation required.

7.4 Pairing-Based Key Agreement with Explicit Authentication

All of the protocols which we have considered in Sect. 7.3 could be extended to include explicit authentication. This is typically achieved by using a key, generated from the shared secret independently from the session key, to form an explicit authentication tag in each direction. This would usually extend the number of message flows from two to three. In this section we consider some protocols which include explicit authentication.

7.4.1 Boyd–Mao–Paterson Protocol

Boyd, Mao and Paterson [139] proposed a protocol which uses pairings only to authenticate a Diffie–Hellman exchange. The protocol uses the SOK protocol, described in Sect. 7.1.3, to derive a static shared secret, which is then used to authenticate the exchanged messages. Protocol 7.16 shows the message exchange and the computation of the shared secret.

Shared information: Static key F_{AB}, derived from SOK key: $F_{AB} = H_1(\hat{e}(q_A, q_B)^s)$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	
		$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xleftarrow{t_B, H(F_{AB}, ID_B, t_A, t_B)}$	
Verify hash	$\xrightarrow{H(F_{AB}, ID_A, t_B, t_A)}$	
		Verify hash
$\mathbf{Z} = H_2(t_B^{r_A})$		$\mathbf{Z} = H_2(t_A^{r_B})$

Protocol 7.16: Boyd–Mao–Paterson protocol

The authenticator used to define the protocol was proven by Boyd *et al.* to satisfy the definition of a secure authenticator in the sense of Canetti and Krawczyk [178] using the random oracle model and assuming that the bilinear Diffie–Hellman problem is hard. Therefore the protocol inherits a proof of security in the Canetti–Krawczyk model, including the forward secrecy property. However, as noted by the

original authors, the protocol does not provide resistance to key compromise impersonation, since the long-term key of either party is sufficient to compute the static secret F_{AB}.

7.4.2 Asymmetric Protocol of Choi *et al.*

Choi *et al.* [206] designed a protocol for use between a low-power client A and a server B. A distinctive feature of their protocol is that A performs no pairings, even though it is a pairing-based protocol. Despite many recent advances, pairing computations are still more expensive than exponentiations, so it is desirable to reduce the number of pairings; by eliminating pairing computation altogether on the client side, there is a corresponding saving in implementation cost too. Protocol 7.17 shows the structure; three hash functions are used, $\hat{H}_1$ is the usual function for SK keys from identity strings to $\mathbb{Z}_q$, and H_2 and H_3 output bit strings.

Shared information: $q'_A = g^{s+\hat{H}_1(ID_A)}$ and $q'_B = g^{s+\hat{H}_1(ID_B)}$.
Information known to A: Private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.
Information known to B: Private key $d'_B = g^{1/(s+\hat{H}_1(ID_B))}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = \hat{e}(g,g)^{r_A}$		
$z_{A,1} = (q'_B)^{r_A}$		
$z_{A,2} = (d'_A)^{r_A+\hat{H}_1(t_A)}$		
	$\xrightarrow{ID_A, z_{A,1}, z_{A,2}}$	
		$t_A = \hat{e}(z_{A,1}, d'_B)$
		$\hat{e}(z_{A,2}, q'_A) \stackrel{?}{=} t_A \cdot \hat{e}(g,g)^{\hat{H}_1(t_A)}$
		$r_B \in_R \mathbb{Z}_q$
		$z_B = H_2(t_A, r_B, z_{A,1}, z_{A,2}, ID_A, ID_B)$
	$\xleftarrow{z_B, r_B}$	
$z_B \stackrel{?}{=} H_2(t_A, r_B, z_{A,1}, z_{A,2}, ID_A, ID_B)$		

$$\mathbf{K} = H_3(t_A, r_B, z_B, z_{A,1}, z_{A,2}, ID_A, ID_B)$$

Protocol 7.17: Protocol of Choi, Hwang, Lee and Seo

Part of Protocol 7.17 is similar to Protocols 7.14 and 7.15 in that the value $z_{A,1}$ can be used by B to recompute the value $t_A = \hat{e}(g,g)^{r_A}$. However, the other part of the message from A, $z_{A,2}$, is intended as a kind of signature to allow B to authenticate the message. We note, however, that B has no way to check if the message from A has been replayed, so it does not provide explicit entity authentication. The server

sends its input r_B in cleartext and both parties can then compute the session key as a hash of $\hat{e}(g,g)^{r_A}$, r_B and other public values. B also includes a value z_B, which can be recomputed by A to authenticate the message and to explicitly authenticate B.

Protocol 7.17 provides only partial forward secrecy; compromise of the long-term key of the client, A, does not reveal expired session keys, but compromise of the long-term key of B does. Owing to the authentication of each message, the protocol appears to achieve key compromise impersonation resistance, although this has not been proven. Choi *et al.* [206] provided a proof of security of Protocol 7.17 on the assumption that the hash functions are random oracles and using two computational assumptions known as the 'k-value modified bilinear inverse Diffie–Hellman' and the 'k-value collusion attack algorithm' assumptions.

Later, Wu and Tseng [743] proposed a protocol related to Protocol 7.17, intended for the same application scenario. The Wu and Tseng protocol uses SOK-type private keys instead of the SK-type keys used in Protocol 7.17. Both protocols avoid the use of pairings on the client side and have the same online computational requirements for the client A. A comparison of the two given by Wu and Tseng [743] indicates that they can save one exponentiation overall for the server compared with Protocol 7.17.

7.4.3 Identity-Based Key Agreement without Random Oracles

In all of the ID-based protocols which we have looked at so far, it has been necessary for the identity string to be hashed before being used in the protocol. In order for the security proof to work, this hash function is usually modelled as a random oracle. Such a requirement is typical of the ID-based encryption algorithms which were developed in the first decade of the explosion in research on pairings which started around the year 2000. Later, it was seen as an important research goal to remove random oracle assumptions wherever possible and some different solutions to that problem were found. It seems a natural goal to do the same for ID-based key exchange, but there has not been a lot of focus on this goal.

One protocol which achieves this goal is due to Tian *et al.* [711]. It is based on the ID-based encryption scheme of Gentry [297] and uses the same parameters and private keys. In Gentry's scheme, the public parameters consist of three values (g, g_1, h) for randomly chosen $g, h \in \mathbb{G}$ and $g_1 = g^{\alpha}$, where α is the master secret key. The private key of entity A is a pair (e_A, h_A), with $h_A = (hg^{-e_A})^{1/(\alpha - ID_A)}$. The protocol also makes use of a secure MAC. The protocol message exchange is shown in Protocol 7.18.

Through the first two messages of Protocol 7.18, A and B exchange what are ciphertexts of empty messages with Gentry's scheme. This allows them to obtain two shared secrets: $K_A = \hat{e}(g,h)^{r_A}$, generated implicitly by A, and $K_B = \hat{e}(g,h)^{r_B}$, generated implicitly by B. These keys are used firstly as keys for MACs which provide explicit authentication, and secondly to form the shared secret $\mathbf{Z} = \hat{e}(g,h)^{r_A r_B}$. The protocol is relatively expensive, requiring two pairings and five exponentiations on each side.

Tian *et al.* [711] provided a security proof in a Bellare–Rogaway-style model. Like Gentry's scheme, their security proof relies on a rather complex computa-

Shared information: Public parameters (g, g_1, h) with $g, h \in_R \mathbb{G}$ and $g_1 = g^{\alpha}$ where α is the master secret key
Information known to A: Private key (e_A, h_A) with $h_A = (hg^{-e_A})^{1/(\alpha - ID_A)}$.
Information known to B: Private key (e_B, h_B) with $h_B = (hg^{-e_B})^{1/(\alpha - ID_B)}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_{A,1} = (g_1 g^{-ID_B})^{r_A}$		
$z_{A,2} = \hat{e}(g,g)^{r_A}$		
	$\xrightarrow{ID_A, z_{A,1}, z_{A,2}}$	
		$r_B \in_R \mathbb{Z}_q$
$K_A = \hat{e}(g,h)^{r_A}$		$z_{B,1} = (g_1 g^{-ID_A})^{r_B}$
		$z_{B,2} = \hat{e}(g,g)^{r_B}$
		$K_A = \hat{e}(z_{A,1}, h_B) z_{A,2}^{e_B}$
	$\xleftarrow{ID_B, z_{B,1}, z_{B,2}, \mathsf{MAC}_{K_A}(ID_A, z_{A,1}, z_{A,2}, ID_B, z_{B,1}, z_{B,2})}$	
Verify MAC		
$K_B = \hat{e}(z_{B,1}, h_A) z_{B,2}^{e_A}$		
	$\xrightarrow{\mathsf{MAC}_{K_B}(ID_B, z_{B,1}, z_{B,2}, ID_A, z_{A,1}, z_{A,2})}$	
		$K_B = \hat{e}(g,h)^{r_B}$
$\mathbf{Z} = K_B^{r_A}$		Verify MAC
		$\mathbf{Z} = K_A^{r_B}$

Protocol 7.18: Protocol of Tian, Susilo, Ming and Wang

tional assumption known as the *truncated decisional ABDHE assumption*. They also claimed, without formal proof, that their protocol also achieves KCI resilience and forward secrecy.

We can divide Gentry's encryption scheme [297] into a key encapsulation method component and a data encapsulation method. This construction is therefore strongly related to the generic KEM-based construction examined in Sect. 5.8. Indeed, by using any identity-based KEM that is secure in the standard model, the construction in Sect. 5.8 can be used to construct alternative ID-based key agreement protocols without random oracles.

7.4.4 Comparison

Table 7.5 summarises the properties of the protocols which have been described in this section. Recall that the protocols in Table 7.5 include direct authentication information as a signature of some sort.

The protocols in this section have differing structures, so we have not tried to compare their message format except for noting the private key type. Each of them

Table 7.5: Summary of two-party ID-based protocols with explicit authentication. FS: forward secrecy; KCIR: key compromise impersonation resistance; ROM: random oracle model; Std: standard model

Protocol	Private key	FS	KCIR	Proof	Computation on/offline
BMP [139] (7.16)	SOK	KGC	No	ROM	$1E/1P+1E$
CHLS [206] (7.17)	SK	No	Yes	ROM	$-/3E$ (client), $3P+1E/-$ (server)
TSMW [711] (7.18)	Gentry	Yes	Yes	Std	$2P+3E/2E$

has a proof of security, with the TSMW protocol being the only explicit protocol we have listed which has a proof in the standard model.

As in Table 7.4, we have also summarised the computation of each party in Table 7.5. Again, we only record pairings (P) and exponentiations (E), and computational requirements are again divided into two parts, online and offline. For the CHLM protocol, the computation is different for the client (all can be done offline) and for the server (all online).

As mentioned at the start of this section, any of the protocols in Sect. 7.3 could be converted into a protocol with explicit authentication. This would make little difference to the computation on each side shown in Table 7.4, and in many cases would allow full strong forward secrecy to be achieved.

7.5 Identity-Based Protocols with Additional Properties

All of the protocols which we have examined in this chapter so far have the same basic infrastructure, namely a KGC which issues private keys to principals based on their identity information. There are a number of ways that this infrastructure can be extended, both for reasons of practicality and in order to provide new properties. In this section we consider a few of these ways, namely how to accommodate domains with different KGCs, how to incorporate user-generated keys, and how to allow more flexible ways to define which principals can participate. Finally, in this section we include a discussion of one-pass key establishment, which has a special significance for the identity-based case.

7.5.1 Using Multiple KGCs

In our descriptions of ID-based protocols we have assumed that all users rely on the same KGC to generate their private keys. In a large-scale system this is not practical. This issue has been noticed for ID-based cryptography in general since it was first described. There have been proposals for a hierarchical structure of KGCs to operate with identity-based encryption. Such structures spread the load on KGCs by allowing entities high in the hierarchy to issue keys for entities that act as KGCs for lower layers. There appears to have been little work on investigating the inclusion of

hierarchies for ID-based key exchange in a generic fashion. However, some schemes have extensions allowing for multiple KGCs.

Chen and Kudla [194] presented a variant of Protocol 7.7 designed to accommodate the situation where two KGCs operate using the same public parameters (i.e. the same groups and generators) but different KGC master secrets. McCullagh and Baretto [531, Section 5] claimed a more efficient protocol. However, as pointed out before (see Sects. 7.3.1 and 7.3.8), both of these papers have limitations in their security proofs. Fujioka *et al.* [287] proposed a specific ID-based protocol using the key hierarchy of Gentry and Silverberg [300]. Guo and Zhang [338] considered a different but related setting in which ID-based and traditional PKI-based settings are combined.

One notable example of usage of multiple KGCs is the protocol of Schridde *et al.* [658], designed as a variant of Protocol 7.1 which allows users with different KGCs to agree on a secret. Although extra computation is required, it is not necessary for the two KGCs to communicate to set up their separate system parameters. Protocol 7.19 shows the protocol messages. The users need to employ the Chinese Remainder Theorem to compute new secret values s'_A and s'_B so that the session key derivation equation still works. The shared secret is a value in the integers modulo $n_1 n_2$, where n_1 is the modulus used by A's KGC and n_2 is the modulus used by B's KGC.

To see that both A and B compute the same value $\mathbf{Z} = (g_1 g_2)^{e_1 e_2 r_A r_B} \bmod n_1 n_2$ in Protocol 7.19, note that the value computed by A is

$$\begin{aligned}
\mathbf{Z} &= ((s'_B t'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 \\
&= ((s'_B)^{e_1 e_2} ID'_B)^{r_A} \cdot ((t'_B)^{e_1 e_2})^{r_A} \bmod n_1 n_2 \\
&= ((s'_B)^{e_1 e_2} ID'_B)^{r_A} \cdot ((g^{r_B})^{e_1 e_2})^{r_A} \bmod n_1 n_2 \\
&= ((s'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 \cdot \mathbf{Z}.
\end{aligned}$$

However,

$$\begin{aligned}
(s'_B)^{e_1 e_2} ID'_B \bmod n_2 &= s_B^{e_1 e_2} ID_B^{e_1} \bmod n_2 \\
&= (ID_B^{-1})^{e_1} (ID_B)^{e_1} \bmod n_2 \\
&= 1.
\end{aligned}$$

Similarly, $(s'_B)^{e_1 e_2} ID'_B \bmod n_1 = 1$, so that $(s'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 = 1$.

Gennaro *et al.* [294] presented a security proof for Protocol 7.19. They made a few adjustments to the protocol to ensure that the security proof holds. As for Protocol 7.1, it is necessary that the ID values are hashed and that the session key is obtained using a key derivation function which includes the exchanged messages as a session identifier. Moreover, Gennaro *et al.* noted that where the exponent $e_1 e_2$ is used in the derivation of $\mathbf{Z}$ in Protocol 7.19, it can be replaced by $E = \operatorname{lcm}(e_1, e_2)$ – this gives the same result and can be significantly more efficient. Another change is that the value of $\mathbf{Z}$ must be squared so that the shared secret becomes $\mathbf{Z} = g^{2Er_A r_B} \bmod n_1 n_2$. Finally, they defined g using the Chinese Remainder theorem so that $g \equiv g_1 \pmod{n_1}$ and $g \equiv g_2 \pmod{n_2}$.

Shared information: Public moduli n_1, n_2 and exponents e_1, e_2. Elements g_1, g_2 of high order in $\mathbb{Z}^*_{n_1}$ and $\mathbb{Z}^*_{n_2}$. Let $g = g_1 g_2$.
Using the Chinese Remainder Theorem, both parties compute ID'_A and ID'_B such that

$$\begin{aligned} ID'_A &\equiv ID_A^{e_2} \bmod n_1, \\ ID'_A &\equiv 1 \bmod n_2, \\ ID'_B &\equiv ID_B^{e_1} \bmod n_2, \\ ID'_B &\equiv 1 \bmod n_1. \end{aligned}$$

Information known to A: s_A such that $s_A^{e_1} = ID_A^{-1} \bmod n_1$. Using the Chinese Remainder Theorem, A computes s'_A such that

$$\begin{aligned} s'_A &\equiv s_A \bmod n_1, \\ s'_A &\equiv 1 \bmod n_2. \end{aligned}$$

Information known to B: s_B such that $s_B^{e_2} = ID_B^{-1} \bmod n_2$. Using the Chinese Remainder Theorem, B computes s'_B such that

$$\begin{aligned} s'_B &\equiv s_B \bmod n_2, \\ s'_B &\equiv 1 \bmod n_1. \end{aligned}$$

A		B
$r_A \in_R \mathbb{Z}_{n_1 n_2}$		
$t'_A = g^{r_A} \bmod n_1 n_2$	$\xrightarrow{s'_A t'_A}$	$r_B \in_R \mathbb{Z}_{n_1 n_2}$
	$\xleftarrow{s'_B t'_B}$	$t'_B = g^{r_B} \bmod n_1 n_2$
$\mathbf{Z} = ((s'_B t'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2$		$\mathbf{Z} = ((s'_A t'_A)^{e_1 e_2} ID'_A)^{r_B} \bmod n_1 n_2$

Both parties compute the secret

$$\mathbf{Z} = (g_1 g_2)^{e_1 e_2 r_A r_B} \bmod n_1 n_2$$

Protocol 7.19: Schridde *et al.* cross-domain identity-based protocol

7.5.2 Girault's Three Levels

Girault [305] introduced a three-level categorisation of key agreement based on a generalisation of identity-based schemes. In the schemes at level 1, the public key of the entity is the identity string ID_I, so these are exactly the normal identity-based schemes. At the higher levels, a value obtained from the KGC is used in combination with the partner's identity to derive a key that can only be calculated by a principal with the correct private key. We call such a value an *implicit certificate*. This allows the private keys to be kept secret from the KGC and can be regarded as a compromise between the basic identity-based scheme and conventional PKI-based schemes. The

difference between levels 2 and 3 in Girault's classification depends on whether or not the owner of the public key can alone compute a valid public key (see Table 7.6).

Table 7.6: Girault's levels of extended identity-based schemes

Level	Properties	Example
1	KGC chooses, or can compute, private key.	Okamoto [589, 590], Protocol 7.1
2	KGC cannot find private key. Principal can generate contradictory public key.	Girault and Paillès [306]
3	Only KGC can generate valid public key.	Girault [305], Protocol 7.20

In order to understand the motivation behind the level 3 schemes, recall that a certificate of a public key is a signature by a third party on certain information that includes the value of the public key. A principal who receives such a certificate, including the owner of the private key, cannot use it to form another contradictory certificate (for example, one that shows a different public key). This is simply because only the third party can form new signatures. A malicious certification authority can generate its own private key and produce a certificate that shows that this private key belongs to any victim principal. This would allow the authority to masquerade as the principal. However, this certificate will contradict the real certificate of the principal. Because only the authority is able to produce a certificate, the two contradictory certificates can be used to show that the authority has cheated. In Girault's level 2 schemes, principals can choose their own private key and also use their private information to form new implicit certificates. Since both the authority and the principal can form contradictory certificates, it is impossible to tell which of them has cheated when two certificates appear. This situation is arguably little better than when the authority has access to the private key, since it is able to masquerade as any principal without being caught.

Girault and Paillès' protocol [306] was classified by Girault as level 2. There is a strong connection with Okamoto's Protocol 7.1. In Girault and Paillès' protocol the public key, y_A, of A satisfies $y_A^e ID_A = g^{-ex_A} \bmod n$, where g^{x_A} is given to the server but x_A is kept secret by A. But if we rearrange this we find that $y_A g^{x_A} = ID_A^{-d}$, so that A could calculate y_A herself when given the same private key from Okamoto's scheme. In the Girault and Paillès protocol A sends the message $y_A g^{x_A - r_A}$ to B. But when we rearrange this message it becomes $ID_A^{-d} g^{-r_A}$, which is the same as the corresponding message sent in Protocol 7.1 except for a change of sign. Similarly, the shared secrets are identical in the two protocols. We conclude that there is essentially no benefit in A choosing the extra secret x_A, even though the Okamoto scheme is at level 1 while the Girault and Paillès protocol is at level 2.

In 1991 Girault, [305] introduced *self-certified* public keys in order to avoid the limitations of level 2 schemes. These are keys that have an implicit certificate that can only be generated by the KGC, and consequently can be used in protocols to reach level 3 of Girault's classification. Girault's level 3 key agreement scheme [305] using

self-certified keys has the same algebraic setting as that used for Okamoto's protocol (Protocol 7.1). An RSA modulus n and key pair e, d are chosen by the server, together with an element g of high order in $\mathbb{Z}_n^*$. When A registers to use the scheme, she chooses her private key x_A and provides g^{x_A} to the KGC, which calculates the self-certified public key $y_A = (g^{x_A} - ID_A)^d \bmod n$. (We have changed the sign of x_A from Girault's original in order to maintain our usual notation.) In order for the scheme to achieve level 3, it is essential that the KGC is unable to find x_A. Saeednia [641] has pointed out that a malicious server that chooses n may be able to find discrete logarithms relatively easily, and for this reason the size of n should preferably be 2048 bits. In addition the KGC should provide a proof that n is the product of two safe primes; Camenisch and Michels [175] provided a method to achieve such a proof.

In Girault's original paper [305], he only suggested using the self-certified keys to produce an authenticated static shared key. If the public keys are already available, this can be achieved with no message exchanges: A calculates $(y_B^e + ID_B)^{x_A}$ in order to produce $\mathbf{Z} = g^{x_A x_B}$ and B calculates the analogous value. However, since the public key is simply a way to find an implicitly certified key for each party, any of the MTI protocols (see Sect. 5.3) can be modified to provide a dynamic protocol. For example, Protocol 7.20 shows a version modified from MTI A(0), as suggested by Rueppel and van Oorschot [637]. The shared key is $\mathbf{Z} = g^{r_A x_B + r_B x_A}$. The usual extra checks should be made in order to avoid potential attacks, as discussed in Sect. 5.3. Analogues of the other MTI protocols can be made similarly, replacing y_A in the MTI original with $y_A^e + ID_A$ in the Girault version, and similarly for y_B.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: x_A with $y_A^e + ID_A = g^{x_A} \bmod n$, y_B.
Information known to B: x_B with $y_B^e + ID_B = g^{x_B} \bmod n$, y_A.

A		B
$r_A \in_R \mathbb{Z}_n$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_n$
	$\xleftarrow{t_B}$	$t_B = g^{r_B}$
$\mathbf{Z} = t_B^{x_A}(y_B^e + ID_B)^{r_A}$		$\mathbf{Z} = t_A^{x_B}(y_A^e + ID_A)^{r_B}$

Protocol 7.20: Girault's identity-based protocol, adapted by Rueppel and van Oorschot

Just like the MTI A(0) protocol, Protocol 7.20 does not provide forward secrecy: knowledge of x_A and x_B allows an adversary to compute $\mathbf{Z}$. However, KCI resistance

does seem to be provided, but there is currently no proof of that, nor indeed that the protocol is secure at all.

Nyberg and Rueppel [585] suggested using their signature with message recovery, previously discussed in Sect. 5.2.2, as the basis of identity-based key agreement. The idea is to make the public key of each principal A equal to the signature of $g^{x_A} ID_A^{-1}$. After recovery of the message from the signature, the implicitly certified public key can be recovered by multiplying by ID_A. With this basis, any of the MTI-style protocols can be applied, in the same manner as in Protocol 7.20. Sakazaki *et al.* [648] explored use of elliptic curves and other variations in order to make the basic idea more efficient.

7.5.3 Certificateless Key Agreement

Despite the efficiency advantages of identity-based key exchange, it has the significant drawback that it has the so-called 'escrow' property: the KGC can obtain the private key of any party. If the protocol has KGC forward secrecy, this may not be as serious a problem as it would otherwise be, because the KGC will be unable to recover session keys from sessions in which it was not active. But the KGC can still masquerade freely as any user. Furthermore, if a malicious KGC is able to obtain ephemeral keys used in the session then it will be able to obtain the agreed session key (since it will know all the secrets).

For the above reasons there is value in considering the *certificateless* setting. Certificateless cryptography was introduced by Al-Riyami and Paterson [24] to provide some of the benefits of identity-based cryptography without allowing the KGC to know the secrets of users. To achieve this, users have two parts to their private key: one is issued by the KGC and the other chosen by the user. A user's public key must be known to communication partners but is not certified. This can be seen as a similar idea to Girault's level 3 scheme discussed in Sect. 7.5.2.

Since the idea of Al-Riyami and Paterson [24] was introduced in 2003, many certificateless cryptographic primitives have been designed, including encryption and signatures. There have also been several certificateless key agreement protocols; indeed, the first was in the original Al-Riyami–Paterson paper. However, it was not until 2009 that a formal model for security was defined [497, 702]. In such a model, we expect the adversary to obtain the KGC private key as well as the ephemeral private keys of the parties and still be unable to obtain the session key. This shows that the model is stronger than what can be achieved in the identity-based setting.

Swanson and Jao [702] showed that all of the protocols published before 2009 are insecure in a strong model of security. Lippold *et al.* [497] seem to have been the first to propose a certificateless key agreement protocol that is secure in a strong security model. Their protocol remains secure even if any two of the three keys (the identity-based private key, the user-selected private key, and the ephemeral private key) become compromised. The security proof is in the random oracle model and relies on the computational Diffie–Hellman assumption. However, the protocol is relatively expensive, requiring 10 elliptic curve pairings for each party. Slightly later, Lippold *et al.* [496] proposed another certificateless key exchange protocol which

is secure in the standard model and is more efficient than their previous proposal. It is essentially the same as Protocol 5.43, where the key encapsulation mechanism used is a certificateless one. Yang and Tan [750] have proposed an even more efficient protocol without relying on pairings; it relies instead on the gap Diffie–Hellman problem.

7.5.4 Protocols with Generalised Policies

In the past few years, identity-based cryptography has been generalised in a few different ways to allow more flexible and expressive properties to be specified. For example, identity-based encryption can be generalised so that ciphertexts can be decrypted by any user who possesses certain properties, such as being a member of a group or being above a certain age. Identity-based encryption is then a special case of this, where the specific property required for decryption is having an identity equal to a specific value. Various flavours of generalisation have been defined, including attribute-based cryptography, predicate cryptography, and, more generally, functional cryptography [124]. At the time of writing, this is still a developing research area where many new results can be expected in the coming years. Here we just mention some early contributions to applying these generalisation to key exchange protocols.

At the same conference in 2010, papers were published on *predicate-based key exchange* by Birkett and Stebila [104] and on *attribute-based key exchange* by Gorantla *et al.* [327]. These two papers are related in that their intent is to generalise the access policy to the shared secret, but they also have significant differences. The first is concerned with two-party key exchange, and one main requirement is to hide the properties (attributes) held by the participants. The second presents a group key exchange protocol in which any user can participate by holding the necessary properties, while keeping the user identity secret. Either of these approaches may be useful, depending on the application scenario. Other papers have built on these ideas [694, 765]. A related notion is credential-based key exchange, introduced by Camenisch *et al.* [174].

7.5.5 One-Pass Identity-Based Protocols

One of the major benefits of identity-based cryptography is that users do not need to access keying material for communication partners before applying cryptographic processing based on the partner's identity. When using conventional (certificate-based) public keys, a user who wishes to encrypt information for a chosen recipient must know the correct public key of the recipient. In contrast, identity-based encryption requires only the recipient's identity (and the public parameters) to be known. In this sense, identity-based encryption is more useful than identity-based signatures. A signer can append a certificate to a conventional public signature to make it identity-based in the sense that only the public parameters (which can include the certifier's public key) and the identity are required to verify the signature.

We can develop this argument in the context of key exchange protocols to arrive at the conclusion that many key exchange protocols with two (or more) messages can be made identity-based simply by adding certificates to each of the (first two) messages. This will always work as long as the first message from the initiator does not depend on the public key of the responder. Of course, this does not mean that such protocols will be efficient or have other desirable properties. Also, it may be desirable to share public keys and parameters from an identity-based infrastructure used for encryption and reuse them for key exchange. Thus we certainly do not claim that the two- and three-message protocols explored in this chapter are not interesting. However, we can say that the relationship of one-pass key exchange to two-pass key exchange is similar to the relationship of identity-based encryption to identity-based signatures. The latter can always be obtained from conventional primitives, while the former cannot.

It is reasonable to expect that protocols with only one message will not achieve as high a level of security as those with more messages. Noticing that an adversary who obtains the recipient's private key has the same information as the recipient during the protocol, we can see that one-pass protocols cannot achieve full forward secrecy. The best that can be achieved is *sender forward secrecy* so that compromise of the sender's private key will not compromise the session key. Similarly, an adversary who can obtain the recipient's private key can impersonate the sender unless the single message is explicitly authenticated (and the adversary can always replay a message), so that key compromise impersonation to the recipient cannot generally be prevented.

Okamoto *et al.* [592] described two protocols and argued that their protocols provide sender forward secrecy and security against key compromise impersonation to the sender. Around the same time, Wang's protocol [731, 732], which we saw in Sect. 7.3.7, was published. Wang pointed out that Protocol 7.12 can be adapted to a one-pass protocol by setting $r_B = 1$ and $w_B = q_B$ so that the message w_B sent from B to A can be removed. However, none of these earlier protocols carries any proof of security. Gorantla *et al.* [325] seem to have been the first to provide a design for one-pass identity-based key exchange with a security proof. Their protocol is shown as Protocol 7.21.

A		B
$r_A \in_R \mathbb{Z}_q$		
$w_A = q_A^{r_A}$		
	$\xrightarrow{w_A}$	
$s = H_2(w_A, ID_A, ID_B)$		$s = H_2(w_A, ID_A, ID_B)$
$\mathbf{Z} = \hat{e}(d_A^{r_A+s}, q_B)$		$\mathbf{Z} = \hat{e}(s_B, w_A q_A^s)$

Protocol 7.21: Protocol of Gorantla, Boyd, and González-Nieto

There is a strong similarity between Protocol 7.21 and Protocol 7.12 by Wang. Indeed, Protocol 7.21 is a simplified version of Wang's one-pass version of Protocol 7.12 mentioned above. Gorantla *et al.* [325] provided a security proof of Protocol 7.21 in the random oracle model, assuming hardness of the BDH problem.

It is a reasonable question to ask whether there is any real difference between a one-pass key exchange protocol and a hybrid encryption scheme. In both cases, a key is set up which is then used to protect other exchanged data, and this key can depend on both the sender's and the recipient's long-term keys. This question was investigated by Gorantla *et al.* [324] who concluded that there is a duality between one-pass key exchange and a primitive known as a signcryption KEM. With suitable assumptions, it is possible to transform a one-pass key exchange protocol into a signcryption KEM and vice versa.

7.6 Conclusion

There has been a huge amount of research on identity-based cryptography, mostly since the year 2000. Identity-based key exchange has been a significant part of this and it may be fair to say that the area is reasonably mature, at least with regard to pairing-based solutions. We have a number of well-understood protocols which are practically efficient and with security proofs based on widely accepted computational assumptions.

One direction where we may see future developments is in lattice-based solutions. Lattices appear to be a more promising long-term foundation for identity-based cryptography. One reason for this is the likelihood that lattice-based algorithms will be able to withstand attacks from quantum computers, which threaten to undermine many current cryptographic technologies, including pairings. The other direction where we anticipate new results is in protocols whose principals are defined by something more flexible than identity. These can be expected to emerge from research into generalised forms of cryptographic primitives, particularly in the area of functional cryptography.

8

Password-Based Protocols

8.1 Introduction

Cryptographic authentication relies on possession of a key by the party to be authenticated. Such a key is usually chosen randomly within its domain and can be of length from around 100 bits up to many thousands of bits, depending on the algorithm used and security level desired. Experience has shown [273, 741] that humans find it difficult to remember secrets in the form of passwords of even seven or eight characters. But if all upper- and lower-case letters are used together with the digits 0 to 9 then a random eight-character password represents less than 48 bits of randomness. Therefore we can conclude that even short random keys for cryptographic algorithms cannot be reliably remembered by humans. Another way to express this is that it can be assumed that a computer is able to search through all possible passwords in a short time.

Cryptographic keys are often stored in secure memory in computers or using special devices such as tamper-resistant cryptographic servers or in smart cards. However, there are situations where this is inconvenient or expensive. Not all devices are tamper resistant, and the memory required for public keys can be scarce. There have been proposals to strengthen passwords through interaction with a server [5, 281] but these either require an additional trusted entity or give away some information about the password. It is desirable to be able to set up secure communications relying only on a short secret that can be remembered by humans. This chapter examines a number of key establishment protocols that have been designed to be secure in the situation that the principals share only a password of small entropy. In accordance with common terminology, we will often refer to such protocols as *password-authenticated key exchange* protocols and employ the acronym PAKE.

At first thought, it might seem impossible to achieve key establishment using only a short secret in such a way that brute force searching to find the secret is not possible. This intuition may account for why it was not until 1989 that the first password-based protocols appeared in the literature. These first protocols, due to Lomas *et al.* [500], used the additional assumption that the client (in a client–server application) has knowledge of the server's public key, in addition to sharing the password with the

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_8

server. In 1992, Bellovin and Merritt [84] introduced a class of protocols that does not require this assumption.

The idea of Bellovin and Merritt's Encrypted Key Exchange (EKE) protocols is that the protocol initiator will choose an ephemeral public key and use the shared password to *encrypt* this key. The responder can decrypt the public key and use it to send the session key securely back to the initiator. On the assumption (not always reasonable) that public keys are random strings, an adversary who tries a brute force search of passwords will not be able to distinguish which ephemeral public key was used; furthermore, even when the correct public key has been found it cannot be used to discover the session key, since it is not possible to obtain the private key from the public key. It is interesting to compare password-based protocols with protocols providing forward secrecy. Both seem to be possible only through the use of ephemeral public keys[1] and, for both, a typical approach is to use long-term keys (which may be passwords) only for authentication of the message exchanges. Many password-based protocols do provide forward secrecy.

We may summarise the special properties of and requirements for password-based key establishment protocols as follows. Additional properties, such as forward secrecy, are often seen as desirable.

- Users only possess a secret of small entropy. Specifically, it is possible for the adversary to search through all possible secrets in a short time.
- Offline dictionary attacks should not be feasible. This means that a passive eavesdropper who records the transcript of one or more sessions cannot eliminate a significant number of possible passwords.
- Online dictionary attacks should not be feasible. This means that an active adversary cannot abuse the protocol so as to eliminate a significant number of possible passwords. An active adversary can always eliminate at least one password per protocol run by attempting to masquerade using that password. Ideally, this should be all that the adversary can gain.

An early idea to limit the leakage of passwords against online dictionary attacks was to use a *collisionful* keyed hash function. In contrast to the usual requirement for hash functions, this is a function for which it is *easy* to find many collisions for any input. Anderson and Lomas [33] proposed to use a password as the key for a collisionful hash function to authenticate Diffie–Hellman key exchange. Then password guessing can only eliminate a proportion of passwords. While an ingenious idea, it does not provide the level of security which we expect in more modern protocols.

Today, we can identify two different main approaches to the design of PAKE protocols. The first is based around the idea from the EKE protocol in which an ephemeral public key is somehow masked by the shared password. The second approach is to make use of smooth projective hash functions which allow only owners of the password to obtain the shared secret (see Sect. 8.3.8). Abdalla [8] provided a good overview of the two approaches.

[1] Halevi and Krawczyk [342] provided formal arguments in support of this view.

The security model used to analyse PAKE protocols is usually that of Bellare, Pointcheval and Rogaway, or some variant of it. One main idea of that model is to carefully manage the adversary's ability to perform active attacks but allow a large number of passive attacks. More details of such models were presented in Chap. 2.

There are several different architectural arrangements in which PAKE protocols can be set, and which we will use later in this chapter to categorise protocols. The following are the main features.

The number of parties. By far the majority of known protocols apply to the case of a user and a server who share a password and want to set up a session key between them; we call this *two-party PAKE*. Another fairly common arrangement is where two different users share a password with a server, which helps the users to authenticate each other and share a session key between them. We call this *three-party PAKE*. Three-party PAKE can be generalised to the situation where the two users share a password with different servers; this is usually known as the *cross-realm* scenario but could also be called *four-party PAKE*. Finally there are protocols where a group of any number of users share a password and wish to set up a shared session key; we call this *group PAKE*.

Server storage of passwords. Many protocols have variants in which the server stores only an image of the client password, rather than the password itself. This means that compromise of the server does not automatically compromise the password; however, it will allow a brute force searching attack, so this is not always a significant advantage. Such variants are often called *augmented* PAKE protocols.

Use of server public keys. The situation in which the clients have access to a valid public key of the server seems natural in some scenarios. There are a number of protocols which use this feature in combination with various of the alternatives mentioned above.

The majority of the proposed protocols use Diffie–Hellman-based key agreement, with the shared password used for authentication purposes. We therefore require notation similar to that used in Chap. 5. The notation used in this chapter is shown in Table 8.1. As in Chap. 5, we describe all the protocols in this chapter in the context of subgroups of $\mathbb{Z}_p^*$, but many of them can be generalised to elliptic curve groups. Many of the protocols examined in this chapter are client–server protocols for which the actions of the client are different from those of the server. In order to avoid confusion, we always assume that principal A is the client and principal B is the server.

In the next section we examine in some detail the version of EKE based on Diffie–Hellman key exchange. This includes (Sect. 8.2.2) the EKE variant in which the server stores only an image of the password: the *augmented* version of EKE. Sections 8.3 and 8.4 examine a variety of two-party PAKE protocols, first in the scenario where the server stores the plain password and then in the augmented scenario. In Sect. 8.5, some of the few PAKE protocols based on RSA are described. Then, in Sect. 8.6, we look at three-party PAKE protocols, and Sect. 8.7 covers group PAKE

Table 8.1: Notation for password-based protocols

π	A key of short length, such as a password, owned by A.
p	A large prime (usually at least 2048 bits).
q	A prime with $q \mid p-1$.
$\mathbb{G}$	A subgroup of $\mathbb{Z}_p^*$. $\mathbb{G}$ is often a subgroup of order q, but sometimes is equal to $\mathbb{Z}_p^*$.
g	A generator of $\mathbb{G}$.
r_A, r_B	Random integers, typically of the same size as the order of $\mathbb{G}$, chosen by A and B, respectively.
t_A, t_B	g^{r_A} and g^{r_B}, respectively. All computations take place in $\mathbb{Z}_p$.
x_A, x_B	The private long-term keys of A and B respectively.
y_A, y_B	The public keys of A and B, g^{x_A} and g^{x_B}, respectively. These public keys will have to be certified in some standard way, which we do not consider here.
$\mathbf{Z}$	The shared secret calculated by the principals.
$\mathbf{K}$	The derived session key.
S_{AB}	The static Diffie–Hellman key of A and B.
$H_i(.)$	One-way hash functions. The functions $H_1, H_2, \ldots$ are often assumed to be independent random functions. Certain protocols may require specific properties and may specify particular functions.

protocols Throughout the chapter we make a comparison of the properties achieved and resources used in a selection of the protocols.

8.2 Encrypted Key Exchange Using Diffie–Hellman

In this section we examine the Encrypted Key Exchange (EKE) protocol of Bellovin and Merritt [84] as applied to Diffie–Hellman key exchange. Although the original protocol has potential weaknesses and lacks a proof of security, it is instructive to understand what may go wrong with such protocols: completely different attacks are possible from those applicable to protocols with strong keys. We later examine the many variants and extensions that have subsequently been developed from the original idea. Steiner *et al.* [690] gave a specification of how to integrate EKE into the TLS protocol, including details of how to use the symmetric encryption algorithm.

8.2.1 Bellovin and Merritt's Original EKE

The general idea of EKE is to transport ephemeral public keys encrypted using the password as a shared key. Only parties that know the password should be able to complete the protocol. This idea can be applied to ephemeral keys from different

public key schemes. Here we consider only Diffie–Hellman key exchange in which both principals choose an ephemeral public key; the password is used to encrypt the ephemeral keys as shown in Protocol 8.1.

Shared information: Shared password π. Security parameter L.

A		B
$r_A \in_R \mathbb{Z}_p$		
$t_A = g^{r_A}$	$\xrightarrow{ID_A, \{t_A\}\pi}$	$r_B \in_R \mathbb{Z}_p$
		$\mathbf{Z} = t_A^{r_B}$
		$t_B = g^{r_B}$
		$n_B \in_R \{1,\ldots,2^L\}$
	$\xleftarrow{\{t_B\}\pi, \{n_B\}_{\mathbf{K}}}$	
$\mathbf{Z} = t_B^{r_A}$		
$n_A \in_R \{1,\ldots,2^L\}$		
	$\xrightarrow{\{n_A, n_B\}_{\mathbf{K}}}$	Verify n_B
Verify n_A	$\xleftarrow{\{n_A\}_{\mathbf{K}}}$	

Protocol 8.1: Diffie–Hellman-based EKE protocol

As in basic Diffie–Hellman key agreement, the shared secret is $\mathbf{Z} = g^{r_A r_B}$, although the key derivation function to obtain the session key $\mathbf{K}$ from $\mathbf{Z}$ is not specified. Protocol 8.1 requires two exponentiations by each party, which is the same as in ordinary Diffie–Hellman key exchange. We will see below that there are versions with fewer message passes.

Symmetric Encryption Algorithm

The choice of the symmetric encryption algorithm using π was left flexible by Bellovin and Merritt. They suggested that many choices were acceptable, even ones that were 'quite weak'. However, it now seems clear that the use of encryption functions without special properties prevents security proofs being obtained and allows attacks in certain cases.

Bellovin and Merritt introduced the notion of *partition attacks* against EKE. The idea is that an adversary guessing the password can attempt to decrypt $\{t_A\}\pi$ and $\{t_B\}\pi$ and examine whether the resulting plaintext is a valid Diffie–Hellman ephemeral value. If not, then the guessed password is incorrect and can be discarded. Given several runs of the protocol, successive 'partitions' of the passwords into valid and invalid sets may be obtained. The success of partition attacks can depend on two factors: the symmetric encryption used, and the parameters defining the group $\mathbb{G}$ in which the protocol works.

Consider an example in which the symmetric encryption algorithm is a conventional block cipher (the key can be derived from the password using a suitable hash function). If $\mathbb{G} = \mathbb{Z}_p^*$ then the decryption of $\{t_A\}_\pi$ using a candidate password can be assumed to be a random string of the appropriate block length. This string must be interpreted as an element of $\mathbb{Z}_p^*$. During encryption, any padding that must be included owing to the algorithm block length can be chosen randomly (as suggested by Bellovin and Merritt). Even so, there will be some bitstrings that cannot occur as plaintexts and these allow some passwords to be discarded. This is because if decryption with a candidate password results in a string whose value is greater than p then that candidate is invalid.

In order to make partition attacks harder, Bellovin and Merritt suggested choosing p to be slightly less than a power of 2. In this way very few candidate passwords will be invalid. Jaspan [396] conducted an analysis which concluded that it is adequate in practice to ensure that $1-(p/2^n) < 10^{-4}$, where 2^n is the smallest power of 2 greater than p. However, it seems preferable to choose the symmetric encryption algorithm to be matched to the group so as to completely eliminate such attacks; we will explain below how this may be done.

Omitting Symmetric Encryption

Bellovin and Merritt suggested that either of the two encryptions using π in Protocol 8.1 could be omitted. Subsequent authors have shown that this can result in weaknesses, depending on the precise format of the messages used for authentication.

First, suppose that the encryption using π is omitted from the first message so that this message becomes A, t_A. This change does not appear to help a passive adversary and still prevents $\mathbf{K}$ from being found without knowledge of π. However, Patel [603] showed how an active adversary masquerading as A may be able to exploit this change. The adversary can send the first message by choosing r_C and setting $t_A = g^{r_C}$. Now, if the authenticator $\{n_B\}_\mathbf{K}$ returned in the second message contains redundancy, then the adversary can use it to eliminate any potential password $\tilde{\pi}$ by decrypting $\{t_B\}_\pi$ with $\tilde{\pi}$ to obtain $\tilde{t}_B$, finding the corresponding value of the session key $\tilde{\mathbf{K}}$ from $\tilde{\mathbf{Z}} = (\tilde{t}_B)^{r_C}$, and checking for the redundancy after decrypting $\{n_B\}_\mathbf{K}$ with $\tilde{\mathbf{K}}$. Jablon [388] noted that small subgroup attacks (see Sect. 5.2.1) may also become possible in this situation; the adversary can move t_A into any small subgroup and use the authenticator in the third message to identify the correct $\mathbf{K}$ in a brute force attack. Standard precautions against small subgroup attacks can prevent this.

If encryption using π is omitted from the second message then the adversary can attempt to masquerade as B. Steiner *et al.* [691] showed that this allows an attack if the authenticators *do not* contain redundancy. The adversary chooses a random X and r_C, sets $t_B = g^{r_C}$, and sends t_B, X as the second message, which will be accepted by A. Now the adversary can decrypt $\{t_A\}_\pi$ with a candidate password $\tilde{\pi}$ to obtain $\tilde{t}_A$ and the corresponding session key $\tilde{\mathbf{K}} = (\tilde{t}_A)^{r_C}$. The adversary can then decrypt the third message using $\tilde{\mathbf{K}}$ and check whether the second field equals the decryption of X using $\tilde{\mathbf{K}}$, in order to confirm whether $\tilde{\pi}$ is the correct password. Patel [603] also noted an attack here when there is limited redundancy in the authenticator, but this time it

is necessary for the adversary to wait to see if a random authenticator is accepted by A: the adversary sends t_B, X as the second message for a random value X. If this message is accepted then the adversary knows that the redundancy is present, but if it is not then all passwords that result in the desired redundancy can be eliminated.

It may seem reasonable to conclude that it is safest not to omit either of the symmetric encryptions using π. However, a better solution is to be more careful in the use of authenticators. We now examine variants in which this is done; not only is the result a protocol with fewer rounds, but also a proof of security becomes feasible.

Security Proofs Using Generic Encryption

Bellare *et al.* [70] have provided proofs that the exchange of the password-encrypted Diffie–Hellman messages at the core of EKE is a secure protocol using the Bellare–Pointcheval–Rogaway (BPR00) model discussed in Chap. 2. Although they specify the key derivation function to compute the session key, the symmetric encryption algorithm used in the protocol is not concretely defined; it is assumed to be an *ideal cipher* that can be regarded as a random function (analogous to the functions used in the random oracle model). Furthermore, the decryption function must take strings of a fixed size and map them to elements of $\mathbb{G}$. This may leave the implementer with some difficulty in choosing an appropriate concrete algorithm. Zhao *et al.* [777] showed that certain concrete instantiations of the ideal cipher can lead to concrete attacks.

Bellare and Rogaway [79] also defined authenticated versions of their abstract protocol, collectively called AuthA, but without a separate security proof. Later, Bresson *et al.* [152] provided new proofs for AuthA in which the encryption is simply multiplication by the password. They assumed a random oracle for the authentication and key derivation functions, and otherwise assumed only the difficulty of the CDH problem.

8.2.2 Augmented EKE

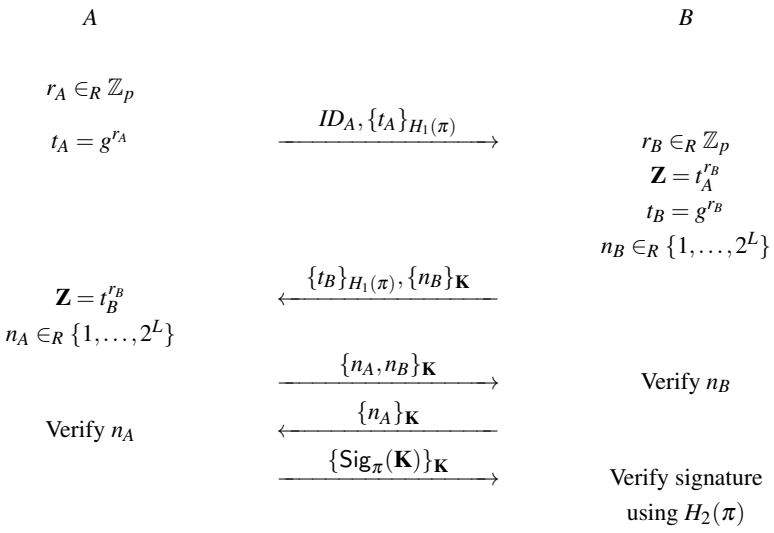
For many years it has been standard practice for passwords to be stored on servers as the image of a one-way function H, so that $H(\pi)$ is stored rather than π itself. In this way, compromise of the password file will not compromise the passwords directly, and when a claimed password π' is submitted to the server it can be checked whether $H(\pi') = H(\pi)$. However, compromise of the password file will nevertheless allow an offline dictionary attack to be mounted, since the adversary can calculate $H(\pi')$ for any password guess π' and compare it with the stored value. Therefore the benefits of this approach will depend on the application. Many designers have provided password-based key establishment protocols that require the server to store only an image of each password.

A number of protocols explicitly include the use of *salt* in the calculation of the password image. Salt is another commonly used mechanism for protecting passwords on a server. For each password π_i, a random salt value s_i is chosen and the pair $(s_i, H(\pi_i, s_i))$ can be stored on the server. The purpose of salt is to frustrate bulk

guessing attacks if the password file stored on the server becomes compromised. Even though the salt becomes known to the adversary, any password guess can only be verified against one password image at a time.

The original EKE protocol requires that the server stores the plaintext password in order to be able to decrypt the protocol exchanges. Subsequently, Bellovin and Merritt [85] designed an 'augmented' version, shown in Protocol 8.2, which requires the server to store only an image of the password.

Information held by A: Password π.
Information held by B: Two images of password, $H_1(\pi)$ and $H_2(\pi)$. $H_2(\pi)$ is suitable for signature verification.
Shared information: Security parameter L.



Protocol 8.2: Augmented Diffie–Hellman-based EKE protocol

The first four messages of this protocol are the same as in Protocol 8.1 with the plain password π replaced by the hashed password $H_1(\pi)$. These messages are insufficient on their own, since the user would require only $H_1(\pi)$ in order to complete the protocol and there would be no effective difference from the original EKE. Therefore a fifth message is added, which consists of a signature of the shared secret constructed using π as the secret key and so that B can hold the corresponding public key. Bellovin and Merritt did not make an explicit choice for the signature, but an example could be an ElGamal signature [267] with private key π and public verification key $H_2(\pi) = g^{\pi}$. (It is possible to allow $H_1 = H_2$ but if, as in this example, H_2 requires an exponentiation, computation is reduced for A if H_1 is a simple hash function.)

Steiner *et al.* [691] have pointed out that an adversary who is able to obtain an old **K** can decrypt the final message flow of Protocol 8.2 and attempt to verify the signature using a guessed π' value and therefore mount a dictionary attack. This protocol is therefore arguably weaker than the original in the usual scenario where old session keys may be compromised. However, this problem could be avoided by using a different key derived from **Z** to protect the protocol messages, instead of using the session key **K** for this purpose.

8.3 Two-Party PAKE Protocols

After the Encrypted Key Exchange protocol was developed by Bellovin and Merritt, many new PAKE protocols were designed, aiming to find potential improvements over EKE. This section looks at the prominent examples of two-party PAKE protocols. Consideration of augmented PAKE protocols is deferred to Sect. 8.4, but it is worth noting that many protocols in this section have a corresponding augmented version. The protocols in this section do not provide protection against server compromise.

8.3.1 PAK

Boyko *et al.* [145] specified a variant of Diffie–Hellman-based EKE called PAK (Password Authenticated Key exchange). Protocol 8.3 shows the basic version of PAK.

A key feature of Protocol 8.3 is that the element P in the group $\mathbb{G}$ is derived from the password π using a special hash function: $P = H_1(ID_A, ID_B, \pi)$. In typical applications the computing device of A will not have the value π available beforehand, and so this calculation is part of the computational requirements for A. (In contrast, we assume that B is a server which can store the derived value of P.)

The original version of PAK [145] uses a prime p of the form $p = rq + 1$, with r and q relatively prime. The Diffie–Hellman key exchange takes place in the subgroup $\mathbb{G}$ of order q. Computation of H_1 then involves an exponentiation with exponent r in order to move a bitstring into the subgroup $\mathbb{G}$. Since the typical lengths of q and r could be 256 bits and 1792 bits, respectively, this calculation is more expensive than the subsequent Diffie–Hellman exchange. (The computational requirements may be optimised for different applications by choosing a larger size for q so that the size of r becomes smaller.)

The original PAK protocol uses three other hash functions. It is important that these functions are different, and the formal proof assumes they are independent random functions. In practice, we may define the different hash functions by prepending a different fixed string to the input and then using a standard hash function. For example, we may define $H_i(x) = H(\text{ASCII}(i), x)$, where H is SHA-2 [579] and $\text{ASCII}(i)$ is the ASCII representation of i.

The PAK protocol was proven by Boyko *et al.* [145] to be secure in Shoup's simulation model assuming the hash functions act as random oracles. Originally this

Shared information: Generator g of $\mathbb{G}$ of order q. Hash functions H_1, H_2, H_3, H_4.
Information held by A: Password π.
Information held by B: Derived password image $P = H_1(ID_A, ID_B, \pi) \in G$.

A		B
$P = H_1(ID_A, ID_B, \pi)$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A P$	$\xrightarrow{m}$	Check $m \neq 0 \in G$
		$t_A = m/P$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$\mathbf{Z} = t_A^{r_B}$
$\mathbf{Z} = t_B^{r_A}$	$\xleftarrow{t_B, k}$	$k = H_2(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$
$k \stackrel{?}{=} H_2(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$		
$k' = H_3(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$	$\xrightarrow{k'}$	$k' \stackrel{?}{=} H_3(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$
$\mathbf{K} = H_4(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$		$\mathbf{K} = H_4(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$

Protocol 8.3: PAK protocol

result relied on the decision Diffie–Hellman assumption, but later MacKenzie [509] provided proofs in the Bellare–Pointcheval–Rogaway (BPR00) model and showed that the ordinary (computational) Diffie–Hellman assumption is sufficient.

We may regard the PAK protocol as a variant of the original Diffie–Hellman-based EKE protocol with the following instantiations and modifications:

- the key derivation function is specified;
- symmetric encryption is defined as multiplication in $\mathbb{G}$ by the image P of π;
- symmetric encryption of the ephemeral Diffie–Hellman key in the second message is omitted;
- a simplified authentication mechanism is used which allows a more efficient protocol.

It is interesting to consider how these changes ensure that the potential weaknesses in Diffie–Hellman EKE are avoided. The choice of symmetric encryption is very simple and matched to the group $\mathbb{G}$. (Although a modular multiplication may be computationally more expensive than encryption with a block cipher, in practice the additional effort over the required exponentiations is very small.) In the partition attack described in Sect. 8.2.1, the adversary attempts decryption with candidate passwords π'. With this natural choice of encryption, every π' maps to an image $P' \in G$ and attempted decryption of $t_A P$ results in $t_A P/P'$ which will always lie in $\mathbb{G}$. Therefore the partition attack cannot even discount a single candidate π'. If the

adversary tries to exploit the omission of the symmetric encryption with π in the second message by sending t_B, X for a random X then, with overwhelming probability, A will reject the message. Instead, the adversary can calculate a correct value k for any candidate password, and this would allow checking of that one password.

The PAK protocol has appeared in various standards, including IEEE P1363 [373]. There are also versions in ITU [386] and IETF [163] standards, but although these versions are essentially identical to each other, there are small differences from Protocol 8.3 as specified in the academic paper [145]. In particular, in the ITU [386] and IETF [163] versions both A and B multiply their Diffie–Hellman value by a hashed version of the password. Although both standards claim that their version has been proven secure, the published proof [145] does not, strictly speaking, apply to this variant. A Russian standard known as Security Evaluated Standardized Password-Authenticated Key Exchange, or SESPAKE, is a PAK variant and is described in RFC 8133 [683].

Boyko *et al.* [145] also specified a variant of PAK without explicit authentication. The result is shown as Protocol 8.4 and is known as PPK (Password Protected Key exchange). Apart from omitting the authenticators, another difference is that both messages, instead of just the first, must now use the symmetric encryption with π. A consequence of using two encryptions is that A requires an extra exponentiation, in comparison with Protocol 8.3, to derive the extra password image P_2.

Shared information: Generator g of $\mathbb{G}$ of order q. Hash functions H_1, H_2, H_3.
Information held by A: Password π.
Information held by B: $P_1 = H_1(ID_A, ID_B, \pi)$ and $P_2 = H_2(ID_A, ID_B, \pi)$ both in $\mathbb{G}$.

A		B
$P_1 = H_1(ID_A, ID_B, \pi)$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A P_1$	$\xrightarrow{m}$	Check $m \neq 0 \in G$
		$t_A = m/P_1$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
Check $m' \neq 0 \in G$	$\xleftarrow{m'}$	$m' = t_B P_2$
$P_2 = H_2(ID_A, ID_B, \pi)$		
$t_B = m'/P_2$		
$\mathbf{Z} = t_B^{r_A}$		$\mathbf{Z} = t_A^{r_B}$

$$\mathbf{K} = H_3(ID_A, ID_B, m, m', \mathbf{Z}, P_1)$$

Protocol 8.4: PPK protocol

If the first message alone was protected by the password, as in Protocol 8.3, then an adversary could masquerade as B and use knowledge of the derived session key to mount a brute force searching attack. For example, the adversary could choose random $r_C \in \mathbb{Z}_q$ and send g^{r_C} to A as the second message. Then the shared secret accepted by A is $\mathbf{Z} = g^{r_C r_A}$. The adversary can check any potential password $\tilde{\pi}$ by calculating $\tilde{P}_1 = H_1(ID_A, ID_B, \tilde{\pi})$, $\tilde{t}_A = m/\tilde{P}_1$ and the corresponding secret $\tilde{\mathbf{Z}} = (\tilde{t}_A)^{r_C}$. If $\tilde{\pi}$ is the correct password then $\tilde{\mathbf{Z}} = \mathbf{Z}$, which can be checked against subsequent messages from A protected with $\mathbf{K}$.

MacKenzie [507, 509] designed several other enhancements and variations of the PAK protocol and proved their security. These include allowing different groups instead of $\mathbb{Z}_p^*$, such as elliptic curve groups and the XTR group [481], and removing identifiers from the inputs to H_1 (and H_2 for PPK) to simplify the protocol. MacKenzie and Patel [511] showed that short Diffie–Hellman exponents can be chosen to improve efficiency even when the group $\mathbb{G}$ is large. For example, by choosing $\mathbb{G} = \mathbb{Z}_p^*$ so that $q = p - 1$, a simple H_1 can be used but the Diffie–Hellman exponents can be much shorter than $|p|$.

8.3.2 SPEKE

The SPEKE (Secure Password Exponential Key Exchange) protocol was designed by Jablon [388]. Although SPEKE, like EKE, is based on Diffie–Hellman, it has a distinctive feature that the password is used to define the base to be used for the Diffie–Hellman exchange. Jablon presented both basic and 'fully constrained' versions, the latter designed to prevent a variety of different attacks.

Protocol 8.5 describes the fully constrained SPEKE protocol, but we have included the verifiers V_A and V_B, as defined in the version analysed by MacKenzie [508]. Jablon's original verifiers included only $\mathbf{Z}$ in the hash, which allows a potential attack [772] in which the adversary masquerades as A and makes a guess $\tilde{\pi}$ of the password π. The received verifier V_B then allows the adversary to check if $\tilde{\pi}$ is any value in the set $\{\pi, \pi^2, \pi^3, \ldots\}$. The seriousness of such an attack can be debated, since there may be few potential passwords in this sequence, but MacKenzie's verifiers rule it out altogether.

The prime p and subgroup $\mathbb{G}$ are chosen so that $q = (p-1)/2$ is prime and $\mathbb{G}$ is of order q. The password π is regarded as a number in $\mathbb{Z}_p^*$, and then $P = \pi^2$ is guaranteed to lie in $\mathbb{G}$ and have order q (assuming that π is not equal to 1, -1, or 0). The value P is used as the generator of $\mathbb{G}$ for protocol runs involving π. Apart from this special way of defining the group generator, the protocol is exactly the basic Diffie–Hellman key exchange with key confirmation. The shared secret is therefore $\mathbf{Z} = P^{2r_A r_B}$ and forward secrecy is provided. The check that $\mathbf{Z}$ is not in the set $\{-1, 0, 1\}$ ensures that small subgroup attacks are avoided. In comparison with Protocol 8.3, calculation of P from π is much simpler, so that we can almost ignore it in assessing A's computational requirements. The cost of the Diffie–Hellman exchange calculations depends on the size of the exponents r_A and r_B. Typical lengths might be 256 bits.

The original version of SPEKE does not have a proof of security, but later MacKenzie [508] did provide a proof of a slight variant using Shoup's simulation

Shared information: Subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Shared derived password image $P = \pi^2$ where π is interpreted as an element of $\mathbb{Z}_p^*$. Security parameter L. Hash functions H_1, H_2, H_3.

A		B
$r_A \in_R \{1,\ldots,2^L\}$		
$t_A = P^{r_A}$	$\xrightarrow{t_A}$	$r_B \in_R \{1,\ldots,2^L\}$
		$\mathbf{Z} = t_A^{2r_B}$
		Check $\mathbf{Z} \notin \{-1,0,1\}$
		$t_B = P^{r_B}$
$\mathbf{Z} = t_B^{2r_A}$	$\xleftarrow{t_B, V_B}$	$V_B = H_1(t_A, t_B, \mathbf{Z}, \pi)$
Check $\mathbf{Z} \notin \{-1,0,1\}$		
Verify V_B		
$V_A = H_2(t_A, t_B, \mathbf{Z}, \pi)$	$\xrightarrow{V_A}$	Verify V_A
	$\mathbf{K} = H_3(\mathbf{Z})$	

Protocol 8.5: SPEKE protocol

model. This proof relies on the difficulty of a non-standard decision problem called the *inverted-additive Diffie–Hellman problem*. It shows that at most two passwords can be tested per login attempt, whereas it is proven that only one password test is possible for the PAK protocol. The differences between the protocol used by MacKenzie and Protocol 8.5 are:

- P is defined to include the identities of A and B: $P = (H(ID_A, ID_B, \pi))^2$;
- the hashes used to form the session key include the identities of A and B, the exchanged messages t_A and t_B, the password π, and $\mathbf{Z}$;
- the exponents are chosen randomly in $\mathbb{Z}_q$.

An important practical consequence of the last difference is that exponents are the same size as q. Since $q = (p-1)/2$, this would typically make the exponents around 2048 bits, a significant overhead compared with the exponents of size 160 bits originally suggested by Jablon. Versions of SPEKE suitable for use on elliptic curves, which are potentially much more efficient, are specified in the IEEE P1363.2 standard [373] and the ISO 11770-4 standard [385].

Hao and Shahandashti [347] described two attacks which apply to certain versions of SPEKE, including a version in an older version of the ISO 11770-4 standard [379]. The first attack is an impersonation attack in which the adversary replays a (randomised) message back to the victim. In the second attack the adversary is able to randomise the agreed key without obtaining it, disturbing the conversations

seen by the principals so that they no longer match. Both attacks can be prevented by including sufficient details of the parties and protocol components in the protocol messages. The SPEKE version shown in Protocol 8.5 is not vulnerable to these attacks, owing to the inclusion of the t_A and t_B values in the verifiers V_A and V_B. The version in the later ISO 11770-4 standard [385] includes the t_A and t_B values and the principal identities in the session key derivation function in order to prevent the attack.

SPEKE Variants

Pointcheval and Wang [617] analysed a variant of SPEKE called *two-basis password exponential key exchange* (TBPEKE), in which the basis used for the Diffie–Hellman exchange is $U \cdot V^{\pi}$ for two random group elements U and V. Their security proof relies on assumptions which they believe can be satisfied by elliptic curves, thus allowing instantiations more efficient than those satisfying the proof assumptions of MacKenzie [508].

A flawed variant of SPEKE appeared in the original paper of Jablon [388]. In this variant the derived password is $P = g^{\pi}$, which, as in Protocol 8.5, is used as the generator of the group $\mathbb{G}$ in a Diffie–Hellman exchange. The shared secret is again $\mathbf{Z} = P^{r_A r_B}$. A password-guessing attack was found by Jablon and others and resulted in the deletion of this option from later papers on SPEKE [389]. An essentially identical protocol also appears in the paper of Bakhtiari *et al.* [50]. A protocol designed by Seo and Sweeney [663], known as SAKA (Simple Authenticated Key Agreement), was an unfortunate rediscovery of almost the same protocol. The main difference is that the shared secret is $\mathbf{Z} = g^{r_A r_B}$, but this seems to have little influence. Mitchell [557] showed that a password-guessing attack is still possible. A number of other authors [458, 495, 715] have also found various attacks concerned with the explicit authentication exchange phase of SAKA.

A protocol by Kaufman and Perlman [421] is connected with SPEKE in that the password is used to generate the parameters for the Diffie–Hellman exchange. However, the difference is that in this case the password is used as a seed to generate the prime modulus p, and this is done in such a way that 2 is a generator (at least with high probability) of $\mathbb{Z}_p^*$. The protocol is known as PDM (for Password Derived Moduli). Like SPEKE, it is essentially basic Diffie–Hellman with key confirmation. One computational overhead is the time taken to reconstruct p at both ends, which entails testing for primality (indeed, testing for a strong prime), starting from the seed value. In addition, no formal security proof was provided.

8.3.3 Dragonfly Protocol

A protocol, originally known as Simultaneous Authentication of Equals (SAE) and later known as Dragonfly, was designed by Harkins for use in sensor networks [350]. Like SPEKE (Protocol 8.5), Dragonfly uses the shared password to define the base used for a Diffie–Hellman key exchange.

Shared information: Subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Hash functions H_1, H_2.
Shared derived password image $P = H_1(ID_A, ID_B, \pi)$, where H_1 maps into $\mathbb{G}$.

A		B
$r_A, m_A \in_R \mathbb{Z}_q$		
$s_A = r_A - m_A$		
$t_A = P^{m_A}$	$\xrightarrow{t_A, s_A}$	Check that $t_A \in \mathbb{G}, s_A \in \mathbb{Z}_q^*$
		$r_B, m_B \in_R \mathbb{Z}_q$
		$s_B = r_B - m_B$
	$\xleftarrow{t_B, s_B}$	$t_B = P^{m_B}$
Check that $t_B \in \mathbb{G}, s_B \in \mathbb{Z}_q^*$		
$\mathbf{Z} = (P^{s_B} t_B)^{r_A}$		$\mathbf{Z} = (P^{s_A} t_A)^{r_B}$
$V_A = H_2(\mathbf{Z}, t_A, s_A, t_B, s_B)$	$\xrightarrow{V_A}$	Verify V_A
Verify V_B	$\xleftarrow{V_B}$	$V_B = H_2(\mathbf{Z}, t_B, s_B, t_A, s_A)$
$\mathbf{K} = H_2(\mathbf{Z}, t_A \cdot t_B, s_A + s_B \bmod q)$		

Protocol 8.6: Dragonfly protocol

In order to emphasise the similarity with SPEKE, in Protocol 8.6 we have adjusted the details slightly by changing the signs of m_A and m_B in the computation of s_A and s_B and by splitting the computation of V_A, V_B and $\mathbf{K}$ into three separate operations. It is also possible to convert the protocol into a three-message protocol by allowing B to compute verifier V_B earlier and sending it with the second message. The shared secret is $\mathbf{Z} = P^{(s_B+m_B)(s_A+m_A)}$.

The requirement to check that the elements t_A, t_B, s_A, s_B are in their correct groups was missing originally [350] but was added later [352] because of a small subgroup attack identified by Clarke and Hao [219] when these checks were missing. Clarke and Hao also provided a detailed efficiency comparison with SPEKE (Protocol 8.5), which indicates that even after adding these checks Dragonfly is the more efficient of the two, mainly owing to the groups in which they are specified to operate.

Lancrenon and Škrobot [472] provided a security proof (including forward secrecy) for Protocol 8.6 in the BPR00 model. The proof uses random oracles and assumes the difficulty of the CDH problem. As in their specification, we have assumed that the mapping of the password into the group element P is done using a random oracle. In the original version [352, 350] a concrete function, referred to as *hunting and pecking*, was used for this purpose.

8.3.4 SPAKE

The Simple Password-based Key Exchange (SPAKE) protocol was introduced by Abdalla and Pointcheval [18] with the aim of reducing (but not eliminating) the use

of random oracles in the security proof. A structural difference from other EKE-based protocols is that the password is used as an exponent for the public value associated with its owner, but otherwise the encryption of the Diffie–Hellman public key is similar to Protocol 8.4. Protocol 8.7 shows the exchanged messages. Note that the password must be interpreted as an element in $\mathbb{Z}_q$.

Public information: Hash functions H. Public values $M_A, M_B \in \mathbb{G}$ with $q = |\mathbb{G}|$.
Information shared by A and B: Password $\pi \in \mathbb{Z}_q$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A M_A^{\pi}$	$\xrightarrow{m}$	$t_A = m/M_A^{\pi}$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
	$\xleftarrow{m'}$	$m' = t_B M_B^{\pi}$
$t_B = m'/M_B^{\pi}$		
$\mathbf{Z} = t_B^{r_A}$		$\mathbf{Z} = t_A^{r_B}$

$$\mathbf{K} = H(ID_A, ID_B, m, m', \pi, \mathbf{Z})$$

Protocol 8.7: SPAKE protocol

Abdalla and Pointcheval [18] proved the security of Protocol 8.7 in the BPR00 model, relying on the computational Diffie–Hellman assumption and taking H as a random oracle. They defined two SPAKE variants. Protocol 8.7 is actually what they called SPAKE2, while SPAKE1 looks identical except that π is absent in the definition of $\mathbf{K}$. SPAKE1 relies on different assumptions but was only shown to be secure in a non-concurrent setting (where only one instance of the protocol can be run at any one time). SPAKE2 (Protocol 8.7) is secure in the usual concurrent setting.

Note that the values M_A and M_B are fixed values, so that the computation of the values of m and m' need only be done once. Thus Protocol 8.7 is very efficient, requiring only two exponentiations for each principal. Abdalla and Pointcheval [18] credited Kobara and Imai [435, 436] with a similar prior protocol which works like Protocol 8.7 but uses a MAC to derive the key, instead of a random oracle. Abdalla and Pointcheval pointed out that in order for this to be secure special properties of the MAC are required, and they therefore questioned the claim that a proof without random oracles can be practically achieved.

8.3.5 J-PAKE

A different approach to using the shared password is applied in the J-PAKE (Juggling PAKE) protocol of Hao and Ryan [346]. One of the motivations for designing J-PAKE was that many other PAKE protocols are covered by patents. J-PAKE has been deployed in a number of real-world products and was standardised in the ISO 11770-4 standard [385].

As shown in Protocol 8.8, the protocol runs in two rounds, so the first pair and second pair of messages can be sent in either order. This means that the protocol can also run in three messages and three rounds by combining the second and fourth messages into one.

Information shared by A and B: Password $\pi \in \mathbb{Z}_q$.

A		B
$r_A, r'_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}, t'_A = g^{r'_A}$		$r_B, r'_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}, t'_B = g^{r'_B}$
	$\xrightarrow{t_A, t'_A, KP(r_A), KP(r'_A)}$	
	$\xleftarrow{t_B, t'_B, KP(r_B), KP(r'_B)}$	
Check $KP(r_B), KP(r'_B)$ $\alpha = (t_A t_B t'_B)^{r'_A \pi}$		Check $KP(r_A), KP(r'_A)$ $\beta = (t_B t_A t'_A)^{r'_B \pi}$
	$\xrightarrow{\alpha, KP(r'_A \pi)}$	
	$\xleftarrow{\beta, KP(r'_B \pi)}$	
Check $KP(r'_B \pi)$ $\mathbf{Z} = (\beta {t'_B}^{-r'_A \pi})^{r'_A}$		Check $KP(r'_A \pi)$ $\mathbf{Z} = (\alpha {t'_A}^{-r'_B \pi})^{r'_B}$

Protocol 8.8: J-PAKE protocol

The shared secret is the value $\mathbf{Z} = g^{(r_A + r_B) r'_A r'_B \pi}$. Knowledge proofs are used to show that the respective senders know the discrete logs of t_A, t'_A, t_B, t'_B with respect to base g, and the discrete logs of α and of β with respect to bases $t_A t_B t'_B$ and $t_B t_A t'_A$. While no concrete instantiation was required in the original specification, Schnorr signatures were suggested as suitable. Note the similarity with the YAK protocol (Protocol 5.40) regarding usage of proofs of knowledge.

Although Hao and Ryan [346] provided a security analysis of Protocol 8.8, they did not provide a security reduction in any known computational model. Later, Abdalla *et al.* [9] provided such a proof in the BPR00 model, assuming the difficulty of the decisional square Diffie–Hellman assumption. They remarked that the usage of Schnorr signatures as the proof of knowledge in the protocol causes technical prob-

lems in the proof, which they resolved by assuming a so-called *algebraic adversary* with restricted abilities. Adding the signer identity to the Schnorr signatures was recommended by Hao and Ryan [346] and assumed in the security proof of Abdalla *et al.* [9]. Toorani [712] pointed out that unknown key-share attacks are possible if this is not done.

In comparison with many other PAKE protocols. J-PAKE is somewhat inefficient, largely owing to the use of the proofs of knowledge, which add one exponentiation for proof generation and two for proof verification (the latter can be combined into a multi-exponentiation). Lancrenon *et al.* [471] proposed two more efficient variants of Protocol 8.8, removing one of the pair of ephemeral values in each of the first pair of messages. This saves four exponentiations on each side.

Public information: Full domain hash function H.
Information shared by A and B: Password $\pi \in \mathbb{Z}_q$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
	$\xrightarrow{t_A, KP(r_A)}$	
	$\xleftarrow{t_B, KP(r_B)}$	
Check $KP(r_B)$ $D = H(ID_A, ID_B, t_A, t_B)$ $\alpha = (Dt_B)^{r_A \pi}$		Check $KP(r_A)$ $D = H(ID_A, ID_B, t_A, t_B)$ $\beta = (Dt_A)^{r_B \pi}$
	$\xrightarrow{\alpha, KP(r_A \pi)}$	
	$\xleftarrow{\beta, KP(r_B \pi)}$	
Check $KP(r_B \pi)$ $\mathbf{Z} = (\beta {t_B}^{-r_A \pi})^{r_A}$		Check $KP(r_A \pi)$ $\mathbf{Z} = (\alpha {t_A}^{-r_B \pi})^{r_B}$

Protocol 8.9: J-PAKE variant of Lancrenon, Škrobot and Tang

Protocol 8.9 shows one of the two variants, which Lancrenon *et al.* [471] called RO-J-PAKE. As its name implies, it requires that the hash function H is modelled as a random oracle. Their other variant, called CRS-J-PAKE, replaces D with a common reference string. These differences influence the security proofs, but the computational requirements are the same except for the hash computation for D. Lancrenon *et al.* [471] provided security proofs in the BPR00 model following the proofs of Abdalla *et al.* [9].

8.3.6 Katz–Ostrovsky–Yung Protocol

The protocol of Katz, Ostrovsky, and Yung (KOY) [414] has a strong theoretical significance because of its security proof. The proof of security for many PAKE protocols uses the random oracle model. From a theoretical viewpoint, eliminating such an idealisation is very desirable. The KOY protocol was proven secure in the BPR00 model without using random oracles, assuming the Decision Diffie–Hellman problem is hard. The encryption algorithm used in the protocol is a variant of the Cramer–Shoup algorithm [224], chosen because it has been proven to provide non-malleability without itself using random oracles. As shown in Protocol 8.10, there are a number of public parameters.

The first message, sent from A to B, consists of the encryption of the plaintext message g_1^{π} in the Cramer–Shoup variant. However, an amusing aspect of the protocol is that the encryption is turned around; the public parameters g_1, g_2, h, c, d form a public key for which no corresponding private key is known. This ensures that information about the password is not leaked. But B already knows the value g_1^{π}, and so can obtain the correct parameters to form the session key. The second message, returned from B, is also an encryption of g_1^{π} using a different Cramer–Shoup variant but with the same public parameters. Notice that the signature sent in the third message uses a signing key, SK, generated randomly by A at the start of the protocol, and is not intended to provide entity authentication. Indeed, neither party obtains key confirmation or entity authentication. Katz *et al.* noted that the usual additional checks are required, in particular that received values lie in the group $\mathbb{G}$.

Although the protocol of Katz *et al.* is certainly practical, its computational requirements are significantly greater than for protocols assuming random oracles, around 5–10 times as great depending on which variant is compared and how the computation is measured. There have been various optimisations of Protocol 8.10 (see also Sect. 8.3.8). Gennaro [293] designed a similar protocol which uses MACs instead of signatures to obtain a more efficient variant.

8.3.7 Protocol of Jiang and Gong

Jiang and Gong [399] designed a simplified version of the Katz–Ostrovsky–Yung protocol with a significantly reduced computational requirement. Indeed, it remains one of the most efficient PAKE protocols with a security proof in the standard model. Protocol 8.11 shows the protocol messages. The function F_{σ} denotes a member of a pseudo-random function family with key σ.

In contrast to Protocol 8.10, there is no need to generate an ephemeral signature key in Protocol 8.11, which makes use of a fixed encryption key PK (formally speaking, this is a common reference string). The number of required exponentiations for each principal is reduced from 15 in Protocol 8.10 to 4 if individual exponentiations are counted, or more modestly if multi-exponentiations are considered. This count excludes short exponentiations with the password and comes in addition to the cost of encryption/decryption (in Protocol 8.11) or the cost of signing/verification (in Protocol 8.10). In addition to saving on computation, the Jiang–Gong protocol provides mutual authentication within its three rounds, in contrast to the KOY protocol.

Shared information: Generators g_1, g_2, h, c, d of $\mathbb{G}$, where $p-1=qr$. Hash function H. Password π.

A		B
Generate ephemeral signature keys PK, SK		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g_1^{r_A}$		
$u_A = g_2^{r_A}$		
$v_A = h^{r_A} g_1^{\pi}$		
$\alpha = H(ID_A, PK, t_A, u_A, v_A)$		
$w_A = (cd^{\alpha})^{r_A}$		
	$\xrightarrow{ID_A, PK, t_A, u_A, v_A, w_A}$	
		$x_B, y_B, z_B, z'_B, r_B \in_R \mathbb{Z}_q$
		$\alpha = H(ID_A, PK, t_A, u_A, v_A)$
		$E = g_1^{x_B} g_2^{y_B} h^{z_B} (cd^{\alpha})^{z'_B}$
		$t_B = g_1^{r_B}$
		$u_B = g_2^{r_B}$
		$v_B = h^{r_B} g_1^{\pi}$
		$\beta = H(ID_B, E, t_B, u_B, v_B)$
		$w_B = (cd^{\beta})^{r_B}$
	$\xleftarrow{ID_B, E, t_B, u_B, v_B, w_B}$	
$\beta = H(ID_B, E, t_B, u_B, v_B)$		
$x_A, y_A, z_A, z'_A \in_R \mathbb{Z}_q$		
$K = g_1^{x_A} g_2^{y_A} h^{z_A} (cd^{\beta})^{z'_A}$		
	$\xrightarrow{K, \mathsf{Sig}_{SK}(\beta, K)}$	
		Verify signature
$\mathbf{Z} = E^{r_A} t_B^{x_A} u_B^{y_A} (v_B / g_1^{\pi})^{z_A} w_B^{z'_A}$		$\mathbf{Z} = K^{r_B} t_A^{x_B} u_A^{y_B} (v_A / g_1^{\pi})^{z_B} w_A^{z'_B}$

Protocol 8.10: Katz–Ostrovsky–Yung protocol

Jiang and Gong [399] proved the security of Protocol 8.11 in the BPR00 model assuming the difficulty of the DDH problem. They also required that the encryption function used provides CCA2 security.

8.3.8 Protocols Using Smooth Projective Hashing

The KOY protocol is based on the Cramer–Shoup encryption scheme [224]. Some time after the concrete Cramer–Shoup cryptosystem was published, Cramer and

Public information (common reference string): Public encryption key PK.
Shared information: Generators g, h of $\mathbb{G}$, where $p-1=qr$. Hash function H. Password π.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$v_A = h^{r_A} g^{\pi}$		
	$\xrightarrow{t_A, v_A}$	$r_B, r'_B \in_R \mathbb{Z}_q$
		$\mu = g^{r_B} h^{r'_B}$
		$P = v_A g^{-\pi}$
		$\sigma = t_A^{r_B} P^{r'_B}$
		$h = H(\mu, t_A, P, ID_B, ID_A)$
		$r = F_\sigma(3)$
		$C = \mathsf{Enc}_{PK}(h; r)$
	$\xleftarrow{\mu, C}$	
$P = h^{r_A}$		
$\sigma = \mu^{r_A}$		
$r = F_\sigma(3)$		
$h = H(\mu, t_A, P, ID_B, ID_A)$		
$C \stackrel{?}{=} \mathsf{Enc}_{PK}(h; r)$		
$t = F_\sigma(1)$	$\xrightarrow{t}$	$t \stackrel{?}{=} F_\sigma(1)$
	$\mathbf{K} = F_\sigma(2)$	

Protocol 8.11: Jiang–Gong protocol

Shoup developed an abstracted version [225] by introducing the idea of a *smooth projective hash function* (SPHF). This allowed them to find other versions of their scheme, instantiated by different SPHFs. Even if these alternatives are not more efficient than the original Cramer–Shoup cryptosystem, they rely on different computational assumptions.

The generalisation process which occurred with the Cramer–Shoup cryptosystem has been repeated in the case of PAKE protocols. A generalised version of the KOY protocol was developed by Gennaro and Lindell [296], who proposed an abstract protocol which can be instantiated by any suitable SPHF. The shared knowledge of the password enables both parties to obtain a shared secret owing to the different ways of computing using the SPHF.

An SPHF computes values in some domain which includes a distinguished *language* L. There are two ways of computing the hash on a value v; one is with the hashing key hk, denoted $H(hk; v)$, and the other is with a projected key hp in addi-

tion to a *witness* w in the language, denoted $HP(hp, w; v)$. If $v \in L$ then these two computations are equal; if not, they are almost certainly different.

For PAKE protocols the language is the set of encryptions of the password π and the witness is π itself. The basic idea is that one party, A, generates an encryption of the password c and sends it to the other party, B. B generates a hashing key, hk, with a corresponding projected key, hp, and sends hp back to A. Both parties can compute the same value, $H(hk; c) = HP(hp, w; c)$, and B knows that A can only compute this value with possession of π. The process is then also run in the opposite direction.

Later, a generalised version of the Jiang–Gong protocol was developed by Groce and Katz [296]. Later still, Katz and Vaikuntanathan [417] optimised previous frameworks by using *non-adaptive* SPHFs, which allowed them to achieve one-round PAKE protocols that were secure in the standard model (without explicit authentication). The benefit of such generalisations is that new SPHF constructions can be used to realise new concrete PAKE protocols. For example, Katz and Vaikuntanathan [416] constructed a lattice-based SPHF and then applied the Gennaro and Lindell [296] framework to obtain a PAKE based on lattice problems.

Abdalla *et al.* [10] noticed that the CCA encryption used in the frameworks of Gennaro and Lindell [296], Groce and Katz [296] and Katz and Vaikuntanathan [417] can be replaced by a weaker form of encryption secure against only *plaintext-checkable attacks*. This allowed them to find more efficient concrete protocols using what they called Short Cramer–Shoup encryption. They proposed three abstract protocols with corresponding concrete protocols, which they called Secure Password-Only Key Exchange (SPOKE) protocols:

- a two-message, two-round protocol using the Gennaro and Lindell framework;
- a two-message, two-round protocol using the Groce and Katz framework with server-side explicit authentication;
- a two-message, one-round protocol using the Katz and Vaikuntanathan framework.

Protocol 8.12 shows the third of these concrete protocols, which Abdalla *et al.* [10] called KV-SPOKE.

The triple (h, c, d) constitutes a public key for the Short Cramer–Shoup encryption scheme, for which no decryption key is known. These form a common reference string for the protocol. As with other protocols in this section, the application of the generic framework implies a proof using the BPR00 model without any ideal cryptographic assumptions. For the instantiation of Protocol 8.12, the security proof relies on the difficulty of the DDH problem.

Despite the optimisation over several years in these generic protocols based on SPHFs, there is still a significant penalty in efficiency for protocols proven secure in the standard model, as compared with those assuming random oracles. Protocol 8.12 requires 14 regular exponentiations from each principal.

Shared information: Generator g of group $\mathbb{G}$ with prime order q. Hash function H. Password π.

Public information (common reference string): Public key h, c, d for Short Cramer–Shoup encryption scheme.

A		B
$r_A, r'_A, r''_A, s_A, \gamma_A \in_R \mathbb{Z}_q$		$r_B, r'_B, r''_B, s_B, \gamma_B \in_R \mathbb{Z}_q$
$t_A = g^{r_A} h^{s_A} c^{\gamma_A}$		$t_B = g^{r_B} h^{s_B} c^{\gamma_B}$
$t'_A = g^{r'_A} d^{\gamma_A}$		$t'_B = g^{r'_B} d^{\gamma_B}$
$u_A = g^{r''_A}$		$u_B = g^{r''_B}$
$v_A = h^{r''_A} g^{\pi}$		$v_B = h^{r''_B} g^{\pi}$
$\alpha = H(ID_A, ID_B, t_A, t'_A, u_A, v_A)$		$\beta = H(ID_B, ID_A, t_B, t'_B, u_B, v_B)$
$w_A = (cd^{\alpha})^{r''_A}$		$w_B = (cd^{\beta})^{r''_B}$
	$\xrightarrow{t_A, t'_A, u_A, v_A, w_A}$	
	$\xleftarrow{t_B, t'_B, u_B, v_B, w_B}$	
$\beta = H(ID_B, ID_A, t_B, t'_B, u_B, v_B)$		$\alpha = H(ID_A, ID_B, t_A, t'_A, u_A, v_A)$
$K_A = (t_B t'^{\alpha}_B)^{r''_A}$		$K_B = (t_A t'^{\beta}_A)^{r''_B}$
$K_B = u_B^{r_A + \beta r'_A} (v_B / g^{\pi})^{s_A} w_B^{\gamma_A}$		$K_A = u_A^{r_B + \alpha r'_B} (v_A / g^{\pi})^{s_B} w_A^{\gamma_B}$
	$\mathbf{K} = K_A K_B$	

Protocol 8.12: KV-SPOKE protocol of Abdalla, Benhamouda and Pointcheval

8.3.9 Protocols Using a Server Public Key

Some of the earliest proposed password-based protocols assume that users not only share passwords with a server but also are in possession of, or at least can obtain, an authentic server public key. This is a significant additional assumption, but it may be realistic in certain applications; a user may trust a workstation to obtain the server's public key or there may be a means to allow the user to check the public key. Since the user's password must be entrusted to some computing device, it does not seem unreasonable to give such a device some trust in obtaining the server public key. Halevi and Krawczyk [341, 342] have proposed a method whereby users can verify the hash of the server public key using sequences of (English) words.

In contrast to EKE and its variants, password-based protocols employing a server key have no need to use the password as a key for encryption, but, instead, the password can be encrypted by the user with the server public key. A critical issue with this approach is whether the adversary is able to gain any useful information from the ciphertext containing the password. One immediate objection may be that the adversary can make trial encryptions of candidate passwords and check if the same ciphertext is obtained. Such a possibility illustrates that it is usually necessary for the

encryption to be probabilistic so that a different ciphertext is obtained each time. We will make this requirement more precise later.

In this section, we include both two-party and three-party protocols. Public key computations on the client side are typically restricted to a single encryption in this class of protocols. The two-party protocols tend to be more efficient for the client than the two-party protocols using ephemeral public keys. However, in contrast to the Diffie–Hellman-based PAKE protocols examined earlier, forward secrecy is often sacrificed.

Kwon and Song [467, 468] proposed a set of password-based protocols. Protocol 8.13 is their basic protocol. The session key is generated by B and masked using nonces sent by A in the first message. The protocol is simple and the computation required for both parties is small. A drawback is that this protocol does not provide forward secrecy.

Shared information: Hash function H. Password π. Authentic public key of B.

A		B
Choose random r_A, r'_A		
	$\xrightarrow{\mathsf{Enc}_B(r_A, r'_A, \pi \oplus r'_A)}$	Decrypt and verify
		Choose $\mathbf{K}$
		$C = \mathbf{K} \oplus r_A \oplus r'_A$
Decrypt C and verify hash	$\xleftarrow{C, H(\pi \oplus r_A, \mathbf{K}, r'_A)}$	
	$\xrightarrow{H(\pi \oplus r'_A, \mathbf{K}, r_A)}$	Verify hash

Protocol 8.13: Kwon–Song basic protocol

The final message in Protocol 8.13 prevents a replay attack. An adversary who obtains an old $\mathbf{K}$ value can replay the first message and obtain the new session key, since B will reuse the same r_A and r'_A values. However, the adversary cannot complete the protocol, since r_A and r'_A are still unknown and so the correct final message cannot be formed.

Kwon and Song provided a GNY logic analysis of their basic protocol. However, there was no specification of the requirements for the public key encryption algorithm. There does not seem to be any specific requirement for non-malleability, since an adversary should not be able to obtain any of the encrypted values even if an old session key is obtained.

Kwon and Song [467, 468] also provided some variant protocols. Their 'Challenger's Public Key Protocol' requires the responder to know the authentic public key of the initiator but includes an unspecified symmetric-key encryption algorithm using the password. They also presented two key agreement protocols, the first of

which does not provide forward secrecy, while the second is essentially the same as Diffie–Hellman-based EKE.

Halevi and Krawczyk [341, 342] designed password-based protocols in a modular way and conducted a formal analysis. They provided a formal proof of security for the challenge–response protocol that forms the basis of their other protocols, but only an outline argument of how the proof can be extended to key establishment. Boyarsky [127] argued that the protocols in the original paper [341] cannot be secure without a stronger assumption on the asymmetric encryption algorithm used, unless only one user is involved. A stronger assumption was used in the later paper of Halevi and Krawczyk [342] but still not as strong as suggested by Boyarsky. However, all parties agree that a cryptosystem that provides non-malleability is sufficient for security.

In all their protocols Halevi and Krawczyk assume that the user A already has possession of a 'public password', which is a hashed version of the public key of the server B. This value can be used to verify the server's full public key, which is sent to the user as part of the protocol. We omit this in our descriptions. Protocols for entity authentication only were proposed, in addition to protocols for key exchange with and without forward secrecy.

The key establishment protocols use a similar design to the SKEME protocols defined in Protocols 5.30 and 5.42 in Chap. 5. We present the version without forward secrecy in Protocol 8.14. In order to preserve our usual convention that A is the client and B the server, the protocol is shown with the first message originating from B. In practice, A would often need to start the protocol with a request to B to connect, which would precede the protocol shown.

Shared information: Password π. Security parameter L.

A		B
		$r_B \in_R \{0,1\}^L$
Choose random K_0	$\xleftarrow{r_B}$	
	$\mathsf{Enc}_B(K_0, \pi, r_B, ID_A, ID_B) \xrightarrow{}$	Decrypt and verify
		$y = \mathsf{MAC}_{K_0}(r_B, ID_B, ID_A)$
Verify y	$\xleftarrow{y}$	$\mathbf{K} = \mathsf{MAC}_{K_0}(y)$
$\mathbf{K} = \mathsf{MAC}_{K_0}(y)$		

Protocol 8.14: Halevi–Krawczyk password-based protocol

Protocol 8.14 is a simple version of the protocol of Halevi and Krawczyk. More generally, they proposed that the encrypted message sent from A to B can take the more complex form $\mathsf{Enc}_B(K_0, f_\pi(r_B, K_0, ID_A, ID_B))$ for a function $f_\pi(X)$ used

to combine the password with the other protocol fields requiring verification. It is required that f is one-to-one (injective) when either π or X is fixed.

However, Kolesnikov and Rackoff [445] showed that an explicit attack is possible for certain valid choices of f. This attack does not apply when $f_\pi(X)$ is the concatenation of π and X as used in Protocol 8.14. Note that although K_0 is transported from A to B, the session key $\mathbf{K}$ is formed by key agreement since it includes inputs from both parties.

The properties and computational requirements are similar to those of Protocol 8.13 but a difference is that in Protocol 8.14 the server and client both contribute to the session key, whereas in Protocol 8.13 it is the server that generates the key. Like the SKEME variant Protocol 5.42, Protocol 8.14 does not provide forward secrecy. Another protocol with the same basic design as Protocol 8.14, but incorporating a Diffie–Hellman exchange, was proposed by Halevi and Krawczyk to provide this property. As usual, there is a computational cost in so doing, which amounts to two extra exponentiations per user.

8.3.10 Comparing Two-Party PAKE Protocols

Some of the features of the prominent two-party PAKE protocols examined in this section are summarised in Table 8.2. We compare only protocols which do not require a server public key, thus excluding the protocols of Kwon and Song (Protocol 8.13) and Halevi and Krawczyk (Protocol 8.14) from the table.

Table 8.2: Properties of two-party PAKE protocols

Properties → ↓ *Protocol*	No. of messages	Security proof	Mutual authentication	Client exponentiations
DH-EKE (8.1)	4	No	Yes	2
PAK (8.3)	3	ROM	Yes	2
PPK (8.4)	2	ROM	No	2
SPEKE (8.5)	3	ROM	No	2
Dragonfly (8.6)	4	ROM	Yes	3 (4)
SPAKE (8.7)	2	ROM	No	2
J-PAKE (8.8)	4	ROM	Yes	14 (10)
LST-J-PAKE (8.9)	4	ROM	Yes	10 (7)
KOY (8.10)	3	Std	No	15 (6)
Jiang–Gong (8.11)	3	Std	Yes	4
KV-SPOKE (8.12)	2	Std	No	14 (7)

Efficiency

The number of messages is one measure of the communications efficiency that is easy to compare for the different protocols. However, some of the protocols can run with fewer message flows, but that may result in an increased number of rounds. Usually the protocols with four message flows achieve explicit mutual authentication, but sometimes it is possible to run protocols, such as SPEKE, without authenticators to use only two messages like PPK.

The public key operations dominate the other computations in all protocols in this chapter, and so we count only these. Table 8.2 attempts to compare the comutational performance of the two-party PAKE protocols from the client side. The requirements for each protocol are estimated by counting the number of exponentiations required. It must be appreciated that this is only an indication of the relative computational effort, which depends on several detailed factors. Furthermore, when security is based on different computational problems the relationship between security and parameter size can be quite different too.

In Table 8.2, we are normally counting exponentiations in the group $\mathbb{G}$. For protocols set in $\mathbb{Z}_p^*$, the group $\mathbb{G}$ is often a much smaller subgroup. For PAK and PPK we assume that small exponents are used as proposed by MacKenzie and Patel [511], although, strictly speaking, those authors suggested exponents slightly bigger than the size of $\mathbb{G}$. The cost of exponentiations can be reduced through the use of multi-exponentation algorithms, and in Table 8.2 the counts in brackets are the numbers of multi-exponentiations required. For the J-PAKE protocol, we assume that the knowledge proof is a Schnorr signature and costs one exponentiation to compute and two to verify (or one multi-exponentation). For the Jiang–Gong protocol, we assume that encryption costs one exponentiation.

For Diffie–Hellman-based protocols, the basic requirements for each principal are two exponentiations: one to calculate t_A and the other to calculate $\mathbf{Z}$. Although this minimum is apparently achieved in the original Diffie–Hellman EKE, its vulnerabilities make it a dubious choice today. However, we can still achieve this minimum for protocols such as PAK, SPEKE and SPAKE. It is perhaps remarkable that today it seems that the best password-based protocols possess most of the good properties that we expect of protocols using long secrets.

Security

When selecting a protocol for use, undoubtedly the most important factor is security. Following the trend in cryptography generally, key exchange protocols, and PAKE protocols in particular, are today usually expected to come with a security proof. Since there are a number of protocols with proven security, and especially bearing in mind the many subtle attacks found on earlier protocols, it seems prudent to use one of these. Examination of Table 8.2 reveals that we can select a two-party PAKE protocol with a formal proof of security with little sacrifice in performance.

Although today we have many protocols with a security proof, many of them require some *idealised* model for the cryptographic primitives they make use of. This is

usually the ideal cipher model or the random oracle model. Table 8.2 denotes proofs which use random oracles as ROM. Since 2001 there has been a line of research dedicated to avoiding such idealised models; protocols with proofs that avoid those models are typically said to be secure in the *standard model*.

For the protocols examined in other chapters, it has been common to assume that all parties have a public key known to all other entities. In practice, this requires a pubic key infrastructure and trust in one or more certification authorities. For PAKE protocols it is not obvious that any kind of trusted entities are required, but many protocols assume the existence of a *common reference string*, generated once independently of other protocol parameters. This is a less strong assumption than a full public key infrastructure but avoiding even a common reference string is preferable, at least theoretically. Goldreich and Lindell [310] proved the existience of secure PAKE protocols based only on very general assumptions regarding the existence of one-way functions and trapdoor one-way permutations.

8.4 Two-Party Augmented PAKE Protocols

Just as the original EKE protocol of Bellovin and Merritt has a corresponding augmented protocol, examined in Sect. 8.2.2, many two-party PAKE protocols have an augmented version in which the server stores only an image of the password, not the password itself. We examine a number of such protocols in this section.

Most of the augmented PAKE protocols do not specify how to set up the password image on the server side. Of course, the password itself could simply by transported to the server using some secure channel, and the server itself could then compute the image. However, a better method may be to allow the server to obtain the image without ever knowing the password itself, thus reducing trust in the server and minimising exposure of the password. Kiefer and Manulis [426] introduced the notion of *blind password registration*, which allows a user to transfer an image of a password to the server while at the same time allowing the server to ensure that a chosen policy on password selection has been satisfied. Their protocol does not work for all types of password image, but applies at least to a set known as verifier-based PAKE, described by Benhamouda and Pointcheval [87].

Gentry *et al.* [299] proposed a generic method to convert any secure two-party PAKE protocol into a secure augmented two-party PAKE protocol. The general idea is to first run the original PAKE protocol with the image of π, $f(\pi)$, in place of π, and then for the client to prove knowledge of π using a signature whose signing key can only be obtained with knowledge of π. Generally, this process may add a round of communication, as well as adding the cost of computing and verifying the signature. For specific protocols, the extra messages could be piggybacked onto existing protocols to reduce any increase in communication rounds, such as is done in the PAK-Z+ protocol examined in Sect. 8.4.1. Gentry *et al.* [299] proved security of their construction in the universal composability framework. Strictly speaking, this means that their security theorem is only relevant when the conversion process

is applied to a PAKE protocol with a universally composable proof already, which is not the case for most PAKE protocols.

8.4.1 PAK-X, PAK-Y and PAK-Z

Several augmented variants of PAK have been designed. The first was called PAK-X, designed by Boyko *et al.* [145]. PAK-X is very similar to Protocol 8.3 but incorporates an additional verifier. MacKenzie [507] then devised a conceptually simpler version called PAK-Y, where the verifier takes the form of a Schnorr signature. Later, MacKenzie [509] proposed a generalisation of PAK-Y, called PAK-Z, which allows any digital signature to take the place of the Schnorr signature used in PAK-Y. Unfortunately, PAK-Z turned out to be insecure with certain choices of signature [299] and had to be improved again, leading to Protocol 8.15, known as PAK-Z+, proposed by Gentry *et al.* [298].

The structure of Protocol 8.15 essentially follows the generic conversion method of Gentry *et al.* [299] applied to the PAK protocol (Protocol 8.3).The first two messages in Protocol 8.15 are similar to those of Protocol 8.3, but in the last message A sends a signature of the fields $ID_A, ID_B, m, t_B, \mathbf{Z}, P$ using the predefined signing key V. The signature algorithm is not specified except that it must be existentially unforgeable under a chosen message attack. The signature allows B to verify that A possesses V by using the corresponding signature verification key W. Since knowlege of π is necessary to obtain V, this also shows that A possesses π. (Note that B neither possesses V nor is able to compute it.)

As in the basic PAK protocol (Protocol 8.3), the hash function H_1 must take values uniformly in $\mathbb{G}$. This may require an expensive hash function or, otherwise, short exponents can be used if $\mathbb{G}$ is chosen to be $\mathbb{Z}_p^*$ [511]. The other hash functions, H_i for $2 \leq i \leq 6$, map to bit strings of appropriate size. As well as allowing any strongly secure signature to be used, PAK-Z+ uses the same verifier k as used in Protocol 8.3 so that the result is a modular approach to the set of PAK protocols.

The PAK-Z+ protocol was proven secure by Gentry *et al.* [298] in the BPR00 model assuming the difficulty of the computational Diffie–Hellman problem. The proof models the hash functions as random oracles. PAK-Z+ is standardised in the IEEE P1363-2 standard [373], where it is known simply as PAKZ. The standard allows the protocol to be run in elliptic curve groups, as well as in $\mathbb{Z}_p^*$ as shown in Protocol 8.15.

8.4.2 B-SPEKE

Jablon [389] discussed the method used by Bellovin and Merritt to augment EKE and proposed an alternative. He pointed out that Bellovin and Merritt's technique could be applied generally and, in particular, to his own SPEKE protocol, leading to a variant he called A-SPEKE. His own augmentation technique he called the 'B' extension.

Information held by A: Password π.
Information held by B: $P = H_1(ID_A, ID_B, \pi)$, $V' = H_2(ID_A, ID_B, \pi) \oplus V$, $V'' = H_3(V)$, W. Here W is the verification key for a signature scheme with corresponding signing key V.
Shared information: Group $\mathbb{G}$ of prime order q and generator g of $\mathbb{G}$.

A		B
$P = H_1(ID_A, ID_B, \pi)$		
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
$m = t_A \cdot P$	$\xrightarrow{m}$	Check $m \neq 0 \in \mathbb{G}$
		$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
		$\mathbf{Z} = (m/P)^{r_B}$
		$sid = (ID_A, ID_B, m, t_B)$
		$a = H_6(sid, \mathbf{Z}, P) \oplus V'$
		$k = H_4(sid, \mathbf{Z}, P)$
	$\xleftarrow{t_B, k, a, V''}$	
$\mathbf{Z} = t_B^{r_A}$		
$sid = (ID_A, ID_B, m, t_B)$		
$k \stackrel{?}{=} H_4(sid, \mathbf{Z}, P)$		
$V = H_2(ID_A, ID_B, \pi) \oplus H_6(sid, \mathbf{Z}, P) \oplus a$		
$H_3(V) \stackrel{?}{=} V''$		
$s = \mathsf{Sig}_V(sid)$	$\xrightarrow{s}$	Verify s using W
$\mathbf{K} = H_5(ID_A, ID_B, m, t_B, \mathbf{Z}, P)$		

Protocol 8.15: PAK-Z+ protocol

Instead of employing a signature to prove knowledge of the password, as used in Bellovin and Merritt's 'A' extension, the 'B' variant uses an additional Diffie–Hellman exchange. The server B stores two password images as before: one is the square of the hash, $P = H(\pi)^2$, and the other is $V = g^{\pi}$. During the protocol, B sends an additional challenge $g^{\tilde{r}_B}$ to A, and A must respond by showing knowledge of the Diffie–Hellman key $\tilde{Z}_{AB} = g^{\pi \tilde{r}_B}$. A cannot simply send $\tilde{Z}_{AB}$ to B, since this value could be used together with the challenge to form a dictionary attack on π. However, if A returns a hash containing both $\mathbf{Z}$ and $\tilde{Z}_{AB}$, the hash can be used both to check knowledge of π and for confirmation of $\mathbf{Z}$. Employing this technique with the SPEKE protocol leads to B-SPEKE, Protocol 8.16.

Jablon was not explicit about the construction of the verifiers V_A and V_B in Protocol 8.16. We have adapted the verifier format used in Protocol 8.5. In any case, the proof of MacKenzie for Protocol 8.5 does not cover Protocol 8.16. Jablon [389]

Information held by A: Password π, interpreted as an element of $\mathbb{Z}_q$.
Information held by B: Derived password image $V = g^{\pi}$. Derived password image $P = H(\pi)^2$ where $H(\pi)$ is interpreted as an element of $\mathbb{Z}_p^*$.
Shared information: Subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Hash functions H, H_0, H_1, H_2. Security parameter L.

A		B
$P = H(\pi)^2$		
$r_A \in_R \{1,\ldots,2^L\}$		
$t_A = P^{r_A}$	$\xrightarrow{t_A}$	$r_B, \tilde{r}_B \in_R \{1,\ldots,2^L\}$
		$\mathbf{Z} = t_A^{2r_B}$
		Check $\mathbf{Z} \notin \{-1,0,1\}$
		$t_B = P^{r_B}, \tilde{t}_B = g^{\tilde{r}_B}$
$\mathbf{Z} = t_B^{2r_A}$	$\xleftarrow{t_B, V_B, \tilde{t}_B}$	$V_B = H_1(t_A, t_B, \mathbf{Z}, P)$
Check $\mathbf{Z} \notin \{-1,0,1\}$		$\tilde{Z}_{AB} = V^{\tilde{r}_B}$
Verify V_B		
$\tilde{Z}_{AB} = (\tilde{t}_B)^{\pi}$		
$V_A = H_2(t_A, t_B, \mathbf{Z}, \tilde{Z}_{AB}, P)$	$\xrightarrow{V_A}$	Verify V_A
	$\mathbf{K} = H_0(\mathbf{Z})$	

Protocol 8.16: B-SPEKE protocol

provided a number of variants of Protocol 8.16 to optimise various features; for example, he presented a version designed to minimise the total time by allowing various calculations to take place in parallel.

The security proof of SPEKE due to MacKenzie [508] does not extend to B-SPEKE and there is no other known proof. However, Pointcheval and Wang [617] designed a protocol conceptually similar to B-SPEKE, called Verifier-based Two-Basis Password Exponential Key Exchange, or VTBPEKE. Although their protocol is less efficient than B-SPEKE they did provide a security proof, which even includes a proof of forward secrecy.

8.4.3 SRP

The SRP (Secure Remote Password) protocol was designed by Wu [740] as a more efficient alternative to the original augmented EKE. The server B holds a salt value s and the password image $V = g^{H(s,\pi)}$. Protocol 8.17 shows an optimised version of SRP, minimising the number of messages.

The shared secret is $\mathbf{Z} = g^{r_A r_B} \cdot V^{u r_B}$, which is unusual in that it is asymmetric with regard to the inputs of A and B. Another unusual feature is the public random value u chosen by B and included in the derivation of $\mathbf{Z}$. The purpose of u is to ensure that

Shared information: Subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Hash function H.
Information held by A: Password π.
Information held by B: Password image $V = g^{H(s,\pi)}$. Salt value s.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{ID_A, t_A}$	$u, r_B \in_R \mathbb{Z}_q$
		$\mathbf{Z} = (t_A V^u)^{r_B}$
$V = g^{H(s,\pi)}$	$\xleftarrow{S, s, u}$	$S = V + g^{r_B}$
$\mathbf{Z} = (S-V)^{r_A + uH(s,\pi)}$		
$M_1 = H(t_A, S, \mathbf{Z})$	$\xrightarrow{M_1}$	Verify M_1
		$M_2 = H(t_A, M_1, \mathbf{Z})$
		$\mathbf{K} = H(\mathbf{Z})$
Verify M_2	$\xleftarrow{M_2}$	
$\mathbf{K} = H(\mathbf{Z})$		

Protocol 8.17: SRP protocol

A knows the value of π and not just its image V. If A could know u in advance then she could send $g^{r_A} V^{-u}$ in the first message instead of t_A and then B would calculate $\mathbf{Z} = g^{r_A r_B}$. This means that A could complete the protocol with knowledge only of V. In order to avoid this possibility, it is evident that u cannot be a fixed value. However, for efficiency reasons, Wu suggested that u may be defined as a public function of S, which would allow it to be omitted from the exchanged messages.

The symmetric encryption algorithm used with the password is addition modulo p. Forward secrecy is provided since the ephemeral Diffie–Hellman key, $g^{r_A r_B}$, is needed in order to find $\mathbf{Z}$. The main drawback of SRP is the lack of a formal security proof. However, Wu did point out that security against passive eavesdroppers is provided in the sense that an oracle that can find the session key from the exchanged messages t_A, S, and u is able to solve the Diffie–Hellman problem.

A minor weakness in Protocol 8.17, whose discovery was attributed by MacKenzie [508] to Bleichenbacher, allows the adversary to eliminate two passwords, π_1 and π_2, with each protocol run. An adversary masquerading as B can exploit the symmetry of the value S by choosing $V_1 = g^{H(s,\pi_1)}$ and $V_2 = g^{H(s,\pi_2)}$ and sending $S = V_1 + V_2$ instead of a correctly formed S in the second message. Then if π_1 is the correct password, the adversary can check that $\mathbf{Z} = V_2^{r_A + uH(s,\pi_1)}$, while a symmetric test can be used for π_2.

The calculation of V must be done explicitly by A during the protocol, but Wu implied that $H(s,\pi)$ can be a 'tiny' exponent since the entropy of π is small. Wu regarded it as optional whether the values p and g were fixed or were sent as part of

the protocol itself (the former situation is intended in Protocol 8.17). If p and g are sent as parameters by B at the start of the protocol, it is essential that A checks the primality of $q = (p-1)/2$ and that g has order q.

The version of SRP shown in Protocol 8.17 became known as SRP-3 and was standardised in the IEEE P1363 standard [373]. A later version, known as SRP-6, was proposed by Wu [742] and was also standardised in the same IEEE P1363 standard [373], as well as in an ISO standard [385]. SRP-6 is shown as Protocol 8.18.

Shared information: Subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2$. Hash function H. Constant c which is a function of q and g.
Information held by A: Password π.
Information held by B: Password image $V = g^{H(s,ID_A,\pi)}$. Salt value s.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{ID_A, t_A}$	$r_B \in_R \mathbb{Z}_q$ $\mathbf{Z} = (t_A V^u)^{r_B}$ $u = H(t_A, S)$ $S = c \cdot V + g^{r_B}$
$V = g^{H(s,ID_A,\pi)}$ $u = H(t_A, S)$ $\mathbf{Z} = (S - c \cdot V)^{r_A + uH(s,ID_A,\pi)}$ $M_1 = H(t_A, S, \mathbf{Z})$	$\xleftarrow{S, s}$ $\xrightarrow{M_1}$	Verify M_1 $M_2 = H(t_A, M_1, \mathbf{Z})$ $\mathbf{K} = H(\mathbf{Z})$
Verify M_2 $\mathbf{K} = H(\mathbf{Z})$	$\xleftarrow{M_2}$	

Protocol 8.18: SRP-6 protocol

SRP-6 has two main changes from the SRP-3 protocol shown in Protocol 8.17.

- The value u is no longer chosen randomly by B but instead is computed as a hash of the two exchanged messages. The purpose of this change is to counter attacks that could occur if the message ordering is changed. Specifically, in the original SRP protocol it is insecure to allow A to choose t_A as a function of u (specifically, to choose $t_A = V^{-u}$). The change prevents this attack even if messages are reordered.
- The computation of S is changed to include the constant multiplier c for V. Wu [742] originally suggested that $c = 3$ could always be used, but the standards

instead say that c should be chosen for each set of parameters to be a function of q and g. The purpose of this change is to defend against the attack mentioned above, where one trial could elminate two passwords, by destroying the potential symmetry in the S value.

8.4.4 AMP

The AMP (Authentication via Memorable Password) protocol of Kwon [465, 466] is a variant of Diffie–Hellman-based EKE that employs a number of different features found in other protocols. Like the PAK family, Protocol 8.19 masks one of the Diffie–Hellman exchanges by multiplying it by a derived version of the password. However, there is a different method to calculate the shared secret, which depends in an unusual way on the exchanged values.

Shared information: Subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of prime order $q = (p-1)/2r$. Hash functions H_i, $1 \le i \le 5$.
Information held by A: Password π.
Information held by B: Password image $V = g^{H_1(ID_A,\pi)}$.

A		B
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		
	$\xrightarrow{ID_A, t_A}$	$r_B \in_R \mathbb{Z}_q$
$e = H_2(t_A, ID_A, ID_B)$		$e = H_2(t_A, ID_A, ID_B)$ $T = (t_A^e V)^{r_B}$
	$\xleftarrow{T}$	
$P = H_1(ID_A, \pi)$ $\mathbf{Z} = T^{(r_A+1)/(r_A e+P)}$		$\mathbf{Z} = (t_A g)^{r_B}$
	$\xrightarrow{H_3(ID_A, ID_B, t_A, T, Z)}$	Verify hash
Verify hash	$\xleftarrow{H_4(ID_A, ID_B, t_A, T, Z)}$	
	$\mathbf{K} = H_5(ID_A, ID_B, t_A, T, Z)$	

Protocol 8.19: AMP protocol

The shared secret is $\mathbf{Z} = g^{r_B(r_A+1)}$, which, interestingly, is asymmetric like the shared secret of SRP. The AMP protocol requires only two exponentiations for both client and server. Both B-SPEKE and SRP also use two main exponentiations but have additional exponentiations with 'small' exponents. In contrast, AMP requires an inversion modulo q. Although the AMP specification allows a group $\mathbb{G}$ which is a relatively small subgroup of $\mathbb{Z}_{p^*}$, Kwon [466] recommended that p should be a

strong prime so that $q = (p-1)/2$. Informally, we can see that forward secrecy is provided, since the password does not influence the value of the session key.

There are many versions of AMP [465]. Protocol 8.19 is a version known as AMP+ and was standardised in the IEEE P1363 standard [373], as well as in an ISO standard [385]. Both standards include elliptic curve versions, while there is also a three-pass version available.

The AMP protocols lack a formal proof of security. One version, like Protocol 8.17, allowed an adversary to eliminate two passwords in one protocol run.[2] However, this attack is prevented in Protocol 8.19.

8.4.5 AugPAKE Protocol

The AugPAKE protocol [672, 673] was proposed by Shin and Kobara aimed at the IETF standards track, originally in 2010. It is standardised in the ISO 11770-4 standard [385]. AugPAKE is a four-round augmented EKE protocol with many similarities to Protocols 8.17 and 8.19 but, unlike the latter two, has a proof of security in the BPR00 model, where the hash functions H_1, H_2, H_3 are modelled as random oracles. Protocol 8.20 shows the message flows.

Shared information: Group $\mathbb{G}$ of prime order q. Hash functions H (mapping to $\mathbb{Z}_q$), and H_1, H_2, H_3 (mapping to $\{0,1\}^*$).
Information held by A: Password π.
Information held by B: Password image $V = g^{\pi}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$	$\xrightarrow{ID_A, t_A}$	$r_B \in_R \mathbb{Z}_q$
		$u = H(ID_A, ID_B, t_A)$
		$\mathbf{Z} = g^{r_B}$
$u = H(ID_A, ID_B, t_A)$	$\xleftarrow{ID_B, Y}$	$Y = (t_A V^u)^{r_B}$
$\mathbf{Z} = Y^{(r_A + \pi u)^{-1}}$		
$M_1 = H_1(ID_A, ID_B, t_A, Y, \mathbf{Z})$	$\xrightarrow{M_1}$	Verify M_1
		$M_2 = H_2(ID_A, ID_B, t_A, Y, \mathbf{Z})$
Verify M_2	$\xleftarrow{M_2}$	
	$\mathbf{K} = H_3(ID_A, ID_B, t_A, Y, Z)$	

Protocol 8.20: AugPAKE

[2] This attack was described by Michael Scott in a message on the IEEE P1363 mailing list in July 2001.

The computational requirements for Protocol 8.20 are very similar to those for SRP (Protocol 8.17). The designers pointed out that pre-computation of t_A by A and of $\mathbf{Z}$ by B can aid efficiency. They proposed to implement Protocol 8.20 in a subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ where $(p-1)/2$ has no small divisors, a so-called *secure prime*. When a secure prime is used, the receiving parties must check that the values t_A and Y do not have trivial values 0, 1 or -1, but group membership tests are not required. The security proof uses the strong Diffie–Hellman assumption. Although the protocol is the subject of a patent application, the holder has declared willingness to 'grant a non-exclusive royalty-free license' to implementations conforming to the standard.[3]

8.4.6 Using Multiple Servers

The motivation for augmented PAKE is that if the server is compromised then only a hashed image of passwords is revealed. However, these passwords can be expected to have low entropy, and such a breach will then allow an offline guessing attack. In order to protect against such attacks, an alternative to storing the hash of a password is to split the password across multiple servers. Several protocols have been designed for this scenario. Secret sharing is part of the solution, but in addition it is necessary to find ways to establish a shared session key without reconstructing the password. The cost to be paid for such security enhancements is that two or more independent servers must be active in each user session.

The first multiple-server PAKE protocols were due to Ford and Kaliski [281] and to Jablon [390], but these lacked security proofs. The first protocol with a proof appears to be due to MacKenzie *et al.* [512], but it required users to have authentic public keys for each of the servers rather than requiring only a password. The same can be said of a protocol of Brainard *et al.* [146, 706], which is dedicated to a dynamic way of sharing passwords between two servers without specifying a specific key exchange protocol.

Multi-server password-only PAKE protocols have also been proposed. Di Raimondo and Gennaro [247] designed such a protocol based on Protocol 8.10. Security is guaranteed as long as less than one third (these authors claim that this can be extended to one half) of the servers are corrupted. The efficiency of this secure version requires the client to run n copies of the KOY protocol. Although Di Raimondo and Gennaro also designed a *transparent* protocol, in which the user's view is the same as in the single-server case, that version is secure only against leaking password information and not against leakage of the session key.

Finally, there have been protocols designed specifically for the two-server case. Note that the Di Raimondo and Gennaro protocol [247] does not handle this case. Katz *et al.* [413] designed such a protocol, again based on Protocol 8.10. Clients need to do twice as much work as in Protocol 8.10, while servers do around four times as much. Kiefer and Manulis [427] improved the situation by designing protocols that are secure in the universal composability model. Yi *et al.* [759, 760] designed compilers, based on ID-based cryptography, which can transform any two-party PAKE protocol into a two-server version.

[3] https://datatracker.ietf.org/ipr/2411/

8.4.7 Comparing Two-Party Augmented PAKE Protocols

The properties of the protocols examined in this section are summarised in Table 8.3. As in Table 8.2, we have attempted to capture the main features regarding efficiency and security.

Table 8.3: Properties of augmented password-based protocols

Properties → ↓ *Protocol*	No. of messages	Security proof	Mutual authentication	Client exponentiations
Augmented EKE (8.2)	5	Broken	Yes	2
PAK-Z+ (8.15)	3	ROM	Yes	2
B-SPEKE (8.16)	3	No	Yes	2
SRP (8.17, 8.18)	4	No	Yes	2
AMP (8.19)	4	No	Yes	2
AugPAKE (8.20)	4	ROM	Yes	2

All of the protocols in Table 8.3 have at least three message flows and are all designed to achieve mutual authentication as well as key agreement. The computational requirements for each protocol are estimated as the exponentiations required for each side. B-SPEKE and SRP require additional exponentiations with very small exponents, which can be similar to the entropy of the password, perhaps 32 bits. We assume that PAK-Z+ uses small exponents even if the group is $\mathbb{Z}_p^*$ [511]. PAK-Z+ also requires signature generation by the client (and verification by the server) for some generic secure signature scheme.

The range of augmented protocols is smaller than those without the augmented property (compare Table 8.2). Furthermore, we have listed only two examples, PAK-Z+ and AugPAKE, with security proofs, and these are both in the random oracle model. (The VTBPEKE protocol referred to in Sect. 8.4.2 should also be mentioned; its proof is also in the random oracle model.)

The efficiency properties of augmented protocols typically match those of protocols without the augmented property, except for the number of messages. Note that four-message protocols, such as AugPAKE, which run in two rounds can typically be converted to three-message protocols which can be run in three rounds by combining the fourth message with the second message flow.

8.5 RSA-Based Protocols

Although most password-based protocols rely on Diffie–Hellman key exchange, it is not surprising that the widespread popularity of the RSA algorithm [630] has been the basis of some alternatives. We examine such protocols in this section.

8.5.1 RSA-Based EKE

Bellovin and Merritt's original EKE protocol is applicable to a number of public key algorithms, and they included a version for RSA in their paper [84]. The basic model for EKE dictates that A should choose an ephemeral RSA public key and send it to B encrypted with π. B will then choose the session key and return it encrypted with the RSA key and, optionally, with π.

Bellovin and Merritt recognised a number of potential problems with this approach and discussed some possible remedies. The main problem is how to make an RSA key appear 'random' in order to avoid a partition attack, which might eliminate candidate passwords that do not provide a valid RSA key pair when applied to decrypt the first message. They pointed out that sending the RSA key pair encrypted with the password is dangerous, since most integer values do not make a good RSA modulus. Therefore a partition attack is possible, whereby the adversary decrypts the ciphertext with a candidate password and can eliminate that candidate if the plaintext modulus has small factors. This would allow an adversary to eliminate a large proportion of passwords.

In view of the above problem Bellovin and Merritt proposed that only the RSA exponent e should be encrypted with π, while the modulus n would be sent as cleartext. They further suggested that, since e must always be odd in the RSA algorithm, either e or $e+1$ should be sent randomly to ensure that almost all integers less than n could occur as the encrypted value (they suggested making n the product of two safe primes[4] to enhance this property). The first two messages in the protocol are then as follows.

1. $A \rightarrow B : ID_A, n, \{e\}_\pi$
2. $B \rightarrow A : \{\mathbf{K}^e \bmod n\}_\pi$

....

Notice that an eavesdropper who obtains an old session key K'_{AB} can decrypt the first message with a candidate $\tilde{\pi}$ to obtain a candidate $\tilde{e}$ and then calculate $\{(K'_{AB})^{\tilde{e}} \bmod n\}_{\tilde{\pi}}$. If the RSA encryption is deterministic, this can be used to verify whether $\tilde{\pi} = \pi$ by comparing the result with the second message in the old protocol run which used K'_{AB}. Therefore it is important that randomness is included with the session key as part of the RSA encryption.

Another necessary measure to prevent partition attacks is to ensure that candidate decrypted values in the second message are smaller than n; as with Diffie–Hellman-based EKE, this may be addressed by ensuring that the modulus is slightly less than a power of 2. However, Patel [603] showed that these precautions are still not sufficient to protect the protocol. The attack of Patel requires the adversary to replace the first message with A, n', X, where X is random and n' is carefully chosen so that $n' = pq$ with $3|(p-1)$ and $3|(q-1)$. This choice of n' ensures that the mapping $x \mapsto x^3 \bmod n$ is a 9-to-1 mapping in $\mathbb{Z}^*_{n'}$. On receipt of this message, B will 'decrypt' X with π and use the resulting value e' to encrypt the random value $\mathbf{K}$. There is a probability of

[4] A prime p is a *safe prime* if $(p-1)/2$ is also prime.

1/3 that e' is divisible by 3. Although the adversary cannot know if this is the case, the attack can be repeated if it fails initially. Assuming it is the case, the adversary can partition the candidate passwords into those that decrypt the second message into a cubic residue and those that do not. The latter case occurs with probability about 8/9 and these candidates can be discarded, since the value $\mathbf{K}^e$ is a cubic residue. Cubic residues can easily be recognised by the adversary by raising to the power $(p-1)/3 \bmod p$ and also raising to the power $(q-1)/3 \bmod q$, and then checking that both results equal 1. By repeating the attack with the same first message, the adversary can obtain a new reply and use it in the same way to discard a proportion 8/9 of the remaining candidate passwords. The attack can easily be generalised to use any small value in place of 3 as the divisor. An attempt to avoid the attack by having the receiver check that the decrypted e has no small factors fails because the adversary can now discard passwords that result in such e values.

8.5.2 OKE and SNAPI

Although the attack of Patel seems fatal for RSA-based EKE, a new approach was taken by Lucks [506] to revive the protocol. The protocol of Lucks was called OKE (Open Key Exchange), which emphasises that the public ephemeral RSA key is sent 'in the open' (as plaintext). Lucks provided a proof in a Bellare–Rogaway-style model that OKE is secure in a general setting. However, there are problems in satisfying the assumptions of the proof in the RSA setting; the difficulties revolve around the ability of the adversary to select the public RSA key in such a way as to obtain information about the password, in much the same way as in Patel's attack outlined above. Indeed, MacKenzie *et al.* [510] subsequently did find an attack on the RSA version of OKE along these lines. They also proposed a variant of RSA-based OKE called SNAPI (Secure Network Authentication with Password Information), which avoids the attack and is described in Protocol 8.21.

A critical factor in avoiding the attack on the OKE protocol is the choice of RSA public key. The RSA modulus N is chosen using a security parameter L_1 so that $2^{L_1-2} \leq N \leq 2^{L_1}$. The exponent e must be in the range $2^{L_1} \leq e \leq 2^{L_1+1}$. A consequence is that exponentiation with e is more than a full-length exponentiation; with typical values, e will be 1025 bits long.

An important check is that the parameter p must lie in the set $S_N = \{x : x \leq 2^\eta - (2^\eta \bmod N) \wedge (x,N) = 1\}$. Since p is an output of H which has length at least $L_1 + L_2$ and N has length not more than L_1, this will fail with a tiny probability if N is chosen honestly, but an adversary could choose N with small factors to make the probability high that $(x,N) > 1$. If the condition is not satisfied by the value of p generated then q is set to the random value a, so that in any event q appears to A to be random.

MacKenzie *et al.* [510] provided a proof of security of SNAPI in Shoup's simulation model. As well as possessing a security proof, a potential benefit of SNAPI is that the ephemeral RSA key can be reused in several protocol runs. Since RSA key generation is a computationally costly process, this is a useful advantage and means

Shared information: Password π. Security parameters L_1 and L_2. A 'long' hash function H with output of length η with $\eta \geq L_1 + L_2$, and three 'short' hash functions H_1, H_2 and H_3 with outputs of length L_2. A set $S_N = \{x : x \leq 2^\eta - (2^\eta \bmod N) \wedge (x,N) = 1\}$.

A		B
$n_A \in_R \{0,1\}^{L_2}$		
Choose RSA public key N,e		
	$\xrightarrow{A,n_A,N,e}$	
		Check range of N,e
		Check e is prime
		Check length of n_A
		$n_B \in_R \{0,1\}^{L_2}$
		$p = H(N,e,n_A,n_B,ID_A,ID_B,\pi)$
		$a \in_R \mathbb{Z}_N^*$
		If $p \notin S_N$ then $q = a$
		else $q = pa^e \bmod N$
	$\xleftarrow{n_B,q}$	
Check length of n_B		
Check $(q,N) = 1$		
$p = H(N,e,n_A,n_B,ID_A,ID_B,\pi)$		
Check $p \in S_N$		
$a = (q/p)^d \bmod N$		
$r = H_1(N,e,n_A,n_B,ID_A,ID_B,q,a)$	$\xrightarrow{r}$	If $p \notin S_N$ then halt
		Verify r
		$t = H_2(N,e,n_A,n_B,ID_A,ID_B,q,a)$
		$\mathbf{K} = H_3(N,e,n_A,n_B,ID_A,ID_B,q,a)$
	$\xleftarrow{t}$	
Verify t		
$\mathbf{K} = H_3(N,e,n_A,n_B,ID_A,ID_B,q,a)$		

Protocol 8.21: SNAPI protocol

that SNAPI can be more efficient than PAK. A drawback of this approach is that storing the ephemeral key makes it part of the long-term key for A, and then compromise of the private key means that old session keys can be recovered for all sessions in which that RSA key was used: in other words, this computational advantage comes at the cost of a loss in forward secrecy. MacKenzie *et al.* [510] also proposed an extension to SNAPI, called SNAPI-X, which allows the server side to store only an image of π under a hash function, similarly to the protocols in Sect. 8.2.2. Park *et al.* [601] proposed a variant of SNAPI in which a shorter value of e is used (they suggested 96 bits), with the aim of improving the efficiency of the protocol. Unfortunately, this change leads to weaknesses, as pointed out by Youn *et al.* [768].

Another variant of OKE was proposed by Roe *et al.* [214, 633]. A novel feature of their protocol is that A sends only the RSA modulus N, while the exponent e is defined as a deterministic function of π; this makes the protocol potentially more efficient than SNAPI (or the original OKE). In the original version of the protocol [633] B returns $z^{e(\pi)} \bmod N$, where z is a random number of appropriate length used to define various values, including the session key and a nonce n_A. On receipt of this value, A calculates the decryption key $e(\pi)^{-1} \bmod \phi(N)$ and decrypts z as in normal RSA. Then A returns the value n_A to B. Bleichenbacher [119] found that multiple passwords could be checked by a malicious B for each protocol run, by sending $z^{e(\pi_1)e(\pi_2)\ldots e(\pi_n)}$ and using the returned n_A value to check which, if any, of the $e(\pi_i)$ values were removed by A. A later version of the protocol [214] avoids this attack by adding $2^{e(\pi)} \bmod N$ to the value $z^{e(\pi)} \bmod N$ sent by B.

8.6 Three-Party PAKE Protocols

The protocols we have examined so far in this chapter are appropriate for the situation when a client wishes to connect securely to a server; this is the case of two-party PAKE. We now consider three-party PAKE protocols, designed to allow two users to establish a new key through the cooperation of a mutually trusted server. Although the server shares a different password (or an image of the password) with each user in both cases, the goals are different.

The proposed three-party PAKE solutions include both special-purpose protocols and generic constructions. The idea of the latter is to run two two-party protocols, each running between the server and one of the two principals; the two secure channels established can be then used to distribute a session key from the server. Some of the three-party protocols in this section require that users have knowledge of a server public key, while others are password-only protocols.

8.6.1 GLNS Secret Public Key Protocols

Gong, Lomas, Needham and Saltzer (GLNS) [319] (and the same authors earlier with names permuted [500]) published the first password-based protocols for a variety of authentication and key establishment scenarios. We call their set of proposals the GLNS protocols. The majority of their protocols assume that users have knowledge of the server public key – these protocols are examined in Sect. 8.6.3 below. Here we examine their protocols which use 'secret public keys'; these are ephemeral asymmetric keys which might normally be made public but, in order to prevent guessing attacks, are kept secret from eavesdroppers on the protocol. Gong *et al.* [319] provided variants of their protocols for both the three-party and the two-party situations.

The GLNS three-party secret public key protocol is shown in Protocol 8.22. Their two-party 'Direct Authentication Protocol' uses similar ideas by essentially combining the roles of A and S. In the initial two messages, the server S sends encrypted ephemeral public keys K_S for A and K'_S for B. Subsequently, these are used by A and

S has passwords π_A and π_B. S chooses random value n_S and ephemeral encryption keys K_S and K'_S.
A has password π_A. A chooses random values n_A, n'_A, c_A, r_A.
B has password π_B. B chooses random values n_B, n'_B, c_B, r_B.

1. $A \rightarrow S: \ ID_A, ID_B$
2. $S \rightarrow A: \ ID_A, ID_B, n_S, \{K_S\}_{\pi_A}, \{K'_S\}_{\pi_B}$
3. $A \rightarrow B: \ \mathsf{Enc}_S(ID_A, ID_B, n_A, n'_A, c_A, \{n_S\}_{\pi_A}), n_S, r_A, \{K'_S\}_{\pi_B}$
4. $B \rightarrow S: \ \mathsf{Enc}_S(ID_A, ID_B, n_A, n'_A, c_A, \{n_S\}_{\pi_A}), \mathsf{Enc}'_S(ID_B, ID_A, n_B, n'_B, c_B, \{n_S\}_{\pi_B})$
5. $S \rightarrow B: \ \{n_A, \mathbf{K} \oplus n'_A\}_{\pi_A}, \{n_B, \mathbf{K} \oplus n'_B\}_{\pi_B}$
6. $B \rightarrow A: \ \{n_A, \mathbf{K} \oplus n'_A\}_{\pi_A}, \{H_1(r_A), r_B\}_{\mathbf{K}}$
7. $A \rightarrow B: \ \{H_2(r_B)\}_{\mathbf{K}}$

Protocol 8.22: GLNS secret public key protocol

B to encrypt the request for a session key in messages 3 and 4; in these messages $\mathsf{Enc}_S(.)$ denotes encryption with K_S and $\mathsf{Enc}'_S(.)$ denotes encryption with K'_S.

Gong *et al.* remarked on the similarity with the Otway–Rees protocol (Protocol 3.26), at least in terms of the general structure. But there are a number of additional fields included in Protocol 8.22 which are worth examining. A (and B similarly) chooses two nonces n_A and n'_A which are transmitted to the server. The first of these is returned with $\mathbf{K}$, with the usual purpose of allowing A to verify the freshness of this key. The second nonce is used to mask $\mathbf{K}$; its purpose is to prevent the adversary from performing a password-guessing attack. If n'_A is omitted from message 5 then the adversary can decrypt the message with a candidate password $\tilde{\pi}_A$ and obtain a candidate session key $\tilde{K}_{AB}$. The adversary can then decrypt the second encrypted field in message 6 using $\tilde{K}_{AB}$ and check for the correct $H_1(r_A)$ value, in order to confirm whether $\tilde{\pi}_A$ is the correct password of A. The encrypted fields using $\mathbf{K}$ in the final two messages are intended to provide key confirmation.

The value c_A (and c_B symmetrically) is a random 'confounder' used to ensure that the encryption with K_S in message 4 is randomised. If c_A were not used, and the public key encryption were deterministic, then an adversary who obtained an old $\mathbf{K}$ value could test a guess $\tilde{\pi}$ for π_A as follows. First the field $\{K_S\}_{\pi_A}$ in message 2 is decrypted with $\tilde{\pi}$ to obtain a possible public key $\tilde{K}_S$. Candidate values for n_A and n'_A can also be obtained by decrypting the first field in message 6 with $\tilde{\pi}$ (remember that we assume the adversary knows $\mathbf{K}$). Then all the required inputs are available to re-encrypt the first field in message 3, and check this against the actual value sent in order to verify whether $\tilde{\pi}$ is correct. The 'confounders' c_A and c_B can be omitted by requiring the asymmetric encryption algorithm to provide semantic security, since this ensures that the encryption is randomised. Furthermore, the asymmetric encryption algorithm must provide non-malleability; otherwise, the protocol is easily defeated by an adversary C who can change the encrypted name of B to C in message 4 and hence masquerade as B to A.

Just as we have seen previously with Diffie–Hellman-based EKE, the choice of the correct encryption algorithms is a crucial matter. Patel [603] demonstrated that use of RSA as the asymmetric encryption algorithm in Protocol 8.22 allows an active attack similar to the one against OKE described in Sect. 8.5.2. However, Gong *et al.* did specify that the encryption algorithm must have the property that any random number could be a public key, a property not possessed by RSA. A protocol using an algorithm that matches the password representation to the encryption algorithm, such as used that in PAK, appears more promising.

Although Protocol 8.22 seems to be regarded in the literature as a fundamentally sound one, we note that an attack is possible unless specific precautions are taken by users in an implementation. To perpetrate the attack, the adversary masquerades as the server in order to obtain encrypted messages from A and/or B which can be used to verify password guesses. Specifically, the adversary can guess values for π_A and π_B, generate new ephemeral keys K_S and K'_S, and use the guessed passwords to form a possibly correct message 2. This message can be sent back to A following a protocol initiation by A. The adversary can collect the subsequent message 4 intended by B for the server. Now the adversary can detect if either of the password guesses was correct, since, if so, the identifiers A and B will be present in the messages when decrypted with the generated private keys. Further guesses can be verified or discarded in the same way. Notice that A must accept a random value for the field $\{K_S\}_{\pi_A}$ in message 2; otherwise an offline guessing attack is possible.

If the above attack is to be prevented, two precautions must be taken in any implementation. Firstly, failed protocol attempts must be logged by users and the password disabled after a small number of failures. Although such precautions are usually taken for failed logins at a server, it is less usual for this matter to be considered for users. Secondly, it must not be possible for the adversary to start multiple parallel runs of the protocol, since otherwise the correct passwords can be found before any protocol run has failed. It seems likely that in some applications users would be required to enter the password for each protocol run, but in others the user's computing device might cache the password and reuse it for multiple runs.

Tsudik and Van Herreweghen [716], and later Gong [318], have proposed variants of Protocol 8.22 aimed at simplifying it and improving its efficiency. Protocol 8.23 shows the version of Tsudik and Van Herreweghen, which follows the same basic design but reduces the amount of material that needs to be encrypted with the ephemeral public keys. In distinction to Protocol 8.22, there is no attempt to provide key confirmation.

Inclusion of the identities of A and B in the fields encrypted with π_A and π_B in message 2 is critical to the security of Protocol 8.23. This is because there is nothing else that allows the principals to know which party the session key is shared with. The nonces n_B and n_A, when XOR'd with the encrypted fields using π_B and π_A in message 5, act as 'confounders'. The outer XOR prevents B (and A similarly), having also obtained $\mathbf{K}$, from mounting a brute force guessing attack on π_A.

Ding and Horster [254] found an online attack on Protocol 8.23, one of several *undetectable* online attacks that they discovered. The important feature of such attacks is that an insider can perpetrate them in such a way that the server cannot detect

S has passwords π_A and π_B. S chooses ephemeral encryption keys K_S and K'_S.
A has password π_A. A chooses random value n_A.
B has password π_B. B chooses random value n_B.

1. $A \rightarrow S: \ ID_A, ID_B$
2. $S \rightarrow A: \ \{K_S \oplus ID_B\}_{\pi_A}, \{K'_S \oplus ID_A\}_{\pi_B}$
3. $A \rightarrow B: A, \mathsf{Enc}_S(n_A), \{K'_S \oplus ID_A\}_{\pi_B}$
4. $B \rightarrow S: \ ID_A, ID_B, \mathsf{Enc}'_S(n_B), \mathsf{Enc}_S(n_A)$
5. $S \rightarrow B: \ n_B \oplus \{n_B \oplus \mathbf{K}\}_{\pi_B}, n_A \oplus \{n_A \oplus \mathbf{K}\}_{\pi_A}$
6. $B \rightarrow A: \ n_A \oplus \{n_A \oplus \mathbf{K}\}_{\pi_A}$

Protocol 8.23: Simplified GLNS secret public key protocol

that they have taken place and so cannot restrict the number of repetitions. Therefore the adversary may be able to conduct an online exhaustive key search. In Ding and Horster's attack on Protocol 8.23, the insider adversary B masquerades as A to initiate the protocol (the real A does not take part in the attack). The adversary collects the response from S in message 2 and makes a guess at π_A. This allows the adversary to forge a corresponding value for message 3 and hence send a trial value to S in message 4. On receipt of message 5, the adversary can verify whether the guess for π_A was correct, since this means that the same $\mathbf{K}$ value will be found. This attack is undetectable by S as long as there is no redundancy in the encrypted fields in message 4.

A closer look at the attack of Ding and Horster reveals that there may be no need for the adversary to be an insider in order to mount the attack. Instead, the adversary can masquerade as both A and B by guessing both passwords π_A and π_B. After initiating the protocol by masquerading as A, the adversary finds candidate values for K_S and K'_S and uses these to construct message 4. On receipt of message 5, incorrect candidate passwords can be eliminated if the decrypted values of $\mathbf{K}$ are different. In this version of the attack, both passwords must be found and so the maximum number of trials required before success is the square of the number of passwords. Depending on the size of the password space, this may still be a feasible attack.

In Gong's variant [318], the ephemeral secret public keys are generated by A and B instead of S and consequently the number of protocol messages is reduced to five. This variant is shown in Protocol 8.24, where K_A and K_B are the ephemeral public keys chosen by A and B, respectively, and $\mathsf{Enc}_A(.)$ and $\mathsf{Enc}_B(.)$ denote encryption with these keys. The confounders c_S and c'_S are chosen to prevent guessing of encrypted contents and, as elsewhere, can be omitted if a public key encryption algorithm with semantic security is used.

Ding and Horster [254] showed that there is an undetectable online attack on Protocol 8.24 too. Again, the insider adversary B masquerades as A in a protocol run that looks normal to S. Specifically, the adversary guesses π_A and chooses a value for K_A (as well as for K_B). On receipt of message 3, the adversary can decrypt both

S has passwords π_A and π_B. S chooses random values c_S and c'_S.
A has password π_A. A chooses random value n_A and ephemeral encryption key K_A.
B has password π_B. B chooses random value n_B and ephemeral encryption key K_B.

1. $A \rightarrow B: n_A, \{K_A\}_{\pi_A}$
2. $B \rightarrow S: n_A, \{K_A\}_{\pi_A}, n_B, \{K_B\}_{\pi_B}$
3. $S \rightarrow B: \mathsf{Enc}_A(ID_A, ID_B, c_S, \mathbf{K}, \{n_A\}_{\pi_A}), \mathsf{Enc}_B(ID_B, ID_A, c'_S, \mathbf{K}, \{n_B\}_{\pi_B})$
4. $B \rightarrow A: \mathsf{Enc}_A(ID_A, ID_B, c_S, \mathbf{K}, \{n_A\}_{\pi_A}), \{n_A\}_{\mathbf{K}}, n_B$
5. $A \rightarrow B: \{n_B\}_{\mathbf{K}}$

Protocol 8.24: Optimal GLNS secret public key protocol

encrypted messages and see if they give the same value for **K**: if so, then the guess for π_A was probably correct. Notice that S must accept random values in message 2; otherwise an offline guessing attack is possible. However, no explicit encryption algorithm for use with π_A and π_B was specified by Gong.

As with the insider attack on Protocol 8.23, this attack may be extended to an outsider attack by guessing both candidate passwords together. Indeed, an outsider need guess only one of the passwords. For example, the adversary can guess π_A, choose a value for K_A, and send a message to B as if it were from A. On receipt of message 3, the adversary can detect if this guess was correct, since, if so, the identifiers A and B will be present in the encrypted field using K_A in this message, when decrypted with the generated private key. This attack is not detectable by the server S, but B will detect a failed password guess, since the adversary is not able to find the correct **K** value and so cannot form the correct message 5. Therefore similar remarks to those regarding the related attack on Protocol 8.22 apply.

8.6.2 Steiner, Tsudik and Waidner Three-Party EKE

Steiner *et al.* [691] proposed Protocol 8.25 as a direct generalisation of Diffie–Hellman-based EKE. They remarked that, in distinction to many three-party protocols, the server does not generate, and indeed cannot obtain, the session key. Each of A, B and S generates an ephemeral private key, r_A, r_B, and r_S, respectively, and the shared secret for A and B is $\mathbf{Z} = g^{r_A r_B r_S}$. The encrypted fields using **K** in the final two messages are intended as 'authenticators' that can be used for key confirmation of **Z**.

Protocol 8.25 provides forward secrecy against eavesdroppers. However, Ding and Horster [254] showed that it is vulnerable to online undetectable guessing, as shown in Attack 8.1. The insider adversary C records an old protocol run with A and replays A's input in message 2. Only the interaction with S in messages 2 and 3 is relevant for the attack. By guessing π_A, the adversary can find the corresponding value $\tilde{t}_A$ that A would have sent and use this value as C's input to message 2. On receipt of message 3, C can check if both returned values are the same in order to confirm a correct guess of π_A.

S has passwords π_A and π_B. S chooses random value r_S.
A has password π_A. A chooses random value r_A and calculates $t_A = g^{r_A}$.
B has password π_B. B chooses random value r_B and calculates $t_B = g^{r_B}$.

1. $A \rightarrow B: \{t_A \oplus ID_B\}_{\pi_A}$
2. $B \rightarrow S: A, \{t_A \oplus ID_B\}_{\pi_A}, \{t_B \oplus ID_A\}_{\pi_B}$
3. $S \rightarrow B: t_A^{r_S}, t_B^{r_S}$
4. $B \rightarrow A: t_B^{r_S}, \{\text{Message 1}\}_{\mathbf{K}}$
5. $A \rightarrow B: \{\{\text{Message 1}\}_{\mathbf{K}}\}_{\mathbf{K}}$

Protocol 8.25: Steiner, Tsudik and Waidner three-party EKE

C has password π_C and recorded message $\{t_A \oplus ID_C\}_{\pi_A}$. C guesses A's password is $\tilde{\pi}$ and chooses $\tilde{t}_A$ such that $\{t_A \oplus ID_C\}_{\pi_A} = \{\tilde{t}_A \oplus ID_C\}_{\tilde{\pi}}$.

2. $C \rightarrow S: A, \{t_A \oplus ID_C\}_{\pi_A}, \{\tilde{t}_A \oplus ID_A\}_{\pi_C}$
3. $S \rightarrow C: t_A^{r_S}, \tilde{t}_A^{r_S}$

C checks if $t_A^{r_S} = \tilde{t}_A^{r_S}$. If so, $\tilde{\pi} = \pi_A$.

Attack 8.1: Ding and Horster's attack on Protocol 8.25

Lin *et al.* [493] also discovered an offline guessing attack on Protocol 8.25, as shown in Attack 8.2. The adversary C masquerades as both A and S in order to find the password of B. The values X and Y are chosen randomly by C and are simply placeholders for missing values. The value r_C is chosen by C in place of $r_A r_S$ in a real protocol run. C knows that B will calculate $\mathbf{Z} = (g^{r_C})^{r_B} = t_B^{r_C}$, so messages 2 and 4 can be used to check a guess for π_B.

C chooses random values X, Y and r_C.

1. $C_A \rightarrow B: X$
2. $B \rightarrow C_S: A, X, \{t_B \oplus ID_A\}_{\pi_B}$
3. $C_S \rightarrow B: g^{r_C}, Y$
4. $B \rightarrow C_A: Y, \{\text{Message 1}\}_{\mathbf{K}}$

C guesses $\pi_B = \tilde{\pi}$ and decrypts $\{t_B \oplus A\}_{\pi_B}$ with $\tilde{\pi}$ to obtain $\tilde{t}_B$ and calculates $\tilde{\mathbf{Z}} = \tilde{t}_B^{r_C}$. Message 4 is used to check if guess is correct.

Attack 8.2: Lin–Sun–Hwang attack on Protocol 8.25

Lin *et al.* [493, 494] proposed two alternative versions of Protocol 8.25. One requires the client to know the correct public key of the server, while the later one, from 2001, avoids the known attacks without the need for a server public key.

8.6.3 GLNS Protocols with Server Public Keys

In addition to their 'secret public key' protocol examined in Sect. 8.6.1, Gong *et al.* [319] published several protocols assuming server public keys are available to clients. We examine some of these and subsequent variants from other authors. Unfortunately, none of these protocols carries a security proof.

Protocol 8.26 shows the 'compact' version of the GLNS protocol for establishing a new session key between A and B, who initially share passwords π_A and π_B with the server S. There is a strong similarity with messages 3 to 7 of Protocol 8.22.

S has passwords π_A and π_B.
A has password π_A. A chooses random values n_A, n'_A, c_A, r_A and timestamp T_A.
B has password π_B. B chooses random values n_B, n'_B, c_B, r_B and timestamp T_B.

1. $A \rightarrow B$: $\mathsf{Enc}_S(ID_A, ID_B, n_A, n'_A, c_A, \{T_A\}_{\pi_A}), r_A$
2. $B \rightarrow S$: $\mathsf{Enc}_S(ID_A, ID_B, n_A, n'_A, c_A, \{T_A\}_{\pi_A}), \mathsf{Enc}_S(ID_B, ID_A, n_B, n'_B, c_B, \{T_B\}_{\pi_B})$
3. $S \rightarrow B$: $\{n_A, \mathbf{K} \oplus n'_A\}_{\pi_A}, \{n_B, \mathbf{K} \oplus n'_B\}_{\pi_B}$
4. $B \rightarrow A$: $\{n_A, \mathbf{K} \oplus n'_A\}_{\pi_A}, \{H_1(r_A), r_B\}_{\mathbf{K}}$
5. $A \rightarrow B$: $\{H_2(r_B)\}_{\mathbf{K}}$

Protocol 8.26: GLNS compact protocol

As in Protocol 8.22, the confounders c_A and c_B may be omitted by requiring that the asymmetric encryption algorithm provides semantic security. Also, the asymmetric encryption algorithm must provide non-malleability. If the long-term decryption key of S is compromised then n_A and n'_A can be found, allowing a brute force search for π_A, and consequently revealing the session key. Therefore we conclude that forward secrecy is not provided.

The encrypted timestamp $\{T_A\}_{\pi_A}$ included in the first message is used by S to ensure that the message is freshly generated by A. Without this timestamp, the adversary could replay message 1 and obtain two messages $\{n_A, \mathbf{K} \oplus n'_A\}_{\pi_A}$ and $\{n_A, K'_{AB} \oplus n'_A\}_{\pi_A}$, the only difference being in the encrypted session keys. Then the adversary could again mount a brute force attack on π_A since a correct guess can be identified when the first components of the two decrypted messages are identical. (This attack was observed by Tsudik and Van Herreweghen [716], as well as by Gong *et al.* [319].) The use of timestamps requires the server to record all messages received for a period equal to the maximum time window allowed for clock differences and message delay.

In order to remove this drawback, Gong *et al.* also provided a version of the protocol in which the server generates a nonce that must be sent with the client requests;

its drawback, in turn, is the addition of two messages to the protocol. Tsudik and Van Herreweghen [716], and later Gong [318], proposed a variant of this nonce-based protocol, aimed at simplifying it and reducing computational requirements. Protocol 8.27 shows Gong's optimal version, in which the number of messages exchanged is reduced to five. Gong [318] observed that this is the same as the number of messages used in the timestamp-based version.

S has passwords π_A and π_B.
A has password π_A. A chooses random values $n_A, n'_A, n''_A, c_A, r_A$.
B has password π_B. B chooses random values $n_B, n'_B, n''_B, c_B, r_B$.

1. $A \rightarrow B: \mathsf{Enc}_S(ID_A, ID_B, n_A, n'_A, c_A, n''_A, \{n''_A\}_{\pi_A}), r_A$
2. $B \rightarrow S: \mathsf{Enc}_S(ID_A, ID_B, n_A, n'_A, c_A, n''_A, \{n''_A\}_{\pi_A}), \mathsf{Enc}_S(ID_B, ID_A, n_B, n'_B, c_B, n''_B, \{n''_B\}_{\pi_B})$
3. $S \rightarrow B: \{n_A, \mathbf{K} \oplus n'_A\}_{n''_A}, \{n_B, \mathbf{K} \oplus n'_B\}_{n''_B}$
4. $B \rightarrow A: \{n_A, \mathbf{K} \oplus n'_A\}_{n''_A}, \{H_1(r_A), r_B\}_{\mathbf{K}}$
5. $A \rightarrow B: \{H_2(r_B)\}_{\mathbf{K}}$

Protocol 8.27: Optimal GLNS nonce-based protocol

The main difference between the nonce-based and timestamp-based protocols is that in the former A and B send a third nonce to S, which is used both to authenticate them to S and also as a shared secret to encrypt the session key in the reply from S.

8.6.4 Three-Party Protocol of Yen and Liu

The protocols of Yen and Liu [758] use ideas from the simplified GLNS three-party EKE of Tsudik and Van Herreweghen (Protocol 8.23). By making use of a server public key, they aim to avoid the need for ephemeral public keys. Protocol 8.28 shows their main protocol, in which the server generates the session key $\mathbf{K}$. They also proposed variants in which either the initiator or the responder can generate $\mathbf{K}$.

S has passwords π_A and π_B. S chooses $\mathbf{K}$.
A has password π_A. A chooses random values n_A, n'_A.
B has password π_B. B chooses random value n_B.

1. $A \rightarrow B: ID_A, n'_A, \mathsf{Enc}_S(n_A \oplus \pi_A, n_A \oplus ID_B \oplus n'_A)$
2. $B \rightarrow S: ID_A, ID_B, \mathsf{Enc}_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A), \mathsf{Enc}_S(n_B \oplus \pi_B, n_B \oplus A \oplus n'_A)$
3. $S \rightarrow A: n_A \oplus \{n_A \oplus \mathbf{K}\}_{\pi_A}, \{n'_A\}_{\mathbf{K}}, n_B \oplus \{n_B \oplus \mathbf{K}\}_{\pi_B}$
4. $A \rightarrow B: n_B \oplus \{n_B \oplus \mathbf{K}\}_{\pi_B}, \{n'_A + 1\}_{\mathbf{K}}$

Protocol 8.28: Yen–Liu protocol

On receipt of message 2, S decrypts both ciphertexts to obtain two pairs of values, say X_A, Y_A and X_B, Y_B. Then S sets $n_A = X_A \oplus \pi_A$ and $n_B = X_B \oplus \pi_B$ and checks that $n_A \oplus Y_A \oplus B = n_B \oplus Y_B \oplus A$. If not, then S aborts the protocol. Despite a detailed analysis by its authors, the Yen–Liu protocol does not provide authentication of both parties. An insider adversary C is able to masquerade as B and successfully complete a protocol run with A, including obtaining the new session key.

1. $A \to C_B : ID_A, n'_A, \mathsf{Enc}_S(n_A \oplus \pi_A, n_A \oplus ID_B \oplus n'_A)$
2. $C \to S : \; ID_A, ID_C, \mathsf{Enc}_S(n_A \oplus \pi_A, n_A \oplus B \oplus n'_A), \mathsf{Enc}_S(n_C \oplus \pi_C, n_C \oplus A \oplus n''_A)$
3. $S \to C_A : n_A \oplus \{n_A \oplus K_{AC}\}_{\pi_A}, \{n''_A\}_{K_{AC}}, n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}$
3′. $C_S \to A : n_A \oplus \{n_A \oplus K_{AC}\}_{\pi_A}, \{n'_A\}_{K_{AC}}, n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}$
4. $A \to C_B : n_C \oplus \{n_C \oplus K_{AC}\}_{\pi_C}, \{n'_A + 1\}_{K_{AC}}$

Attack 8.3: Attack on Protocol 8.28

In Attack 8.3, A wishes to complete the protocol with B, but in fact completes it with the adversary C. After receiving message 1 from A, C generates a new value n''_A such that $n''_A \oplus C = n'_A \oplus B$. This enables C to send a correct message 2 to S as if A is intending to run the protocol with C. C needs to intercept message 3 from S in order to replace the field $\{n''_A\}_{K_{AC}}$ with the field $\{n'_A\}_{K_{AC}}$ expected by A. This can be done, since C is able to extract K_{AC} from message 3. When A receives the altered message 3′, the session key K_{AC} will be extracted by A and used to confirm that the value n'_A was correctly received. Thus A will accept the key as shared with B, whereas it is actually shared with C.

8.6.5 Generic Protocol of Abdalla, Fouque and Pointcheval

Abdalla *et al.* [16] seem to have been the first to achieve a three-party PAKE protocol with a proof of security. However, this was not a single protocol but rather a generic method of combining any normal two-party PAKE protocol with any server-based key distribution protocol in the manner of a protocol compiler. Protocol 8.29 illustrates the process.

A and S initially share a password π_A and, similarly, B and S initially share a password π_B. These passwords are used in two-party PAKE protocols (denoted 2-PAKE) during stage 1. Evidently, the compiler still works in the case that the generic two-party PAKE protocol is an augmented-type protocol where S possesses only an image of the passwords. In stage 2, S uses the two keys established during stage 1 to distribute a MAC key, k_m. Finally, in stage 3 the parties run Diffie–Hellman key agreement with the messages authenticated using k_m.

Abdalla *et al.* [16] provided a security proof which says that, given a secure two-party PAKE, a secure key distribution protocol and a secure MAC, the compiled protocol is secure based on the decisional Diffie–Hellman assumption. The security model used is based on the BPR00 model, but is slightly stronger in that it allows

Information shared between A and S: Password π_A.
Information shared between B and S: Password π_B.

A	S	B
Stage 1		
	2-PAKE between S and A Input: π_A Output: sk_A	2-PAKE between S and B Input: π_B Output: sk_B
Stage 2		
	Key distribution from S to A Input: sk_A Output: k_m	Key distribution from S to B Input: sk_B Output: k_m
Stage 3		
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$	$\xrightarrow{t_A, \mathsf{MAC}_{k_m}(t_A, ID_B, ID_A)}$	$r_B \in_R \mathbb{Z}_q$
	$\xleftarrow{t_B, \mathsf{MAC}_{k_m}(t_B, ID_A, ID_B)}$	$t_B = g^{r_B}$
	$\mathbf{K} = g^{r_A r_B}$	

Protocol 8.29: Abdalla–Fouque–Pointcheval compiler for three-party PAKE

multiple test queries in place of reveal queries. This stronger notion is required for the input two-party PAKE and is also achieved for the compiled protocol. Abdalla *et al.* [16] pointed out that many well-known PAKE protocols, such as PAK (Protocol 8.3) and KOY (Protocol 8.10), do satisfy the stronger notion so there are several ways of instantiating the compiler with a concrete protocol. The key distribution protocol used in Stage 2 of the compiler need only be secure in the usual BR model and can be instantiated, for example, by the Bellare and Rogaway three-party protocol [78] (Protocol 3.41).

In addition to security against an outside adversary, the security analysis also showed that an *honest but curious* insider adversary, which runs the protocol honestly, does not get to learn anything useful about the shared secret $\mathbf{K}$. However, the compiled protocol may not be secure against undetectable online guessing attacks, which we consider next.

8.6.6 Stronger Security Models for Three-Party PAKE

As pointed out by Abdalla *et al.* [16], a generic compiler cannot be expected to yield the most efficient concrete protocol. There have been many three-party PAKE protocols proposed attempting to achieve efficiency improvements over the generic constructions. Regrettably, this is an area where many protocols are still proposed

without a proof of security. Furthermore, many protocols have been proposed with a claimed security proof later found to be faulty (see, for example the discussion and references provided by Nam *et al.* [570].) A particular problem has been security against the undetectable online attacks of Ding and Horster [254] discussed in Sect. 8.6.2.

The usual security model for analysis of PAKE protocols is that of Bellare, Pointcheval and Rogaway [74], which we refer to as BPR00. When using this model for two-party PAKE protocols, it is natural to restrict the adversary's access to send queries since such queries allow the adversary to test a password by guessing and running the protocol as a client. In practice, unsuccessful tests will be noticed by the server, and the client account will be locked after a small number of attempts. It is widely understood that such attacks cannot be completely prevented. However, in a three-party PAKE it is not necessarily the aim of the server to explicitly authenticate the two clients who wish to share the session key, and therefore it can be impossible for the server to notice failed password tests. Wang and Hu [729] noticed that Abdalla *et al.*'s compiler may not protect against undetectable online attacks if the protocols used as building blocks do not provide client authentication. For example, this may happen if PPK (Protocol 8.4) is used for the two-party PAKE. This does not indicate any error in the security analysis of Abdalla *et al.* [16], but rather shows that this kind of attack is not captured in the model that they used.

Wang and Hu [729] proposed to add client authentication as part of the security definition for three-party PAKE. When explicit authentication is provided, the server can count failed password attempts and lock out user accounts according to some policy, as in the two-party case. Wang and Hu [729] provided a compiler shown as Protocol 8.30, related to that of Abdalla *et al.* [16] but replacing the key distribution protocol with a custom method of explicit authentication.

Note that, unlike stage 3 in Protocol 8.29, stage 2 in Protocol 8.30 involves the server in the protocol messages, specifically in order that the server can authenticate the clients. It is also evident that this compiler results in more efficient protocols than Protocol 8.29 if the two-party PAKE used is the same in both cases.

Instead of demanding explicit client authentication, Nam *et al.* [571] enhanced the BPR00 model to explicitly capture undetectable online attacks. They also added consideration of insider attacks to their model through the use of corrupt queries. Nam *et al.* [571] designed a generic compiler, very similar to Protocol 8.30, for obtaining secure protocols within their model. It is not yet clear which model is the best one to analyse three-party PAKE protocols while optimal concrete protocols remain to be established.

8.6.7 Three-Party Protocol of Yoneyama

Yoneyama [762, 763] designed a concrete three-party PAKE protocol. Messages to the server are encrypted with the server public key and the server incorporates its own randomness, which is included in the shared secret $\mathbf{Z} = g^{r_A r_B r_S}$. Protocol 8.31 shows the messages.

Information shared between A and S: Password π_A.
Information shared between B and S: Password π_B.

A	S	B
Stage 1		
	2-PAKE between S and A Input: π_A Output: sk_A	2-PAKE between S and B Input: π_B Output: sk_B
Stage 2		
$r_A \in_R \mathbb{Z}_q$ $t_A = g^{r_A}$		$r_B \in_R \mathbb{Z}_q$ $t_B = g^{r_B}$
$t_A, \mathsf{MAC}_{sk_A}(t_A, ID_A, ID_B) \longrightarrow$		$t_A, \mathsf{MAC}_{sk_B}(t_A, ID_A, ID_B) \longrightarrow$
$t_B, \mathsf{MAC}_{sk_A}(t_B, ID_A, ID_B) \longleftarrow$	Check MAC	$t_B, \mathsf{MAC}_{sk_B}(t_B, ID_A, ID_B) \longleftarrow$
	Check MAC $\mathbf{K} = g^{r_A r_B}$	

Protocol 8.30: Wang–Hu compiler for three-party PAKE

We can identify similarities between Protocol 8.31 and the PAK protocol (Protocol 8.3) with regard to the way that the password is used to mask the Diffie–Hellman values. A difference is that in Protocol 8.31 the t_A and t_B values are hidden using encryption when sent to S. In addition, there is a similar requirement that the hash functions H_1 and H_2 map to the group $\mathbb{G}$. Yoneyama [763] provided a security proof in a model which is related to the eCK model, except that leakage of ephemeral keys is not allowed for the target session. It is assumed that the encryption scheme is CPA-secure and that the DDH assumption holds. The hash functions H_1, H_2, and H_3 are modelled as random oracles. The messages C_A and C_B allow the server to explicitly authenticate the users and thereby ensure that online guesses are detectable.

8.6.8 Cross-Realm PAKE Protocols

In the three-party PAKE setting it is assumed that both of the clients wishing to establish a session key share their passwords with the same server. This applies whether the server has a public key (see Sects. 8.6.4 and 8.6.7) or not (see Sect. 8.6.2). In order to make such protocols more scalable, it is natural to consider a situation where the clients A and B use *different* servers, S_A and S_B, respectively. This is clearly a more complex situation, involving four entities, and calls for more complex threat models. Now we may be concerned about malicious servers attempting to learn passwords of clients from other domains. We will refer to such protocols as *cross-realm*. Other names used include *cross-domain* and *4-PAKE* protocols.

There are different communications architectures possible for cross-realm protocols. In particular, some protocols assume direct communication between clients,

Information shared between A and S: Password π_A.
Information shared between B and S: Password π_B.
Shared information: Hash functions H_1, H_2, H_3.

A		S		B
$P_A = H_1(ID_A, ID_B, \pi_A)$				$P_B = H_1(ID_B, ID_A, \pi_B)$
$r_A \in_R \mathbb{Z}_q$				$r_B \in_R \mathbb{Z}_q$
$t_A = g^{r_A}$				$t_B = g^{r_B}$
$m_A = t_A P_A$				$m_B = t_B P_B$
$C_A = \mathsf{Enc}_S(m_A, \pi_A)$	$\xrightarrow{C_A}$		$\xleftarrow{C_B}$	$C_B = \mathsf{Enc}_S(m_B, \pi_B)$
		Decrypt C_A and C_B		
		Verify π_A and π_B		
		$t_A = m_A / P_A, t_B = m_B / P_B$		
		$r_S \in_R \mathbb{Z}_q, N \in \{0,1\}^*$		
		$t'_A = t_A^{r_S}, t'_B = t_B^{r_S}$		
		$P'_A = t'_A \cdot H_2(N, \pi_B, m_B)$		
		$P'_B = t'_B \cdot H_2(N, \pi_A, m_A)$		
	$\xleftarrow{N, C_B, P'_A, P'_B}$		$\xrightarrow{N, C_A, P'_A, P'_B}$	
$t'_B = P'_B / H_2(N, \pi_A, m_A)$				$t'_A = P'_A / H_2(N, \pi_B, m_B)$
$\mathbf{Z} = (t'_B)^{r_A}$				$\mathbf{Z} = (t'_A)^{r_B}$
		$\mathbf{K} = H_3(ID_A, ID_B, ID_S, C_A, C_B, P'_A, P'_B, \mathbf{Z})$		

Protocol 8.31: Yoneyama protocol for three-party PAKE

while others instead use communication between the servers involved. Some protocols assume that servers already share keys in order to provide a secure channel between them, while other do not. As in the case of three-party PAKE, there have been many protocol proposals without a formal security analysis, while those which do have a security proof often use different models. Chen *et al.* [196] provided a good overview of much of this work. In principle, cross-realm protocols do not require servers to have public keys if there is some other way for them to communicate securely. However, in practice most protocols do assume the existence of certified server public keys.

Yoneyama [764] analysed a variant of Protocol 8.31 for the cross-realm setting. The two protocols are very similar but the cross-realm version includes messages between the servers, sent on an authenticated channel, allowing both A and B to compute a shared secret $\mathbf{Z} = g^{r_A r_B r_{SA} r_{SB}}$, where r_A, r_B, r_{SA}, r_{SB} are chosen by A, B, S_A, S_B respectively.

Chen *et al.* [196] provided a generic protocol for the cross-realm case in the same vein as the Abdalla *et al.* generic 3-PAKE construction of Protocol 8.29. The building

blocks for the protocol consist of a two-party PAKE, denoted 2-PAKE, secure MAC and signature schemes, and a key agreement protocol.

Public information: Public keys PK_{S_A} of S_A and PK_{S_B} of s_B
Information shared between A and S_A: Password π_A.
Information shared between B and S_B: Password π_B.

A	S_A	S_B	B
Stage 1			
$r_A \in_R \mathbb{Z}_q$			$r_B \in_R \mathbb{Z}_q$
$t_A = g^{r_A}$	$\xrightarrow{t_A}$		$t_B = g^{r_B}$
	$\xleftarrow{t_B}$		
Stage 2			
	2-PAKE between A and S_A		2-PAKE between B and S_B
	Input: π_A; Output: sk_A		Input: π_B; Output: sk_B
Stage 3			
	$t_A, \mathsf{MAC}_{k_{sk_A}}(ID_A, ID_B, t_A, t_B) \xrightarrow{}$		$t_B, \mathsf{MAC}_{k_{sk_B}}(ID_B, ID_A, t_B, t_A) \xleftarrow{}$
	$\sigma_A = \mathsf{Sig}_{S_A}(ID_A, ID_B, t_A, t_B, PK_{S_B}) \xleftarrow{}$		$\sigma_B = \mathsf{Sig}_{S_B}(ID_B, ID_A, t_B, t_A, PK_{S_A}) \xrightarrow{}$
Stage 4			
	Key agreement between A and B		
	Input: $t_A, t_B, \sigma_A, \sigma_B$ Output: $\mathbf{K}$		
	$\Longleftrightarrow$		

Protocol 8.32: Chen–Lim–Yang generic cross-realm PAKE

In stage 1 of Protocol 8.32, clients A and B exchange ephemeral keys t_A and t_B, which will be used later in stage 4 to agree on a session key. The purpose of stages 2 and 3 is to allow the servers S_A and S_B to provide to the clients an assurance that the exchange ephemeral keys are authentic. First, in stage 2 shared, keys are established between each client and its server using 2-PAKE. Then, in stage 3, the clients provide authenticated versions of the (claimed) t_A and t_B values to the servers, which respond by signing these. Finally in stage 4, the clients run their key agreement protocol, such as basic Diffie–Hellman, using the ephemeral keys, and authenticate the (t_A, t_B) pair by checking the signatures from both servers. Note that the inclusion of the public server keys in the signatures sent in stage 3 is a simple way for the clients to authenticate the signing public key of the server from its peer's domain.

Notice that, in distinction to many other protocol designs in this setting, there is no need for any secure channel between S_A and S_B in Protocol 8.32; indeed, no interaction between servers occurs at all. The protocol includes explicit authentication of A and B to the servers S_A and S_B through the MAC tags sent in stage 3. Although

the model used by Chen *et al.* does not explicitly model undetectable online attacks, this feature does seem to prevent such attacks. Chen *et al.* [196] provided security theorems and proofs in the model of Abdalla *et al.* [16] and assumed that the protocol run in stage 4 is secure in the BR95 model.

Protocol 8.32 is not the most generic protocol designed by Chen *et al.* [196]; they proposed a number of other generic variants, including one which accommodates reuse of ephemeral keys, and one which replaces the public-key key agreement protocol in stage 4 with a symmetric-key one. Overall, there are many variants with many options in this set of protocols and it is difficult to compare their different properties.

8.6.9 Comparing Three-Party PAKE Protocols

As we have seen for other types of PAKE protocols, the early three-party PAKE protocols are characterised by a lack of formal analysis and a gradual understanding of the ways to design secure protocols. The GLNS protocols and their variants examined in Sects. 8.6.1, 8.6.2 and 8.6.3 fall into this category and are vulnerable to various attacks, particularly the undetectable online guessing attack. While there are some interesting structural differences between such protocols, such as whether or not a server public key is used, a detailed comparison of such protocols does not seem worthwhile.

More modern protocols usually have a proof of security, although this is still not always the case. Moreover, many protocols have proofs in quite different security models, making them hard to compare with regard to security properties. Another characteristic of the more modern constructions is that they are usually *generic*, relying on the use of existing two-party PAKE protocols as building blocks. Protocol 8.31 of Yoneyama is one exception to this. Cross-realm protocols, discussed in Sect. 8.6.8, are an area where we can expect new results.

8.7 Group PAKE Protocols

One natural generalisation of PAKE protocols is to allow the number of parties to be larger than two, as in the group key exchange protocols considered in Chap. 9. The situation we focus on in this section is when multiple users share the *same* password (or any short secret). Such a scenario could arise in various realistic real-world applications such as adhoc meetings and civil emergencies.

Before considering the shared-password scenario, we note that there is also the possibility that users who share *different* passwords may desire to set up a shared group key. This alternative can evidently only be achieved with the help of an online server (or possibly multiple servers) sharing each of the user passwords. Such protocols would generalise the three-party PAKE protocols examined in Sect. 8.6. This scenario has not received much attention in the literature so far. One proposal, from Byun *et al.* [172], was broken by Phan and Goi the following year [613]. There are many different options possible here, including augmented versions, protocols with

or without server public keys, and protocols in cross-realm settings. Potentially many new protocols could be proposed and analysed, although it is not clear how many of these would be of independent interest.

Now we turn back to the shared-password scenario, where a group of parties share the same password. There are a number of criteria which can be used to differentiate different group PAKE protocols.

Number of protocol rounds. Ideally, this does not increase with the number of parties involved and is as small as possible.

Computation per party. Again this is ideally minimised, but in practice it increases to some extent with the number of parties.

Security model. Some proofs assume idealised primitives, such as in the ideal cipher model. Often there is a need for a common reference string or a public key infrastructure.

While the security of group PAKE protocols should certainly include protection against offline password attacks, it may be questioned whether insider attacks are meaningful when the parties are identified only by possession of the password. Thus it does not seem relevant for the adversary to aim to masquerade as different parties. However, *contributiveness* is one kind of insider property which can be captured – this property requires that the adversary cannot unreasonably influence the value of the shared secret. Group PAKE protocols with contributiveness have indeed been designed [13, 14].

In the remainder of this section we sketch some of the prominent work on group PAKE protocols. At the time of writing, it seems fair to say that optimal solutions are not yet known, and typically there is some compromise with regard to at least one of the criteria listed above. We divide the work into concrete constructions and generic constructions, the latter using some kind of protocol compiler.

8.7.1 Concrete Protocol Constructions

The first formally analysed group PAKE protocol in the shared-password setting was proposed by Bresson *et al.* [150]. However, this required as many rounds as there are parties which is a problem for efficiency. The protocol has a proof in the ideal cipher model.

Abdalla *et al.* [12] provided the first protocol requiring only a constant number of rounds, the number of rounds being four in their case. Computation is almost constant in the number of users, consisting mainly of four multi-exponentiations per user. However, the security proof still requires the ideal cipher model and relies on the DDH assumption. Protocol 8.33 shows the protocol when executed between parties $U_1, \ldots, U_m$ with identities $ID_1, \ldots, ID_m$.

Protocol 8.33 is based on the Burmester–Desmedt generalised Diffie–Hellman protocol (Protocol 9.9) and the computational requirements are dominated by those of the embedded Burmester–Desmedt protocol. Zheng *et al.* [778] designed a very

Information shared between U_i for $1 \leq i \leq m$: Password π. Hash functions H_1, H_2, H_3.

U_{i-1}	U_i	U_{i+1}
Phase 1		
	Choose random nonce N_i and broadcast (U_i, N_i)	
Phase 2		
	$S = ID_1, N_1, \ldots, ID_m, N_m$	
	$k_i = H_1(S, i, \pi)$	
	$r_i \in_R \mathbb{Z}_q$	
$\xleftarrow{\{t_i\}_{k_i}}$	$t_i = g^{r_i}$	$\xrightarrow{\{t_i\}_{k_i}}$
	Decrypt t_{i-1} and t_{i+1}	
	$X_i = (t_{i+1}/t_{i-1})^{r_i}$	
	$Z_{i-1,i} = t_{i-1}^{r_i}$	
Phase 3		
	Broadcast X_i	
	$\mathbf{Z} = (Z_{i-1,i})^m X_i^{m-1} X_{i+1}^{m-2} \ldots X_{i-2}$	
Phase 4		
	Compute $A_i = H_2(S, \{t_1\}_{k_1}, X_1, \ldots, \{t_m\}_{k_m}, X_m, \mathbf{Z}, i)$	
	Broadcast A_i	
	Check all A_j values	
	$\mathbf{K} = H_3(S, \{t_1\}_{k_1}, X_1, A_1, \ldots, \{t_m\}_{k_m}, X_m, A_m, \mathbf{Z})$	

Protocol 8.33: Abdalla–Bresson–Chevassut–Pointcheval password-based group key exchange

similar protocol by replacing the Burmester–Desmedt protocol with the variant protocol of Horng [363] and thereby achieved an improvement in computational efficiency.

Abdalla and Pointcheval [19] provided a construction in the standard model (i.e. with no idealised primitives) using smooth projective hashing (see Sect. 8.3.8). This protocol requires five rounds and, although theoretically speaking it is efficient, it is still fairly expensive computationally; in particular, it requires each party to verify signatures from all other parties. Contemporaneously, Bohli *et al.* [122] designed a related protocol which is more efficient; in particular, it requires only three rounds.

Xu *et al.* [746] designed two group PAKE protocols which are very efficient in terms of rounds, but pay for this in computational efficiency. Specifically, they designed a one-round protocol with a common reference string and a two-round proto-

col without any trust (or set-up) assumptions. Unfortunately, both of these protocols require the usage of indistinguishability obfuscation, which is not efficiently obtainable at the time of writing.

8.7.2 Generic Constructions

A number of authors have proposed compilers to achieve shared-password group PAKE, starting either from a generic two-party PAKE protocol or from a group key exchange protocol with public keys.

From 2-PAKE to Group PAKE

Abdalla *et al.* [11] gave a protocol construction with a security proof which does not assume ideal primitives but requires a common reference string. Their compiler takes in any 2-PAKE protocol and adds two rounds using a construction inspired by the Burmester–Desmedt protocol. Abdalla *et al.* [14] later gave a protocol construction with stronger security properties, particularly universal composability and contributiveness. However, this newer construction adds four rounds to the underlying 2-PAKE protocol.

Hao *et al.* [348] designed two group PAKE protocols using a generic but informal method of extending two-party PAKE protocols which they called the *fairy-ring dance*. Generally, their idea is that each party runs the two-party PAKE protocol with every other party and thereby obtains a shared key which can be used to authenticate other messages. In particular, a pairwise MAC key is used to authenticate a parallel run of the Burmester–Desmedt group exchange protocol (Protocol 9.9), where each party authenticates its contribution separately to each other party. All parties can then compute the shared secret exactly as in the Burmester–Desmedt protocol.

Because the above construction allows all instances of the 2-PAKE protocol as well as the Burmester–Desmedt protocol to run in parallel, the number of rounds is not increased beyond that of the 2-PAKE protocol (although it must be at least two to allow the Burmester–Desmedt protocol to complete).

Hao *et al.* [348] applied their generic construction to achieve two concrete protocols. One is based on SPEKE (Protocol 8.5) and requires only two rounds, while the second protocol is based on J-PAKE (Protocol 8.8) and uses three rounds. However, despite this attractive round complexity, the need to run the two-party PAKE protocol between all pairs of parties results in high computational cost. Hao *et al.* [348] reported on simulations which showed that the protocol was still practical for groups of modest size. No formal security analysis was provided.

From Group AKE to Group PAKE

An alternative to starting from a two-party PAKE and increasing the number of parties is to start from a group key exchange protocol and replace the long-term keys

with the shared password. Since many group key exchange protocols use the long-term keys only for message authentication, the approach can be to replace signatures with password-based authentication.

Li *et al.* [485] described such a compiler which adds four rounds to any group key exchange protocol that is secure against passive adversaries. However, their proof does require both ideal ciphers and random oracles. Wei *et al.* [733] used a similar idea but removed the need for ideal primitives and also improved the result in terms of computation. Their compiler adds only two rounds to the underlying passively secure group key exchange protocol.

8.8 Conclusion

Password-based protocols allow users to establish a strong shared session key with other principals using no secret other than a short string that can be committed to human memory. Considering their more stringent requirements, it may be expected that such protocols will be harder to design than authentication and key establishment protocols with full-length keys. However, understanding of password-based protocols has advanced rapidly since the early 1990s, when it was first realised that they were possible at all. Today there are many protocols available that have security assurances and practical performance similar to what can be achieved for protocols using full-length keys.

In the basic two-party (or client–server) case it seems unlikely that we will find more efficient protocols than those already known, barring any radical developments. One such radical development may be the introduction of quantum computers, and it is noticeable that most concrete protocols rely on some form of Diffie–Hellman assumption. Thus we can expect post-quantum PAKE protocols to be a topic of emerging interest. In the three-party PAKE and group PAKE scenarios, things look less settled. There may still be opportunities for improvements, particularly when stronger security models are considered such as those protecting against ephemeral-key leakage. If it is desired to achieve security proofs that avoid idealisations such as random oracles, then there is also a chance that improvements may arise; although efficient protocols exist with standard-model proofs, they do not usually match the efficiency of protocol with proofs in idealised models.

9

Group Key Establishment

9.1 Introduction

As electronic communications and information services become more sophisticated, many applications involving multiple entities become necessary. Since these applications will generally require secure communications it is necessary to design protocols that establish keys for groups of principals. There is a great variety of different practical requirements that may be appropriate in different applications, and the number of protocols is very large. In this chapter we will mainly restrict attention to ways in which the two-party protocols that have been explored in previous chapters can be generalised to the multi-party situation.

Just as with two-party key establishment, different types of protocol are appropriate depending on the application. Applications such as video and audio conferencing may have very different requirements from other applications such as satellite TV or Internet video broadcasting. Hardjono and Tsudik [349] discussed the following four factors that influence the requirements for such protocols.

Application type. A fundamental feature is how many of the parties must be able to send information. In a corporate teleconference all parties may wish to transmit. In a satellite broadcast to a large group there is only one sender. There may be many intermediate cases too.

Group size and dynamics. For small groups it is feasible for all principals to take part in interactive key establishment; for very large groups it becomes impractical. Some protocols can easily accommodate addition and deletion of members or subsets, while for others there may be a significant computation or communication cost.

Scalability. The efficiency of protocols may vary as the size of the group of users changes.

Trust model. It is important to define which principals are trusted to generate and authenticate keys.

Similar properties, as well as a number of typical application scenarios, were considered by Canetti *et al.* [177]. Detailed study of these issues is outside the scope of

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_9

this book. However, in certain cases we are able to make some comments regarding these matters.

The rest of this section discusses the generalisations of the efficiency and security goals of two-party key establishment that are relevant for multi-party key establishment. In the next section many generalisations of Diffie–Hellman key agreement are described and analysed. Section 9.3 describes protocols that provide authenticated group key agreement by adding authentication to the generalisations of Diffie–Hellman key agreement. Section 9.4 looks at identity-based multi-party protocols while Sect. 9.5 is concerned with some proposals for group key agreement that do not use Diffie–Hellman as a basis. We then look at multi-party key transport protocols in Sect. 9.6, including the important idea known as *logical key hierarchy*.

9.1.1 Efficiency in Group Key Establishment

Efficiency in group key establishment protocols can be measured in the same way as for two-party protocols, taking into account communications, computation and storage requirements (see Sect. 1.5.12). However, some aspects take more prominence in the group setting.

Number of rounds. Recall that one round (Definition 25) contains messages that can be communicated simultaneously. The number of rounds is often deemed important in the two-party case, but in the multi-party case a particularly significant property is whether the number of protocol rounds is independent of the number of principals. This is not the case for all protocols.

Efficiency for different principals. Two-party protocols typically impose the same efficiency demands on both of the principals involved, even if they use a third-party server which has different demands. Group key establishment protocols have a variety of different stuctures. Sometimes there is one principal which takes on a special role, sometimes called a *group controller*, which takes on a higher load. Sometimes there is a hierarchy of principal roles, each of which has a different load. This diversity can complicate comparison of different protocol efficiencies.

Broadcast messages. Often some protocol messages need to be sent to all of the protocol principals. In some communications environments, such as wireless communications, broadcast of messages to all involved parties may be no more costly than sending to a single party. Therefore when comparing group key establishment protocols, broadcast and point-to-point messages are sometimes differentiated.

9.1.2 Generalised Security Goals

Before discussing some of the different types of group key establishment protocols we first consider how the basic definitions for key establishment can be extended to multi-party protocols. The informal definition of a good shared key given in Chap. 2 can be generalised to a good group key as follows.

Definition 36. *The shared session key is a* good key *for principal U_i to use with the set of principals* **U** *only if U_i has assurance that:*

- *the key is fresh (*key freshness*);*
- *the key is known only to principals in* **U** *(*key authentication*).*

If the group is large it may be too expensive, in terms of both communications and computation, for each principal to receive and verify authentication information from all other group members. Instead many protocols simply allow each group member to implicitly authenticate the key with respect to only one other group member. In a multi-party key transport protocol it is natural for the principal distributing the key to provide key authentication. In a multi-party key agreement protocol group members communicate with each other in some systematic way. For example, members are often arranged in a logical ring and each principal's key input can be authenticated to the 'next' member. This may be regarded as a weaker form of authentication since each principal is relying on every other group member to check the authenticity for the whole group.

Generalisations of enhanced protocol properties can be tricky. We consider in particular the important property of *key confirmation*. The potential problems are illustrated by considering the definition given in the *Handbook of Applied Cryptography* [550] which was discussed in Chap. 2.

> *Key confirmation* is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key.

For a two-party protocol this definition is useful: if A knows that the key is good for use with B, and that a different entity has the key, then A knows that B must have the key. But in a group it may be of quite limited use to know only that some other member of the set of principals has the key.

Saeednia and Safavi-Naini [642] suggested that every principal should be sure that either the same key is shared with *all* other principals or that no two principals share the same key. In particular they considered that a situation in which a session key is established by a subset of the intended set of principals, but which is not known to other members of the set, is a major threat. In contrast Ateniese *et al.* [41, 42] noted that key confirmation for all users requires 'at the very least, one round of simultaneous broadcasts', implying that it may be too costly to justify.

Instead of making a judgement on whether or not group key establishment protocols ought to provide key confirmation we simply generalise our definition from Chap. 2 so that we can examine which protocols provide the property.

Definition 37. *Let* **U** *be a set of principals with $U_i, U_j \in \mathbf{U}$.* Key confirmation *of U_j to U_i is provided if U_i has assurance that a key K is a good key to communicate with every principal in* **U***, and principal U_j has received K.* Complete key confirmation *is provided to U_i if key confirmation of U_j is provided to U_i for all $U_j \in \mathbf{U} \setminus \{U_i\}$.*

A possible additional security goal for group key establishment is that of reusing keying material to derive independent keys for subsets of the original group of parties. Manulis [516] considered the problem of extracting *pairwise keys* by reusing the ephemeral keys in group key exchange protocols. Later, together with Abdalla *et al.* [15], this idea was extended to allow keys to be derived for *any* subgroup. The potential advantage of this approach, compared with running a separate protocol, is that both computation and communication can be saved. Indeed in the solution of Abdalla *et al.* [15], independent pairwise keys can be derived without further interaction. They provided security models combining the aims of securing the full group protocol and securing the subgroup keys.

9.1.3 Static and Dynamic Groups

After a multi-party key has been established between a group of principals it may be desired either to add principals to the group or to remove principals from it. Of course this may be done by simply restarting the protocol to establish a new key for the modified group. However, this may be inconvenient, especially if the group is large or the full protocol is computationally expensive. Therefore many group key establishment protocols have been designed, or optimised, in order to allow members to be easily added or removed.

Notice that this issue does not apply to key establishment protocols between two principals and so we have a new way to classify protocols as catering for either *static groups*, in which the group members are fixed, or *dynamic groups*, in which the group members can change. Protocols for dynamic groups include sub-protocols for joining and removing of principals. They may also include sub-protocols for *merging* two groups together or *partitioning* a group into subgroups.

New security goals may also be required for dynamic groups. Kim *et al.* [431, 432] defined three new properties that may be applicable.

Forward secrecy. An adversary who knows a set of group keys cannot derive any subsequent group key. (This is an unfortunate conflict with the more usual meaning of the term forward secrecy.)

Backward secrecy. An adversary who knows a set of group keys cannot find any earlier group key.

Key independence. An adversary who knows any set of group keys cannot find any other group key.

The motivation for forward secrecy (in this sense) is that any group member who leaves the group should not be able to learn any new group keys after leaving. Similarly, backward secrecy ensures that new group members cannot learn old group keys. Key independence is the strongest of the three properties, and implies the other two.

In this chapter we will concentrate mainly on static groups, generalising the protocols in earlier chapters. However, we will mention extensions to dynamic group protocols in many cases.

9.1.4 Insider Attacks

In the analysis of either two-party or multi-party key establishment, we usually assume that the adversary has the ability to corrupt and then masquerade as a legitimate party. Simply by running the normal protocol, the adversary could obtain the session key in any run in which it was involved. This shows that we cannot expect to defend against attacks on the session key from such a powerful adversary. However, other attacks may be defendable. Attacks against key confirmation, which we looked at above, form one kind of example. A number of other insider attacks have been identified as relevant.

Mutual Authentication

Katz and Shin [415] were the first to formalise the notion of insider security for group key establishment. They observed that protocols secure in the sense of key indistinguishability in the usual group setting can still be insecure with regard to two specific attacks. They called these *agreement*, which is essentially the same as key confirmation, and *impersonation*, in which the adversary attempts to confuse legitimate users about the identities of the protocol participants. Katz and Shin performed their analysis in the universal composability model and defined an ideal functionality capturing insider attacks. They defined a *compiler* which takes any protocol which is secure against attacks on the session key and turns it into one secure according to their functionality. The compiler adds messages from each party to sign an acknowledgement value related to the agreed key. Bresson and Manulis [156] later defined a stronger notion of insider security by allowing the adversary to obtain ephemeral secrets. Their security analysis is in the more popular game-based setting.

Key Compromise Impersonation

While forward secrecy has long been accepted as a valuable property for group key establishment, it was not until relatively recently that the related property of resistance to key compromise impersonation was studied. KCI occurs when the adversary obtains the long-term key of Alice and then misleads her regarding the identities of the other group members involved in a protocol run.

One reason for the delayed interest is that is seems harder to find scenarios in which KCI attacks may be relevant. Gorantla *et al.* [330] suggested a situation where a server, whose long-term key is compromised, will have difficulty to identify an intruder if that intruder masquerades as different group members each time the intruder authenticates. In any case, applying the principle that we should always give the adversary as much power as possible we should allow the possibility of such an attack and defend against it if possible.

Contributiveness

We discussed the notion of *key control* for two-party key exchange in Sect. 5.1.2. Naturally such properties can be considered in group key establishment protocols

too. Pieprzyk and Wang [615] showed that many well-known key establishment protocols do not restrict key control. Later Bresson and Manulis [155] formalised *contributiveness* as a property which prevents key control by insider adversaries. While they argued that earlier models do not capture contributiveness, they provided a compiler based on adding signatures similar to that of Katz and Shin.

Robustness

A multi-party protocol is arguably more vulnerable than a two-party protocol to disruption from participants who aim to prevent others from completing the protocol successfully. Guaranteeing protection from such attacks cannot be achieved in our typical models in which the adversary completely controls the network and can simply block or alter all messages so that no subset of participants ever completes the protocol successfully. Desmedt *et al.* [245] assumed the notion of a *reliable broadcast channel* in order to design a protocol providing robustness against insider adversaries. The reliable broadcast channel guarantees that messages are sent correctly to all protocol principals. Even with such an assumption, robust protocols seem to require much more complexity than protocols without such a property. The protocol of Klein *et al.* described in Sect. 9.3.2 is an example of a protocol designed to be robust, but we do not focus on such protocols in this chapter.

9.1.5 Notation

The notation used in this chapter is shown in Table 9.1. Because many group key establishment protocols are based on generalisations of Diffie–Hellman key exchange there is much similarity with the notation used in Chap. 5.

The Diffie–Hellman type protocols run in some suitable group $\mathbb{G}$. For the purpose of describing protocols we usually assume that $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$, but often $\mathbb{G}$ can be more general, for example an elliptic curve group. We differentiate between the set of all principals $\mathbf{U}$ who may participate in different protcol runs, and the set of principals P who take part in a specific protocol run. In formal models P is often called the *partner identifier* set and is a variable recorded by each user.

9.2 Generalising Diffie–Hellman Key Agreement

There are different ways that the basic Diffie–Hellman two-party key agreement can be generalised to a multi-party protocol. In this section we will look at several of the natural ways to do this without considering how to use them to provide authenticated key agreement. In Sect. 9.3 we explore different ways that these protocols can be incorporated into group protocols to provide establishment of good keys.

Table 9.1: Notation for group key establishment protocols

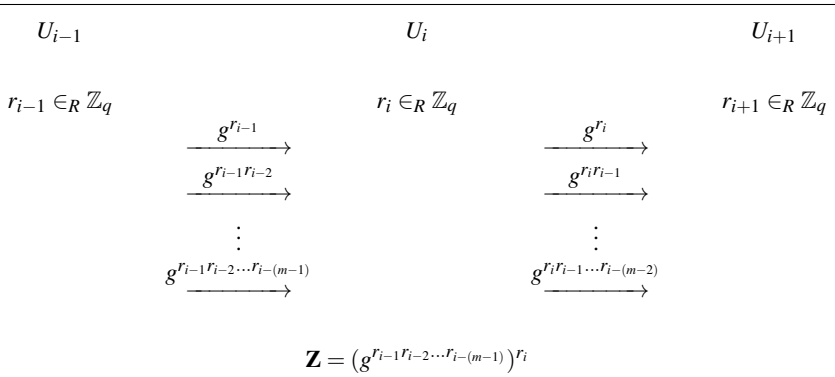
p	A large prime (usually at least 1024 bits).
q	A prime with $q \mid p-1$.
$\mathbb{G}$	The (algebraic) group in which the protocol runs. Common examples are a subgroup of $\mathbb{Z}_p^*$ or an elliptic curve group. $\mathbb{G}$ is often of order q.
g	A generator of $\mathbb{G}$.
$\mathbf{U}$	The set of all principals that may run the protocol.
$\mathbf{P}$	The set of principals in the group intended to share the session key.
m	The size (cardinality) of $\mathbf{P}$.
U_i	The i'th principal in $\mathbf{P}$, where $1 \leq i \leq m$.
$\mathbf{P}_i$	The set of principals with which U_i intends to share the session key.
r_i	Random integer, typically of the same size as the order of $\mathbb{G}$, chosen by U_i.
t_i	The value g^{r_i}. All computations take place in $\mathbb{Z}_p$.
x_i	The private long-term keys of U_i.
y_i	The public key of U_i, the value g^{x_i}. These public keys will have to be certified in some standard way which we do not consider here.
$\mathbf{Z}$	The shared secret calculated by the principals.
$\mathbf{K}$	The shared session key.
$H(.)$	A one-way hash function. Certain protocols may require specific properties and may specify particular functions.

9.2.1 Ingemarsson–Tang–Wong Key Agreement

Ingemarsson *et al.* [374] considered a model in which principals are connected in a ring, so that principal U_i receives messages only from U_{i-1} and sends messages only to U_{i+1}. To enable a general description we allow any index i but U_i and U_j are the same principal when $i \equiv j \bmod m$. Ingemarsson *et al.* described a number of Diffie–Hellman generalisations based on the idea of *symmetric functions*.

Definition 38. *The j'th symmetric function on the set $S = \{r_1, \ldots, r_m\}$ is denoted $S^{(j)}$ and consists of the sum of all possible products of j distinct elements of S.*

For example, if $m = 3$ then $S^{(1)} = r_1 + r_2 + r_3$, $S^{(2)} = r_1 r_2 + r_1 r_3 + r_2 r_3$ and $S^{(3)} = r_1 r_2 r_3$. Ingemarsson *et al.* showed that protocols exist in which the shared key is $g^{S^{(j)}}$ for any j with $1 \leq j \leq m$. However, they pointed out that all these protocols are insecure against a passive adversary who can tap the communications channels between the principals, except for the case $j = m$, which also turns out to be the case that minimises the computations required for each principal. (More precisely they considered how many of the communications channels need to be tapped in order for the eavesdropper to find the shared key in each case.)



Protocol 9.1: Ingemarsson–Tang–Wong generalised Diffie–Hellman protocol

Protocol 9.1 shows the message flows between principals U_{i-1}, U_i and U_{i+1} for the case $j = m$. In the first round, principal U_1 sends g^{r_1} to U_2, principal U_2 sends g^{r_2} to U_3, and so on. In the second round principal U_1 sends $g^{r_1 r_m}$ to U_2, U_2 sends $g^{r_1 r_2}$ to U_3, and so on. At the end of $m-1$ rounds, principal U_i is able to calculate the shared secret $\mathbf{Z} = g^{r_1 r_2 \cdots r_m}$ by raising the final received message to its exponent r_i.

Of interest in all such protocols is the amount of communication and computation required for each principal, as well as the number of rounds required. In this case each principal must calculate m exponentiations, and send and receive $m-1$ messages. As explained above, $m-1$ rounds are required.

9.2.2 Steiner–Tsudik–Waidner Key Agreement

Steiner *et al.* [692] proposed three protocols, named GDH.1, GDH.2 and GDH.3, which can be regarded as variations of Protocol 9.1. Indeed, in each of their protocols the same key as that of Protocol 9.1 is derived by the principals from the same set of messages. The differences between all these protocols lie in where the computation is done and which messages are communicated. This leads to a flexible set of protocols that can be adapted to a variety of applications depending on the priorities in optimising communications or computational requirements.

The two phases of GDH.1 are shown in Protocol 9.2; messages travel in opposite directions along the line in the two phases. During the first phase values are collected up by the principals. Principal U_1 initially sends g^{r_1} to U_2. Then U_2 raises the received value to its secret r_2 and adds this to the received message to form the message sent to U_3. A similar pattern is followed by every other principal. At the end of the first phase principal U_m is able to calculate the shared secret $\mathbf{Z} = g^{r_1 r_2 \cdots r_m}$. In the second phase principal U_m starts sending messages in the opposite direction. The message sent by U_m to U_{m-1} is $g^{r_m}, g^{r_1 r_m}, g^{r_1 r_2 r_m}, \ldots, g^{r_1 r_2 \cdots r_{m-2} r_m}$. Principal U_{m-1} now starts the general pattern by using the last part of the message to form $\mathbf{Z}$ by raising it to its

own random value r_{m-1}, removing it from the message, then raising the remaining $m-2$ parts of the message to its secret r_{m-1} before sending it to U_{m-2}.

U_{i-1}	U_i	U_{i+1}
Phase 1:		
$\xrightarrow{g^{r_1}, g^{r_1 r_2}, \ldots, g^{r_1 r_2 \ldots r_{i-1}}}$	$r_i \in_R \mathbb{Z}_q$	
		$\xrightarrow{g^{r_1}, g^{r_1 r_2}, \ldots, g^{r_1 r_2 \ldots r_i}}$
Phase 2:		
		$h_{i+1} = g^{r_{i+1} r_{i+2} \ldots r_m}$, $h_{i+1}^{r_1}, h_{i+1}^{r_1 r_2}, \ldots, h_{i+1}^{r_1 r_2 \ldots r_{i-1}}$ $\xleftarrow{\quad}$
$h_i = g^{r_i r_{i+1} \ldots r_m}$, $h_i^{r_1}, h_i^{r_1 r_2}, \ldots, h_i^{r_1 r_2 \ldots r_{i-2}}$ $\xleftarrow{\quad}$	$\mathbf{Z} = (h_{i+1}^{r_1 r_2 \ldots r_{i-1}})^{r_i}$	

Protocol 9.2: Steiner–Tsudik–Waidner GDH.1 protocol

In comparison with Protocol 9.1, GDH.1 reduces the average amount of computation required per principal. At the same time it takes twice the number of rounds; in GDH.1 no message can be sent until the previous message has been received, whereas in Protocol 9.1 many of the messages can be sent in parallel. In the next version of their protocol,[1] GHD.2, Steiner *et al.* reduced the number of rounds required by gathering together more partial calculations in the first phase and replacing the second phase with a single broadcast message from principal U_m.

Protocol 9.3 shows the messages in the GDH.2 protocol. In the first phase U_i receives i values from U_{i-1}; one of these values is the *principal value* p_{i-1}, while the remaining $i-1$ values consist of p_{i-1} with one of the exponents $r_1, r_2, \ldots, r_{i-1}$ 'missing'. Initially U_1 starts Phase 1 by sending g^{r_1} and g to U_2. The notation used in Protocol 9.3 is not intended to indicate how these values are calculated: the inverted exponents are written only to conveniently summarise the values sent. On receiving its message, U_i raises all received values to its exponent r_i to form i new message components and also includes the principal value p_{i-1} in the message sent on to U_{i+1}.

The second phase of GDH.2 consists of a single message broadcast by U_m, which includes all the partial calculations necessary for every other U_i to find $\mathbf{Z}$ with a single exponentiation using r_i. On receiving the final message in the first phase, U_m can calculate the shared secret from the principal value p_{m-1} as $\mathbf{Z} = p_{m-1}^{r_m} = g^{r_1 r_2 \cdots r_m}$. The final broadcast message can be calculated by U_m by raising each of the other $m-1$ components of its received message to its secret exponent r_m.

[1] This protocol is also known as IKA.1 in later papers of these authors [693].

U_{i-1}	U_i	U_{i+1}
Phase 1:		
$p_{i-1} = g^{r_1 r_2 \cdots r_{i-1}}$, $p_{i-1}^{r_1^{-1}}, p_{i-1}^{r_2^{-1}}, \ldots, p_{i-1}^{r_{i-1}^{-1}} \longrightarrow$		
	$r_i \in_R \mathbb{Z}_q$	
		$p_i = g^{r_1 r_2 \cdots r_i}$, $p_i^{r_1^{-1}}, p_i^{r_2^{-1}}, \ldots, p_i^{r_i^{-1}} \longrightarrow$
Phase 2:		
U_m calculates $\mathbf{Z} = p_{m-1}^{r_m}$		
U_m broadcasts $\mathbf{Z}^{r_1^{-1}}, \mathbf{Z}^{r_2^{-1}}, \ldots, \mathbf{Z}^{r_m^{-1}}$		
U_i calculates $\mathbf{Z} = (\mathbf{Z}^{r_i^{-1}})^{r_i}$		

Protocol 9.3: Steiner–Tsudik–Waidner GDH.2 protocol

Steiner *et al.* proposed another variant protocol designed to minimise the average computation required for each principal. This protocol[2], GDH.3, has four phases.

Phase 1. Partial information is generated by the first $m-1$ principals.

Phase 2. Principal U_{m-1} broadcasts $g^{r_1 r_2 \cdots r_{m-1}} = \mathbf{Z}^{r_m^{-1}}$.

Phase 3. Each of the principals $U_1, U_2, \ldots, U_{m-1}$ 'removes' its exponent from the broadcast information and sends the result to principal U_m to add the final exponent to these partial values.

Phase 4. Principal U_m applies its exponent r_m to all the received partial calculations and broadcasts the results. This allows each principal to find $\mathbf{Z}$ by applying its exponent to the correct partial value.

Protocol 9.4 summarises the message flows for GDH.3. Again, the notation in Protocol 9.4 is not intended to imply that any inversion of exponents is actually calculated. In Sect. 9.2.9 below the relative features of each of the GDH variants are compared, together with other generalised Diffie–Hellman protocols.

Protocols GDH.2 and GDH.3 are well suited for dynamic groups. Steiner *et al.* provided explicit protocols for adding and deleting group members after the initial key has been agreed. The idea is to reuse most of the keying material from the initial protocol run, but one principal must renew its random value. The introduction of a fresh random exponent makes the new and old keys independent so that any added principal cannot find the initial key and a removed principal cannot find the new key. Steiner *et al.* also provided a protocol for many principals to join the group together.

In the original paper [692] the protocols for addition and deletion of group members required principal U_m to choose a new random input and update the previous

[2] Also known as IKA.2 [693].

Phase 1. (for $i < m-1$)

U_{i-1} $\xrightarrow{g^{r_1 r_2 \ldots r_{i-1}}}$ U_i $\xrightarrow{g^{r_1 r_2 \ldots r_i}}$ U_{i+1}

$r_i \in_R \mathbb{Z}_q$

Phase 2.

U_{m-1} broadcasts $\mathbf{Z}' = g^{r_1 r_2 \cdots r_{m-1}}$

Phase 3.

U_i $\xrightarrow{(\mathbf{Z}')^{r_i^{-1}}}$ U_m

Phase 4.

U_m calculates $\mathbf{Z} = (\mathbf{Z}')^{r_m}$

U_m broadcasts $\mathbf{Z}^{r_1^{-1}}, \mathbf{Z}^{r_2^{-1}}, \ldots, \mathbf{Z}^{r_{m-1}^{-1}}$

U_i calculates $\mathbf{Z} = (\mathbf{Z}^{r_i^{-1}})^{r_i}$

Protocol 9.4: Steiner–Tsudik–Waidner GDH.3 protocol

values, but later Steiner *et al.* [693] pointed out that any principal can act as a *group controller* responsible for this. The protocols for adding and deleting principals are essentially identical for both GDH.2 and GDH.3. In order to add a member the group controller must refresh the message just before the final broadcast message using a new random exponent. Then the added principal sends a new broadcast message and all other principals calculate the new key in the same way as in the final phase of the protocols. In the protocol for removal of a member the group controller broadcasts a new message for the final phase which excludes the component intended for the member to be removed.

9.2.3 Steer–Strawczynski–Diffie–Wiener Key Agreement

The generalised Diffie–Hellman key agreement protocol proposed by Steer *et al.* [688] was designed for use in a secure audio teleconference. In their description the messages between the principals are broadcast to all principals. However, not all messages are required by all principals and in our description principals are arranged in a linear fashion. Following the pattern of Protocol 9.2 there are two phases in which messages flow in opposite directions. The shared secret $\mathbf{Z}$ has the following value, but is calculated in a different way by each principal.

$$\mathbf{Z} = g^{r_m (g^{r_{m-1} (g^{\cdots (g^{r_1 r_2})})})}.$$

U_{i-1}	U_i	U_{i+1}
Phase 1.		
	$r_i \in_R \mathbb{Z}_q$	$\xleftarrow{t_{i+1}, t_{i+2}, \ldots, t_m}$
$\xleftarrow{t_i, t_{i+1}, \ldots, t_m}$	$t_i = g^{r_i}$	
Phase 2.		
$\xrightarrow{w_{i-1}}$		
	$v_i = w_{i-1}^{r_i}$	
	$w_i = g^{v_i}$	
	$\mathbf{Z} = t_m^{(t_{m-1}^{(\ldots^{(t_{i+1}^{v_i})})})}$	$\xrightarrow{w_i}$

Protocol 9.5: Steer–Strawczynski–Diffie–Wiener generalised Diffie–Hellman protocol

Protocol 9.5 shows the messages flowing to and from a typical principal U_i together with the computations made by U_i. Principal U_1 is able to calculate $\mathbf{Z}$ immediately once Phase 1 is complete as follows:

$$\mathbf{Z} = t_m^{(t_{m-1}^{(\ldots^{(t_2^{r_1})})})}.$$

To start Phase 2 principal U_1 sets $w_1 = t_1$ and sends this to U_2 who can now calculate $\mathbf{Z}$. Notice that the principals at the right hand end of the sequence use fewer computations than those at the left hand end. Principals U_1 and U_2 both use m exponentiations, but the number of exponentiations required decreases by one as i increases by one. Principal U_m requires only two exponentiations: one in Phase 1 and one in Phase 2 to find $\mathbf{Z} = v_{m-1}^{r_m}$. As compared with Protocol 9.1 the increase in the average number of computations is accompanied by a large decrease in the number of messages that are sent. In Protocol 9.5 each principal sends and receives only two messages (except for the end points which send and receive only one).

Computation of $\mathbf{Z}$ in Protocol 9.5 uses elements of $\mathbb{G}$ as exponents, but such exponents should be in $\mathbb{Z}_q$. A mapping from $\mathbb{G}$ to $\mathbb{Z}_q$ is easily defined if $\mathbb{G} = \mathbb{Z}_p^*$, but in general there is no efficient mapping known. Therefore implementation of Protocol 9.5 in elliptic curve groups is not staightforward.

9.2.4 Kim–Perrig–Tsudik Tree Diffie–Hellman

Perrig [611] originally designed a key agreement protocol in which principals are considered as leaves in a binary tree structure. Later, together with Kim and Tsudik [431, 432] he considered this form of tree-based key agreement in much more detail. In particular they showed that this protocol is very suitable when the

composition of the group is changed without restarting the whole protocol. They show that protocols designed to add or remove either single principals or groups can be performed efficiently by restructuring the tree of keys.

In the basic protocol a secret is agreed between each pair of sibling nodes of depth j. This secret is known to all principals at leaves that are children of those nodes. This continues until a secret is agreed between the two nodes that are the children of the root node, and this key is the group shared secret. In comparison with the previous protocols we have looked at, an advantage of the tree-based protocol is that the number of rounds required is only logarithmic in the number of principals.

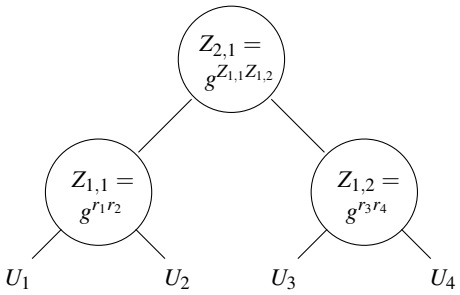


Fig. 9.1: Kim–Perrig–Tsudik tree Diffie–Hellman protocol with four principals

Figure 9.1 illustrates the protocol for the case of four principals. We assume that there are $m = 2^d$ principals involved. The description below works equally well in general with $d = \lceil \log_2 m \rceil$ and letting $r_i = 1$ for $m < i \leq 2^d$. We denote by $Z_{j,k}$ the k'th key shared during round j. A mapping from $\mathbb{G}$ to $\mathbb{Z}_q$ is required in order to compute $Z_{2,1}$ (see discussion in Sect. 9.2.8).

In general there are d rounds as shown in Protocol 9.6. During the first round each pair of sibling leaves shares a standard Diffie–Hellman secret. Conceptually we place each shared secret at the node of depth $d-1$ that is the parent node of the two leaves (principals) that share it. During round 2 one of the principals from the pair that share $Z_{1,k}$ calculates and broadcasts $g^{Z_{1,k}}$; both principals also receive $g^{Z_{1,k'}}$ where $Z_{1,k'}$ is the sibling node of $Z_{1,k}$ and thus both can calculate $Z_{2,k''}$ at the parent node of $Z_{1,k}$ and $Z_{1,k'}$. During round i the two sets of principals who share keys at sibling nodes of depth $d-i+1$ generate a Diffie–Hellman key by using these keys as the input to the Diffie–Hellman protocol.

At the end of round d the shared secret $\mathbf{Z} = Z_{d,1}$ is calculated by each of the principals as the root of the tree. Each principal can calculate the key for every node between its leaf and the root of the tree. In round i one principal must be nominated to broadcast g^K, for each key K at depth $d-i$, to ensure that all principals can calculate the key for the next round. We may regard this large number of broadcast messages as the price to be paid for the reduced number of rounds.

Round 1

$$Z_{1,1} = g^{r_1 r_2}$$
$$Z_{1,2} = g^{r_3 r_4}$$
$$\vdots$$
$$Z_{1,2^{d-1}} = g^{r_{m-1} r_m}$$

Round 2

$$Z_{2,1} = g^{Z_{1,1} Z_{1,2}}$$
$$Z_{2,2} = g^{Z_{1,3} Z_{1,4}}$$
$$\vdots$$
$$Z_{2,2^{d-2}} = g^{Z_{(1,2^{d-1}-1)} Z_{(1,2^{d-1})}}$$

Round d

$$Z_{d,1} = g^{Z_{(d-1,1)} Z_{(d-1,2)}}$$

Protocol 9.6: Kim–Perrig–Tsudik tree Diffie–Hellman protocol

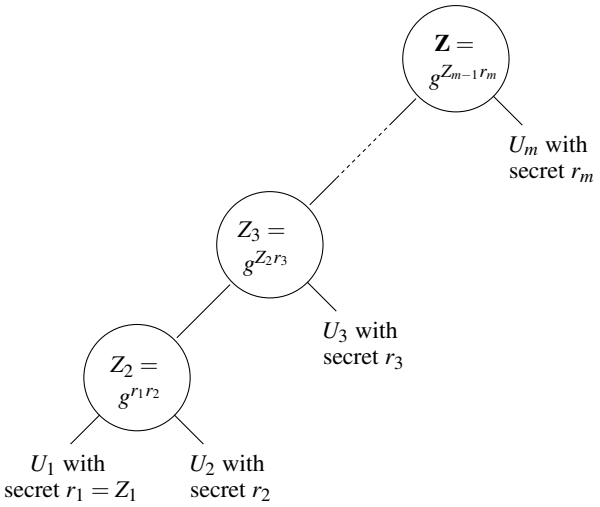
Bresson and Manulis [156] proposed a variant of Protocol 9.6 in which the tree, rather than being balanced as in Fig. 9.1, is maximally *un*balanced. That is, each internal node has one child node which is a leaf. The advantage of this change is that the protocol can be completed in two rounds. A possible disadvantage is that all principals have varying computational requirements; indeed the average computation per principal is higher than when using the balanced binary tree. Protocol 9.7 shows how this works.

In the first round each principal U_i chooses a Diffie–Hellman ephemeral value r_i and broadcasts the public key $t_i = g^{r_i}$. In the second round U_1 at a node with maximal depth computes all values $Z_2, Z_3, \ldots$ up to the root of the tree $Z_m = \mathbf{Z}$. With knowledge of r_1, U_1 computes $Z_2 = (t_2)^{r_1}$ and $Z_j = (t_j)^{Z_{j-1}}$ for $j > 2$. Then U_1 broadcasts the public keys $g^{Z_2}, g^{Z_3}, \ldots, g^{Z_{m-1}}$ corresponding to the internal nodes. Now every other principal U_i, $i > 1$, can compute the root of the tree starting from $Z_i = (g^{Z_{i-1}})^{r_i}$ (where we write Z_1 for r_1). The root of the tree, $\mathbf{Z}$, is the shared secret.

9.2.5 Becker and Wille's Octopus Protocol

Becker and Wille [66] proposed their 'Octopus' protocol in order to provide an example that minimises the number of simultaneous message exchanges between principals. An exchange between two principals can include a pair of messages, one in either direction. We will discuss this matter further in Sect. 9.2.9. A building block for the Octopus protocol is the four-party key agreement shown as Protocol 9.8. The resulting shared secret is the same as in Perrig's four-party protocol in Fig. 9.1.

However, the number of exchanges required is reduced (to four) due to the manner in which the key is derived. Firstly U_1 and U_2 exchange Diffie–Hellman messages



Protocol 9.7: Bresson–Manulis tree Diffie–Hellman protocol

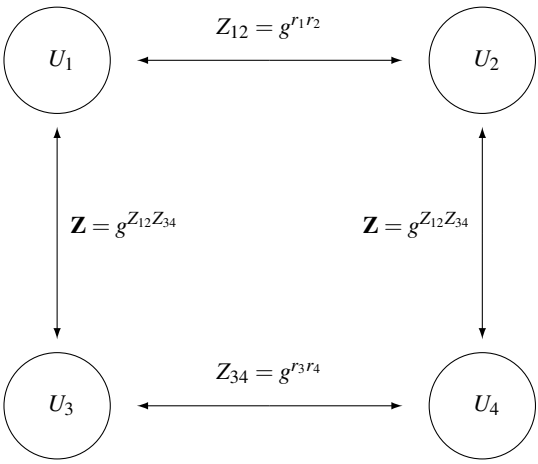
to obtain the shared key Z_{12}, while U_3 and U_4 similarly obtain Z_{34}. Then U_1 and U_3 use these keys as inputs to a further Diffie–Hellman exchange so that both can obtain the shared secret $g^{Z_{12}Z_{34}}$, while U_2 and U_4 do the same. As with Protocols 9.5, 9.6 and 9.7, a mapping is needed to take Z_{12} and Z_{34}, elements of $\mathbb{G}$, to elements of $\mathbb{Z}_q$. Becker and Wille described such a mapping, but we omit it from our descriptions.

To set up the Octopus protocol the principals are partitioned into four subgroups of as near equal size as possible (this is a four-legged Octopus, but Becker and Wille also proposed a version with 2^d legs). Each group has a distinguished member, the *subgroup controller*, who manages the protocol for that subgroup. There are three phases in the Octopus protocol.

Phase 1. Each subgroup controller runs a Diffie–Hellman exchange individually with each of its group members. Let us denote the secret shared with the j'th member of the i'th subgroup as $Z_{i,j}$.

Phase 2. The four subgroup controllers run the basic protocol (Protocol 9.8) with each controller using as input the product of all the keys agreed with members of its subgroup: $r_i = \prod_j Z_{i,j}$. Thus controller U_1 first agrees $Z_{12} = g^{\prod Z_{1,j} \prod Z_{2,j}}$ with controller U_2. Then U_1 sends $g^{Z_{12}}$ to controller U_3 and receives $g^{Z_{34}}$ from U_3. This process allows each subgroup controller to calculate the group shared secret:

$$\mathbf{Z} = g^{(g^{\prod Z_{1,j} \prod Z_{2,j}})(g^{\prod Z_{3,j} \prod Z_{4,j}})}. \tag{9.1}$$



Protocol 9.8: Basic Octopus protocol with four principals

Phase 3. Each subgroup controller sends each member of its subgroup two values, which consist of its two inputs to the last step in Phase 2 except that the secret it shared with that principal during Phase 1 is missing from the exponent. For example, suppose subgroup controller U_1 shared secret $Z_{1,k}$ with one of its subgroup members in Phase 1. Then U_1 sends to that principal the two values $g^{\prod_{j \neq k} Z_{1,j} \prod Z_{2,j}}$ and $g^{(g^{\prod Z_{3,j} \prod Z_{4,j}})}$. From these values the principal can first reconstruct $g^{\prod Z_{1,j} \prod Z_{2,j}}$ using $Z_{1,k}$ and then the group shared secret $\mathbf{Z}$ shown in Eqn. 9.1.

The number of message exchanges required in the Octopus protocol is $m-4$ during each of Phases 1 and 3, and four during Phase 2, making a total of $2m-4$. The exchanges in Phases 1 and 2 include two messages each, while those in Phase 3 consist of only one message. Thus in total there are $3m-4$ messages sent.

9.2.6 Burmester–Desmedt Key Agreement

The protocol of Burmester and Desmedt [168] uses an ingenious method to obtain efficiency both in the number of messages sent per user and in the amount of computation required. Instead of using a symmetric function to define the shared key, as in Protocol 9.1, Burmester–Desmedt key agreement uses a *cyclic* function. Protocol principals are arranged in a ring so that $U_1 = U_{m+1}$. The protocol is simplest to understand in the version that allows broadcast communications; below we also describe how it can be implemented using communication only between adjacent users in the ring. There are two phases in this broadcast version as shown in Protocol 9.9.

U_{i-1}	U_i	U_{i+1}
Phase 1:		
	$r_i \in_R \mathbb{Z}_q$	
$\xleftarrow{t_i}$	$t_i = g^{r_i}$	$\xrightarrow{t_i}$
	$X_i = (t_{i+1}/t_{i-1})^{r_i}$	
	$Z_{i-1,i} = t_{i-1}^{r_i}$	
Phase 2:		
	Principal U_i broadcasts X_i to all other parties.	
	U_i calculates $\mathbf{Z} = (Z_{i-1,i})^m X_i^{m-1} X_{i+1}^{m-2} \ldots X_{i-2}$.	

Protocol 9.9: Burmester–Desmedt generalised Diffie–Hellman protocol with broadcasts

During Phase 1 each adjacent pair of users, U_i and U_{i+1}, performs a basic Diffie–Hellman key exchange. However, instead of calculating the individual secrets, principal U_i calculates the ratio, X_i, of its two secrets with adjacent principals. In Phase 2 each principal broadcasts its X_i value. Once all principals have broadcast their values, every principal can calculate the shared secret. The following calculation shows that all principals calculate the same shared secret if all principals act correctly.

$$\begin{aligned}
\mathbf{Z} &= (Z_{i-1,i})^m \cdot X_i^{m-1} \cdot X_{i+1}^{m-2} \cdot \ldots \cdot X_{i-2} \\
&= (Z_{i-1,i})^m \left(\frac{Z_{i,i+1}}{Z_{i-1,i}}\right)^{m-1} \left(\frac{Z_{i+1,i+2}}{Z_{i,i+1}}\right)^{m-2} \ldots \left(\frac{Z_{i-2,i-1}}{Z_{i-3,i-2}}\right) \\
&= Z_{i-1,i} \cdot Z_{i,i+1} \cdot Z_{i+1,i+2} \cdot \ldots \cdot Z_{i-2,i-1} \\
&= g^{r_1 r_2 + r_2 r_3 + \ldots + r_m r_1}.
\end{aligned}$$

Katz and Yung [419] gave a security proof for Protocol 9.9 in a model where the adversary is passive. The adversary is able to observe protocol runs, and to reveal non-target session keys and corrupt parties not involved in the target session. Their security proof relies on the difficulty of the decision Diffie–Hellman problem. Katz and Yung used this as the main example for their compiler to construct an authenticated group key agreement protocol (see Sect. 9.3.5). Burmester and Desmedt [170] provided their own security proof in the same model as Katz and Yung defined, but assuming different variants of the Diffie–Hellman problem.

Just and Vaudenay [407] have pointed out that the Burmester–Desmedt protocol can be generalised in two ways. Firstly, the keys shared between the adjacent principals, the $Z_{i,i+1}$ values, can be found using any key agreement protocol. Secondly, there are alternative ways to combine these shared keys. One example is to choose $X_i = Z_{i,i+1} - Z_{i-1,i}$ and then define the shared secret as follows.

$$\begin{aligned}\mathbf{Z} &= mZ_{i-1,i} + (m-1)X_i + (m-2)X_{i+1} + \ldots + X_{i-2} \\ &= Z_{1,2} + Z_{2,3} + \ldots + Z_{m,1}.\end{aligned}$$

Horng [363] designed a protocol closely related to Protocol 9.9 but with the principals arranged in a logical line rather than in a ring. The shared secret is also subtly different from that of Protocol 9.9, including only the adjacent exponent pairs on the line: $\mathbf{Z} = g^{r_1 r_2 + r_2 r_3 + \ldots + r_{m-1} r_m}$. The method for calculating the shared secret depends on where the principal stands in the line, but it is more efficient than Protocol 9.9, especially if the number of principals is large. Horng showed that any passive attack on the protocol that can distinguish $\mathbf{Z}$ from a random string is as hard as the decision Diffie–Hellman problem.

Since broadcast messages can be expensive in some communications networks, Burmester and Desmedt also proposed a protocol version that uses only communication between adjacent principals. The basic idea of the protocol is unchanged, but the calculations are distributed between different principals, reducing the computation required for each principal. There are three phases in this pairwise version as shown in Protocol 9.10.

U_{i-1}	U_i	U_{i+1}
Phase 1:		
	$r_i \in_R \mathbb{Z}_q$	
$\xleftarrow{t_i}$	$t_i = g^{r_i}$	$\xrightarrow{t_i}$
	$X_i = (t_{i+1}/t_{i-1})^{r_i}$	
	$Z_{i-1,i} = t_{i-1}^{r_i}$	
Phase 2:		
$\xrightarrow{b_{i-1}, c_{i-1}}$	$b_i = X_i b_{i-1}$	
	$c_i = b_{i-1} c_{i-1}$	$\xrightarrow{b_i, c_i}$
Phase 3:		
$\xrightarrow{d_{i-1}}$	$d_i = d_{i-1}/X_i^m$	
	$\mathbf{Z} = d_{i-1} \cdot Z_{i-1,i}^m$	$\xrightarrow{d_i}$

Protocol 9.10: Burmester–Desmedt pairwise generalised Diffie–Hellman protocol

Phase 1 in Protocol 9.10 is the same as in the broadcast version of the protocol described above. After this phase each principal knows the secrets shared with each adjacent principal. The purpose of Phase 2 is to distribute the computation of the shared secret amongst all principals. This phase requires m rounds starting from U_1 with $b_0 = c_0 = 1$. Alternatively, any one principal could gather all the X_i values and do all the computation for this phase on behalf of the other principals.

When Phase 2 is complete, U_1 has received the value $c_m = X_1^{m-1} \cdot X_2^{m-2} \cdot \ldots \cdot X_{m-1}$ from U_m and sets this value to d_0. In the final phase each principal can calculate the shared secret $\mathbf{Z}$ and also calculates the d_i value to be sent on to the next principal. For example, U_1 calculates $\mathbf{Z} = d_0 \cdot Z_{0,1}^m$ and sends $d_1 = d_0 / X_1^m$ on to U_2. It follows from the equality $X_1 = X_2 \cdot X_3 \cdot \ldots \cdot X_m$ that $d_1 = X_2^{m-1} \cdot X_3^{m-2} \cdot \ldots \cdot X_m$ so that U_2 can calculate $\mathbf{Z}$ in a similar way. Notice that all the messages passed between the principals can be calculated from the public X_i values, which are available to an eavesdropper in the broadcast protocol version.

9.2.7 One-Round Tripartite and Multi-Party Diffie–Hellman

All the Diffie–Hellman generalisations described so far require at least two rounds to complete. Later we will see that there are group key agreement protocols that can be run in one round, but they do not achieve forward secrecy. The protocol of Joux [402] is the only example currently known of a group key agreement protocol that can be run in a single round and still provide forward secrecy; however, the protocol can only work with three parties.

Joux's protocol works in elliptic curve groups and exploits properties known as *pairings* of group points. An overview of elliptic curve pairings is given in Sect. 7.1.2.

If g generates a suitable pairing group then the three protocol parties, A, B and C, each choose a random exponent and broadcast g^{x_A}, g^{x_B} and g^{x_C} to the other parties. Using the bilinear property of pairings, it is possible for any of the principals to calculate the shared secret $\mathbf{Z} = g^{x_A x_B x_C}$ because:

$$\hat{e}(g^{x_B}, g^{x_C})^{x_A} = \hat{e}(g^{x_A}, g^{x_C})^{x_B} = \hat{e}(g^{x_A}, g^{x_B})^{x_C} = \hat{e}(g, g)^{x_A x_B x_C}.$$

The security of Joux's protocol is based on the difficulty of the decision bilinear Diffie–Hellman problem. We examine proposals for authenticated versions in Sect. 9.3.7.

Boneh and Silverberg [125] observed that the existence of cryptographically strong *multilinear* maps, extending the bilinear maps provided by pairings, would allow one-round key agreement for any number of users. Recently some candidates for such maps have been proposed in the literature [291] but at the time of writing it seems fair to say that the security of such constructions is not settled [201].

9.2.8 Security of Generalised Diffie–Hellman

Because we have considered only protocols without authentication in this section, none of these protocols can provide security against an active adversary. However, for many of the protocols in this section it is possible to relate their security to that of two-party Diffie–Hellman key exchange. Since the Diffie–Hellman problem has resisted all challenges for over 35 years this is as good a security property as we can hope for.

Definition 39. *Suppose that n protocol principals each choose respective random inputs* $r_1, r_2, \ldots, r_n$ *and derive the shared secret* $\mathbf{Z} = g^{r_1 r_2 \cdots r_n}$. *The n-party Diffie–Hellman problem is to find* $\mathbf{Z}$ *given all elements of the set* $\{g^{\Pi(S)} | S \subset \{r_1, r_2, \ldots, r_n\}\}$.

Steiner *et al.* [692] gave a formal proof that the n-party decision Diffie–Hellman problem problem is hard if the original two-party version is hard. This gives assurance that if the adversary is able to distinguish between a random value and the shared secret, given the public values, then the adversary can also solve the two-party DDH problem. However, as emphasised later by Steiner [689, page 56], the tightness of that reduction between these two problems decreases exponentially with n. Bresson *et al.* [151] gave a tighter proof for the special cases relevant for their proofs, discussed in Sect. 9.3.3.

Proofs of security seem more difficult when the shared secret is defined through 'recursive' use of Diffie–Hellman, because the output does not lie in the same algebraic group as the input. However, if we work in a group $\mathbb{G}$ which is a special subgroup of $\mathbb{Z}_p^*$ then there exists an efficient bijection from the group $\mathbb{G}$ generated by g to $\mathbb{Z}_q$; this mapping is invoked when moving the outputs of a Diffie–Hellman protocol in $\mathbb{G}$ to the inputs in $\mathbb{Z}_q$.

Kim *et al.* [431, 432] considered the *tree group Diffie–Hellman* (TGDH) problem in a such a group $\mathbb{G}$. They showed that the decisional TGDH problem is no more than a polynomial factor harder to break than the DDH problem. Bresson and Manulis [156] performed a related but more general anlaysis. Similarly Becker and Wille [66], in their description of the Octopus protocol (Protocol 9.8), prove that if ordinary Diffie–Hellman is secure then so is the Octopus protocol.

9.2.9 Efficiency of Generalised Diffie–Hellman

When comparing the relative efficiencies of the protocols in this section there are a number of different measures that may be relevant depending on the implementation and application scenario. Table 9.2 summarises the most important performance features of each of the protocols. The purpose of this table is only to be indicative of the comparative costs of each protocol and therefore we have taken licence in some of the figures where the exact values differ only slightly from those indicated. For example, in GDH.1 the end principals are required to send only one message, while in GDH.2, U_m sends only the broadcast message. Similarly the number of exponentiations for U_m in GDH.1 and GDH.2 is m.

In Table 9.2 the number of messages refers to the number of point-to-point messages sent, while the total number of broadcast messages is in addition to the point-to-point messages. Any message sent to more than one principal is counted as a broadcast. A number of different papers also include tables of comparison [31, 611, 692] but the reader should be aware that these do not all count items in the same way. Amir *et al.* [31] have reported on extensive practical tests of the performance of several of these protocols.

Choosing a suitable protocol for an application may depend on many factors and it may not be sufficient to optimise any one particular performance parameter.

Table 9.2: Computational requirements in generalised Diffie–Hellman protocols

	Exponentiations per U_i	Messages per U_i	Messages in total	Broadcasts in total	Rounds
ITW (9.1)	m	$m-1$	$m(m-1)$	0	$m-1$
GDH.1 (9.2)	$i+1$	2	$2(m-1)$	0	$2(m-1)$
GDH.2 (9.3)	$i+1$	1	$m-1$	1	m
GDH.3 (9.4)	$3^{\dagger}$ (m for U_m)	2	$2m-3$	2	$m+1$
SSDW (9.5)	$m-i+2$	2	$2(m-1)$	0	$2(m-1)$
KPT (9.6)	$\lceil \log_2 m \rceil + 1^{\ddagger}$	1	m	$m-2$	$\lceil \log_2 m \rceil$
BM (9.7)	$m+1-i$	$1^{\ddagger}$	$m+1$	$m+1$	2
Octopus (9.8)	$4^{\ddagger}$	3	$3m-4$	0	4
BDB (9.9)	$3^{\star}$	2	$2m$	m	2
BD (9.10)	$3^{\star}$	4	$4m-1$	0	$2m$

† Calculation of an inverse also required.
‡ Minimum value. Some principals must do more.
* Average of two exponentiations with exponent m also required.

There is no single generalised Diffie–Hellman protocol that is the best for all the criteria used in Table 9.2. If the lowest computation per principal is desired then the Burmester–Desmedt (BD) or GDH.3 protocols are attractive. (Note, however, that one principal has a high computational load in GDH.3.) If the number of rounds is to be minimised then the Burmester–Desmedt broadcast (BDB) protocol looks attractive but the communications network used must support simultaneous broadcasts if the figure of only two rounds is to be achieved. If broadcast communications are to be avoided then GDH.1 or BD are both worth considering, although ITW may be the best if the number of rounds should be minimised at the same time.

Becker and Wille [66] have provided lower bounds on several of the desirable performance parameters. These bounds were derived from an analysis using basic counting and inductive proofs. Their results are summarised in Table 9.3, which is set out to emphasise the clear division between protocols with and without broadcast, as made by Becker and Wille. As well as the number of messages used and the number of (synchronous) rounds, Becker and Wille also considered the number of *exchanges* (connections between pairs of users) and the number of *simple rounds* (in which every party sends and receives at most one message). Notice that synchronous rounds are only relevant when broadcast messages are used.

Comparison between Tables 9.2 and 9.3 shows that in many cases the bounds derived by Becker and Wille can be met by practical protocols. Specifically GDH.1 and GDH.2 meet the bounds on the minimum number of messages for the two cases without and with broadcast messages. In the case of GDH.2, adding the single broadcast message to the $m-1$ point-to-point messages gives the bound of m messages

Table 9.3: Performance bounds for group key agreement (Becker and Wille)

	Messages	Exchanges	Simple rounds	Synchronous rounds
Without broadcasts	$2(m-1)$	$2(m-2)$	$\lceil \log_2 m \rceil$	–
With broadcasts	m	m	$\lceil \log_2 m \rceil$	1

indicated in Table 9.3. When it comes to (simple) rounds, we have seen that Perrig's protocol meets this bound. The Octopus protocol was specifically designed to be optimal in terms of number of exchanges and meets the lower bound of $2m-4$. Although Becker and Wille left it as an open question whether a protocol with only one synchronous round is possible in the case without broadcasts, we will show in Sect. 9.5 examples of protocols that meet this bound.

In the next section we will examine how the protocols introduced in this section can be enhanced by adding authentication to the messages sent. We will also see that there are other features that may prove important in comparing the suitability of protocols for different applications.

9.3 Group Key Agreement Protocols

When examining two-party key agreement based on Diffie–Hellman in Chap. 5, two classes of protocol were evident: those in which the long-term public keys were included in the calculation of the shared secret and those in which long-term keys were used only for message authentication. For general group key agreement the former class of protocols is completely missing. The reason for this is that there seems to be no symmetrical way that long-term keying information from multiple users can be incorporated into the shared key. Therefore all the published authenticated group key agreement protocols make use of either public key encryption or digital signatures to provide authentication of the messages sent in the various generalised Diffie–Hellman key agreement protocols.

9.3.1 Authenticating Generalised Diffie–Hellman

The designers of many of the multi-party Diffie–Hellman generalisations that were examined in Sect. 9.2 have ignored the issue of key authentication, or have simply stated that authentication using digital signatures may be added. When there are no long-term keys providing authentication, forward secrecy is meaningless. If key authentication is added by, for example, signing protocol messages with a long-term key, then forward secrecy will usually be provided automatically as long as the signature mechanism and ephemeral keys are independent.

Burmester and Desmedt proposed that their generalised Diffie–Hellman key exchange (Protocol 9.10) could provide key authentication if each t_i value was authenticated by any chosen means. They suggested that this means could be any good digital

signature scheme but also provided an explicit interactive mechanism. However, Just and Vaudenay [407] pointed out that authenticating the messages is not sufficient, since it does not show that the party authenticating knows the random input r_i and consequently unknown key-share attacks are possible. In any case, this simple authentication can only provide a weak form of key authentication in that each party authenticates the participation of only one other party. There is no direct assurance of which other parties may have the shared secret.

Just and Vaudenay [407] also proposed a generalised form of the Burmester–Desmedt protocol, using their two-party key agreement protocol examined in Chap. 5 (Protocol 5.11) as the basic building block. They prove the security of their protocol against a passive adversary. Although this protocol avoids the unknown key-share attack against the Burmester–Desmedt protocol it still only provides weak key authentication. Just and Vaudenay suggested that such problems may be avoided by making each party broadcast a signed hash of all messages sent in the protocol. This seems to be an expensive way to provide implicit key authentication.

9.3.2 Klein–Otten–Beth Protocol

Klein *et al.* [433] proposed a protocol that is related to the protocol of Ingemarsson *et al.* (Protocol 9.1) but with two distinct enhancements designed to provide authentication and robustness.

- Each message is protected by a digital signature of the sender, which also includes a unique identifier for the protocol run.
- Each intermediate value is calculated multiple times with the aim of detecting and recovering from errors caused by principals deviating from the protocols.

More precisely there are $m-1$ rounds to the protocol during which messages are broadcast, via a write-only bulletin board, to all other principals. Messages all consist of triples of the form $(U_i, \mathbf{P}, \mathsf{Sig}_{U_i}(\mathtt{PID}, g^A))$, where PID is an identifier, $\mathbf{P}$ is a set of users in $\mathbf{U}$, and A is the set of exponents input by the users in $\mathbf{P}$. For example, a particular message sent by U_2 might be $(U_2, \{U_3, U_2\}, \mathsf{Sig}_{U_2}(\mathtt{PID}, g^{r_3 r_2}))$.

Round 1. U_i chooses random ephemeral input x_i and broadcasts the triple

$$(U_i, \{U_i\}, \mathsf{Sig}_{U_i}(\mathtt{PID}, g^{x_i})).$$

Round j $(2 \leq j \leq m-1)$**.** The following steps are taken.

1. U_i collects all messages from round $j-1$ that exclude x_i in the exponents. For each of these U_i raises the g^A value to the exponent x_i, and forms the new message triple (adding its identity to the set $\mathbf{P}$) and broadcasts the result.
2. Each message with the same user set $\mathbf{P}$ is then compared. If there are any differences then the recovery process is invoked.
3. Once these are resolved, all duplicates are deleted and j is incremented.

The recovery protocol has as input a set of principals **P** and a set of message triples using set **P** but with differing values of g^A. All principals belonging to **P** are required to reveal their secret input x_i which allows checking against the signed inputs from the previous round. Principals found to have cheated by a majority of the other principals are expelled from the group. Those principals who have not cheated choose new x_i values and reconstruct their inputs for the current round.

Despite the attraction of having a protocol robust to disruption, this is an expensive protocol. Even without any recovery, each principal is required to perform a total of $2 + \binom{m-1}{1} + \binom{m-1}{2} + \ldots + \binom{m-1}{m-2} = 2^{m-1}$ exponentiations, which becomes impractical for large values of m.

9.3.3 Authenticated GDH Protocols

Ateniese *et al.* [41, 42] proposed two methods to extend their GDH.2 protocol (Protocol 9.3) to provide authenticated group key agreement. They remarked that the same extensions can be applied to GDH.3 too, but implied that GDH.1 is less interesting in practice, since the alternatives have smaller computation and communications requirements (see Table 9.2).

The simplest method of providing authentication changes only the final broadcast message of Protocol 9.3. In order to achieve this extension it is assumed that the distinguished principal U_m shares a secret K_i with each U_i. It was suggested by Ateniese *et al.* that this secret should be calculated from the static Diffie–Hellman key, $S_{i,m} = g^{x_i x_m}$, shared between U_i and U_m; for example, $K_i = F(S_{i,m})$ where F is a function taking elements of $\mathbb{G}$ to $\mathbb{Z}_q$. Then the first phase of the protocol is the same as Protocol 9.3. In the second phase U_m sends $\mathbf{Z}^{r_i^{-1} K_i}$ to U_i. On receipt of this message U_i can find the shared secret $\mathbf{Z} = g^{r_1 r_2 \ldots r_m}$ by raising the received message to the power $r_i K_i^{-1}$. The message flows are shown in Protocol 9.11 which Ateniese *et al.* called A-GDH.2.

The use of the shared secret K_i values means that principals can be sure that the value they calculate for **Z** will be known only to those that actually participate in the protocol with U_m. This is on the assumption that U_m follows the protocol faithfully. In this sense the protocol provides implicit key authentication. However, in common with many multi-party protocols, principals have no direct assurance of which other principals are participating in the protocol. This means that the principals really know which other parties have the shared secret only if the group is fixed or assured by some other means. Ateniese *et al.* claimed a proof of security against passive attacks but not against an active adversary. Indeed they pointed out that an attack similar to Burmester's triangle attack is possible, although they claimed that the unusual circumstances involved make such attacks 'very unlikely in practice'.

Pereira and Quisquater [609] conducted an analysis of the security of Protocol 9.11 using a systematic technique for deciding which powers of g can be obtained by an active adversary. They showed that strong key authentication can indeed fail if the group membership varies. They demonstrated an explicit attack in which the adversary takes part in one protocol run and can find the key in a subsequent run in which the adversary is not intended to be involved. To see how this attack can work

Information shared by U_i and U_m: Key K_i.

Phase 1.

U_{i-1} $\qquad$ U_i $\qquad$ U_{i+1}

$p_{i-1} = g^{r_1 r_2 \ldots r_{i-1}}$,
$p_{i-1}^{r_1^{-1}}, p_{i-1}^{r_2^{-1}}, \ldots, p_{i-1}^{r_{i-1}^{-1}}$ $\longrightarrow$

$r_i \in_R \mathbb{Z}_q$

$p_i = g^{r_1 r_2 \ldots r_i}$,
$p_i^{r_1^{-1}}, p_i^{r_2^{-1}}, \ldots, p_i^{r_i^{-1}}$ $\longrightarrow$

Phase 2.

U_m broadcasts $\mathbf{Z}^{r_1^{-1} K_1}, \mathbf{Z}^{r_2^{-1} K_2}, \ldots, \mathbf{Z}^{r_m^{-1} K_m}$
U_i calculates $\mathbf{Z} = (\mathbf{Z}^{r_i^{-1} K_i})^{r_i K_i^{-1}}$

Protocol 9.11: Ateniese–Steiner–Tsudik A-GDH.2 protocol

consider the particular case of A-GDH.2 when $m = 4$ shown in Protocol 9.12. The attack proceeds as follows.

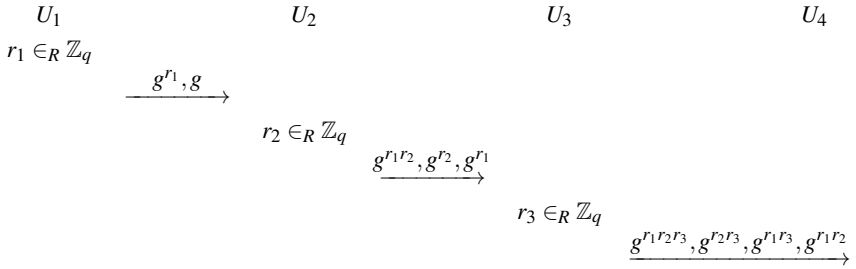
1. The adversary takes part in Protocol 9.12 as U_3 but alters the message sent from U_3 to U_4 by replacing $g^{r_1 r_3}$ with $g^{r_1 r_2}$. As a consequence, U_4 will include $g^{r_1 r_2 r_4 K_2}$ in the broadcast part intended for U_2, instead of the correct value $g^{r_1 r_3 r_4 K_2}$. The adversary also obtains $g^{r_1 r_2 r_4}$ from the last broadcast part, since it knows K_3. The adversary records the values exchanged in this run.
2. The adversary observes a new protocol run involving only the other three principals from the first run. In the new run suppose that new values r_1', r_2' and r_4' are chosen by U_1, U_2 and U_4. The adversary replaces the message from U_1 by $g^{r_1 r_2 r_4}$ obtained from the first run. Then U_2 will send $g^{r_1 r_2 r_4 r_2'}$ as part of its message to U_4. Finally the adversary can replace the part of the broadcast message to U_2 with $g^{r_1 r_2 r_4 K_2}$ recorded from the first run. This means that U_2 will calculate $\mathbf{Z} = g^{r_1 r_2 r_4 r_2'}$ which was sent by U_2 in the second message and so is known to the adversary.

Pereira and Quisquater [609] also found a more convincing attack than that of Ateniese *et al.* in which old keying material is replayed by an active adversary. This attack could be prevented in the case that a one-way key derivation function is used and only the derived session key becomes compromised.

Because the long-term keying material (the K_i values) is used only for the authentication, it follows that forward secrecy from a passive adversary is provided in Protocol 9.11, as long as the multi-party Diffie–Hellman assumption holds. However, Pereira and Quisquater [609] also demonstrated that an active adversary can alter the

Information shared by U_j and U_4: Key K_j.

Phase 1.



Phase 2.

U_4 chooses $r_4 \in_R \mathbb{Z}_q$ and broadcasts $g^{r_2 r_3 r_4 K_1}, g^{r_1 r_3 r_4 K_2}, g^{r_1 r_2 r_4 K_3}$.
U_i calculates $\mathbf{Z} = g^{r_1 r_2 r_3 r_4}$.

Protocol 9.12: Protocol 9.11 when $m = 4$

protocol in such a way that all but one of the principals computes the same key but forward secrecy will subsequently fail.

If principal U_i's long-term key becomes compromised then K_i is also compromised. In this situation the adversary can forge the messages intended for U_i in both phases and hence can find the key that U_i calculates as $\mathbf{Z}$. Therefore resistance to key compromise impersonation is not provided. Key confirmation is not provided either since U_i obtains no assurance that any other party has obtained the same key. However, Ateniese *et al.* suggested that U_m may include $g^{H(\mathbf{Z})}$ in the broadcast message. This gives each U_i key confirmation with respect to U_m only.

In comparison with the unauthenticated GDH.2 protocol, Protocol 9.11 adds only a small overhead. The computations required for U_m and for each U_i remain essentially unchanged except for the K_i values. If the protocol is run frequently the K_i values may already be available.

In order to reduce the reliance on principal U_m, Ateniese *et al.* designed an alternative method to provide authentication in GDH.2, resulting in a protocol that they called SA-GDH.2. The general idea is to allow each principal to mutually authenticate every other principal through use of a long-term shared secret. Each pair of principals U_i and U_j shares two keys $K_{i,j}$ and $K_{j,i}$. Ateniese *et al.* did not define how these are calculated, but an obvious way is to use two different derivations of the long-term static Diffie–Hellman key $S_{i,j}$. Protocol 9.13 shows the message flows.

As in Protocols 9.3 and 9.11 there are two phases. During the first phase each principal sends and receives a message with m ordered fields (for U_1 the fields $V_{0,1}, V_{0,2}, \ldots, V_{0,m}$ should all be taken to equal g). U_i raises the j'th received field to the power $r_i K_{j,i}$ with the exception that the i'th field is left unchanged. The up-

Information shared by U_i and U_j: Keys $K_{i,j}$ and $K_{j,i}$.

Phase 1:

U_{i-1} $\qquad$ U_i $\qquad$ U_{i+1}

$V_{i-1,1}, V_{i-1,2}, \ldots, V_{i-1,m} \longrightarrow$

$r_i \in_R \mathbb{Z}_q$

$V_{i-1,1}^{r_i K_{i,1}}, V_{i-1,2}^{r_i K_{i,2}}, \ldots, V_{i-1,i}, \ldots, V_{i-1,m}^{r_i K_{i,m}} \longrightarrow$

Phase 2:

U_m broadcasts $V_{m,i} = \mathbf{Z}^{r_i^{-1} K_{1,i} \ldots K_{i-1,i} K_{i+1,i} \ldots K_{m,i}}$ for each i

U_i calculates $\mathbf{Z} = V_{m,i}^{r_i K_{1,i}^{-1} \ldots K_{i-1,i}^{-1} K_{i+1,i}^{-1} \ldots K_{m,i}^{-1}}$

Protocol 9.13: Ateniese–Steiner–Tsudik SA-GDH.2 protocol

dated fields are then sent on to the next principal. On completion of the first phase, U_m is able to calculate the shared secret $\mathbf{Z} = g^{r_1 r_2 \cdots r_m}$ from the last field of its received message. This last field will be $g^{r_1 r_2 \cdots r_{m-1} K_{1,m} K_{2,m} \ldots K_{m-1,m}}$ and so U_m needs to remove the $K_{j,m}$ exponents for each $0 \leq j \leq m-1$ and add the r_m exponent.

In the second phase U_m, after performing the same transformations as done by each U_i in Phase 1, broadcasts the results. This allows every other principal to find $\mathbf{Z}$ in the symmetrical way by removing the relevant $K_{i,j}$ exponents and adding the exponent r_i.

The advantage of Protocol 9.13 over Protocol 9.11 is that a stronger form of implicit key authentication is achieved. Each principal U_i knows that only principals that possess one of the shared keys $K_{i,j}$ are able to calculate the same value of $\mathbf{Z}$ calculated by U_i. There is still no confirmation that any other principal does actually possess that key and again U_i must know the other group members' identities by some external means. Furthermore, the computational cost of Protocol 9.13 is greatly increased for each principal, with the exception of U_m: now each principal needs to perform m exponentiations. Ateniese *et al.* [42] provide a more detailed analysis of the communication and computational cost of the A-GDH.2 and SA-GDH.2 protocols.

Bresson *et al.* [154], and later most of the same authors [148, 149], proposed a different authenticated version of GDH.2 and provided proofs in the Bellare–Rogaway model. Their protocol is the same as Protocol 9.3 with two additions.

- The group identity (the names of all principals involved in the protocol) is added to each message flow.
- Each message flow is signed with a long-term key of the sender.

The proof in their first paper [154] reduces the security of the protocol to that of the m-party Diffie–Hellman problem, when the set of exponents is restricted to those exchanged in the protocol. The reduction is not very tight, however. Their next paper [148] extended their results in two ways. Firstly, additional protocols to allow principals to join and leave an existing group were added and proven secure. Secondly, the security reduction was made tighter, at least for a fixed-size group. These first two papers assumed random oracles in the proofs; the latest paper in the series [149] was able to remove this assumption, replacing the computational m-party Diffie–Hellman assumption with a stronger m-party decision problem. The results from all these papers have been combined into one journal paper [153].

9.3.4 Authenticated Tree Diffie–Hellman

Kim *et al.* [431, 432] focused on unauthenticated versions of their tree-based protocol shown in Protocol 9.6. They assumed that all the messages are authenticated using a suitable digital signature scheme and that messages include a protocol message identifier and a timestamp of the sender. All of these features are assumed to be checked by the receiving parties. Kim *et al.* did not consider a formal security model for the authenticated protocol.

Bresson and Manulis [156] specified a slightly different authentication method for their unbalanced Protocol 9.7. They required that each message is signed with a secure digital signature scheme but used principal-chosen nonces instead of timestamps. Protocol 9.14 gives details of the protocol using the notation of Protocol 9.7. Notice that here we use $\mathbf{P}_i$ to denote the set of principals which U_i is expecting to share the session key with, which is not a priori assumed to be the same for all U_i. The values 0, 1 and 2 are constants used to identify each messages position in the protocol.

Protocol 9.14 takes three rounds, the third round allowing each principal to sign a session identifier consisting of all the nonces. Bresson and Manulis also gave a security proof for their authenticated key agreement protocol in a strong model allowing the adversary to obtain ephemeral secrets.

9.3.5 Katz–Yung Compiler

Katz and Yung [418, 419] observed that many group key establishment protocols lack *scalability* because the number of protocol rounds or messages increases as the number of participants increases. They were therefore motivated to find a protocol which can be proven secure in a suitable model but with a constant number of protocol rounds and the minimal number of protocol messages. They therefore settled on the BDB protocol, Protocol 9.9, as a promising candidate but needed to add authentication and provide a proof of security.

Rather than prove the authenticated version of the BDB protocol directly, Katz and Yung proposed a *compiler* to transform a generic secure unauthenticated group key establishment protocol into an authenticated protocol. By applying their compiler to the BDB protocol they then arrived at their goal. Of course this requires

Shared information: Signature verification information for all principals. Public constants V_0, V_1, V_2.

Phase 1.

Each U_i chooses $r_i \in_R \mathbb{Z}_q^*$, nonce $n_i \in_R \{0,1\}^l$ and computes $t_i = g^{r_i}$.

Each U_i broadcasts $n_i, t_i, \mathsf{Sig}_i(0, n_i, t_i, \mathbf{P}_i)$.

All users check the signatures of all received messages.

Phase 2.

U_i sets the session identifier as $SID_i = (n_1, \ldots, n_m)$

U_1 computes the internal node keys $Z_2, Z_3, \ldots, Z_m$ up to the root of the tree.

U_1 computes the set of public keys of internal nodes $\mathbf{Y} = (g^{Z_1}, g^{Z_2}, \ldots, g^{Z_{m-2}})$ U_1 broadcasts $\mathbf{Y}, \mathsf{Sig}_m(1, \mathbf{Y}, \mathsf{sid}_1, \mathbf{P}_1)$

Phase 3.

All users U_i, $1 < i \leq m$, perform the following:

- verify the signatures of all received messages;
- compute the root shared secret $\mathbf{Z}$;
- compute authenticator $\mu_i = \mathsf{MAC}_K(V_1)$ using temporary key K, itself computed iteratively as $K_i = \rho_m$ where $\rho_l = \mathsf{MAC}_{\rho_{l-1} \oplus \pi(n_l)}(V_0)$, $\rho_0 = \mathsf{MAC}_{\mathbf{Z}}(V_0)$ and π is a one-way permutation;
- broadcast $\mathsf{Sig}_i(2, \mu_i, \mathsf{sid}_i, \mathbf{P}_1)$.

Each U_i checks the signatures from all parties in $\mathbf{P}_i$. If all checks pass then U_i computes the session key as $\mathbf{K} = \mathsf{MAC}_{\mathbf{K}_i}(V_2)$.

Protocol 9.14: Bresson and Manulis authenticated tree Diffie–Hellman protocol

proofs that the BDB protocol satisfies the required security property of unauthenticated security, and that the compiler adds the expected security property to make the protocol securely authenticated.

The Katz–Yung compiler takes a protocol π and transforms it to a new protocol π' in a sequence of four steps as shown in Table 9.4. The basic idea is that each party generates a nonce and then subsequent received messages must include the nonce and must be signed. Thus each party in protocol π' has a signature key-pair independent of the long-term keys in π. Since each party has to first send out its own nonce, protocol π' has one more round than protocol π. While Katz and Yung were particularly interested in applying their compiler to the BDB protocol, it could also be used on any other protocol in Sect. 9.2, but security can only be guaranteed if the starting protocol has a security proof as an unauthenticated group key establishment protocol.

Katz and Yung used the security model of Bresson *et al.* [154] to define security of their protocols. The model of Bresson *et al.* is a version of the BR93 model (see

Table 9.4: Katz–Yung compiler [418, 419]

Input: Group key establishment protocol π secure against passive adversaries.
Output: Group key establishment protocol π' secure against active adversaries.

1. Each party generates signature keys independent of the keys for π. Signature verification keys must be available to all protocol parties.
2. Each party U_i generates a nonce r_i and broadcasts it to all others parties in $\mathbf{U}$. After receiving all nonces, each party U_i forms the set of all nonces it received: $N_i = \{(U_1, r_1), (U_2, r_2), \ldots, (U_m, r_m)\}$.
3. Protocol π is now run between all parties with the following changes.
   - When sending a message m from protocol π, party U_i adds its nonce set N_i to message m and then signs the message with its signature key.
   - When receiving a message in π', party U_i checks that the nonces it receives are the same as those in N_i and verifies that the signature is valid from a party in $\mathbf{U}$. If not, then party U_i aborts the protocol. Otherwise U_i proceeds with protocol actions as specified in protocol π.
4. The session key is computed the same way as in π.

Sect. 2.2) extended to the group setting using session identifiers as in the BPR00 model. Although Katz and Yung did not consider a formal definition for mutual authentication, they noted that it is intuitively clear that the signatures from each party can be used to provide such authentication.

The model used by Katz and Yung does not attempt to deal with insider attacks and later Bohli *et al.* [121] pointed out that the authenticated version of the BDB protocol, Protocol 9.9, derived by Katz and Yung using their compiler, does not satisfy the insider security property of *integrity*. Indeed, it is not difficult to see that if two colluding users in Protocol 9.9, say U_1 and U_3, run normally in Phase 1 but swap their values in Phase 2, then two other parties, say U_2 and U_4, will not compute the same session key. However U_2 and U_4 see the same messages so that in the security model this means that they are valid partners. Even though such an attack will not result in leakage of information, it can be considered a serious attack on the availability of information. We emphasise that security against malicious insiders was never claimed by Katz and Yung.

Dutta and Barua [263] adapted the Katz–Yung compiler to use sequence numbers instead of nonces to provide freshness. This allowed them to obtain a two-round authenticated version of the BDB protocol. Like Katz and Yung, they did not consider insider attacks. However, they did include a formal analysis of the dynamic operations of the protocol, defining joining and leaving operations for protocol principals.

As defined in Table 9.4, the compiler requires all parties to verify signatures from all other parties. When applied to unauthenticated protocols where there are messages broadcast to all parties, such as the BDB protocol, this does not add to the overall complexity. However, Desmedt *et al.* [243] observed that there are protocols which require both $O(\log m)$ communication and $O(\log m)$ computation complexity

(see Sect. 9.6.1). To apply the Katz–Yung compiler to such protocols would spoil both the computation and communication complexity. Desmedt *et al.* [243] therefore defined, and proved secure, a variant (actually a generalisation) of the Katz–Yung compiler in which signatures are generated only for the set of parties from whom messages are processed. This allowed them to achieve authenticated versions of the protocols without increasing the computation and communication complexity.

9.3.6 Protocol of Bohli, Gonzalez Vasco and Steinwandt

Bohli *et al.* [121] analysed various insider attacks on group key establishment protocols and proposed a protocol to avoid such attacks. Protocol 9.15 is closely related to an earlier protocol of Kim, Lee and Lee [428]

Shared information: Signature verification information for all principals.

Phase 1.

All users choose $r_i \in_R \mathbb{Z}_q^*$, $k_i \in_R \{0,1\}^l$ and compute $t_i = g^{r_i}$.
For $i \neq m$, U_i broadcasts $k_i, t_i, \mathsf{Sig}_i(k_i, t_i, \mathbf{P}_i)$.
U_m broadcasts $H(k_m), t_m, \mathsf{Sig}_m(H(k_m), t_m, \mathbf{P}_i)$.
All users check the signatures of all received messages.

Phase 2.

U_i calculates $x_i^L = H(t_{i-1}^{r_i})$, $x_i^R = H(t_{i+1}^{r_i})$ and $T_i = x_i^L \oplus x_i^R$.
U_i sets the session identifier as $SID_i = H(\mathbf{P}_i, k_1, \ldots, k_{n-1}, H(k_n))$.
For $i \neq m$, U_i broadcasts $SID_i, T_i, \mathsf{Sig}_i(SID_i, T_i)$.
U_m broadcasts $(k_m \oplus x_m^R), SID_m, T_m, \mathsf{Sig}_m(k_m \oplus x_m^R, SID_m, T_m)$.
All users perform the following checks:

- verify the signatures of all received messages;
- check that $T_1 \oplus \ldots \oplus T_m = 0$;
- check that $SID_i = SID_j$ for all j;
- compute $k'_m = (k_m \oplus x_m^R) \oplus T_{m-1} \oplus \ldots \oplus T_{i+1} \oplus x_i^R$ and check that $H(k'_m) = H(k_m)$ where the second value is that sent during Phase 1 by U_m.

If all checks pass then U_i computes the shared secret as $\mathbf{Z} = H(\mathbf{P}_i, k_1, k_2, \ldots, k'_m)$.

Protocol 9.15: Bohli–Gonzalez Vasco–Steinwandt protocol

Each user chooses an input k_i to the shared secret in addition to the Diffie–Hellman ephemeral value $t_i = g^{r_i}$. There is one distinguished principal U_m and k_m is the only key input which is kept secret. Note that the Diffie–Hellman values are not used in the shared secret calculation, instead they are used to mask the value of

k_m which can be recovered only by those participating. This allows the protocol to achieve forward secrecy. Note that this protocol cannot be achieved by application of the Katz–Yung compiler to any one-round protocol.

Bohli *et al.* [121] proved that Protocol 9.15 satisfies the following three properties, in addition to providing indistinguishability of the session key as usual. Their proof models the hash function H as a random oracle and relies on the difficulty of the computational Diffie–Hellman problem. Their security definition requires that the protocol has a well-defined session identifier.

Strong authentication. For any honest user U_i who has accepted with a certain set of partners $\mathbf{P}_i$, each honest $\mathbf{U}_j \in \mathbf{P}_i$ has an instance with the same session identifier and which includes $\mathbf{U}_i$ in its set of expected partners: $\mathbf{U}_i \in \mathbf{P}_j$.

Contributory. No subset of up to $m-1$ users is able to force the session key into a pre-defined subset

Integrity. All accepting honest principals share the same session identifier and the same set of principals $\mathbf{P}_i$.

Later Gorantla *et al.* [330] described notions of KCI resistance for both outsider and insider adversaries. They proved that Protocol 9.15 satisfies their strongest notion of insider KCI resistance under the same assumptions as used by Bohli *et al.*

Bohli *et al.* [121] noticed that party U_m in Protocol 9.15 can control the shared secret $\mathbf{Z}$ to some extent by waiting to see the k_i values of all other users before choosing and committing to k_m in Phase 1. They therefore mentioned a slight variation in which all principals U_i defer sending their k_i values until Phase 2, instead sending $H(k_i)$ in Phase 1, and then adding k_i to the message broadcast by U_i in Phase 2. Gorantla *et al.* [326] showed that this variant protocol is secure in the universally composable model. They commented that using the UC model naturally addresses insider attacks; moreover, Protocol 9.15 is a two-round protocol which cannot be reached by application of the compiler of Katz and Shin [415] (see Sect. 9.1.4).

An adversary who has access to the ephemeral key, r_i, of any protocol participant U_i can obtain x_i^R and hence k'_m and the session key. Zhao *et al.* [774] therefore proposed another variant of Protocol 9.15 by applying the NAXOS trick to combine the long-term key with the ephemeral key before using it. They then proved security of this variant in a strong model of security where ephemeral key leakage is available to the adversary.

Surprisingly, Gao *et al.* [290] later significantly simplified Protocol 9.15 both in terms of the messages sent and the computations required by each party. Protocol 9.16 shows that the structure is still essentially the same as Protocol 9.15 but the k_i values are no longer needed and the signatures are only used in the second phase.

Gao *et al.* [290] provided proofs that the optimised Protocol 9.15 is still secure in the same model as in the original. Note that the protocol now becomes symmetric, unlike Protocol 9.15 where principal U_m sends messages and performs computations different from other principals. On the theoretical side, Gao *et al.* [290] also pointed out that the security proof can be made to work without assuming that H is a random oracle, although it still requires a common reference string.

Shared information: Signature verification information for all principals.

Phase 1.

All users choose $r_i \in_R \mathbb{Z}_q^*$ and compute $t_i = g^{r_i}$

Each U_i broadcasts t_i

All users check that none of the received values equals the identity element in $\mathbb{G}$.

Phase 2.

U_i calculates $x_i^L = H(t_{i-1}^{r_i})$, $x_i^R = H(t_{i+1}^{r_i})$ and $T_i = x_i^L \oplus x_i^R$

Each U_i broadcasts $T_i, \mathsf{Sig}_i(T_i, t_1, t_2, \ldots, t_m, \mathbf{P}_i)$

All users perform the following checks:

- verify the signatures of all received messages;
- check that $T_1 \oplus \ldots \oplus T_m = 0$;

If all checks pass then U_i recovers each x_j^R value using its own x_i^R and the T_j values and computes the shared secret as

$$\mathbf{Z} = H(x_1^R, x_2^R, \ldots, x_m^R, t_1, t_2, \ldots, t_m, \mathbf{P}_i, 0).$$

Principal U_i also sets the session identifier as $SID_i = H(x_1^R, x_2^R, \ldots, x_m^R, t_1, t_2, \ldots, t_m, \mathbf{P}_i, 1)$.

Protocol 9.16: Optimised Bohli–Gonzalez Vasco–Steinwandt protocol of Gao, Neupane and Steinwandt

9.3.7 Authenticated Tripartite Diffie–Hellman

When wishing to achieve one-round authenticated key agreement protocols it is natural to start from the one-round generalisations of Diffie–Hellman from pairings mentioned in Sect. 9.2.7. Al-Riyami and Paterson [25] considered a number of alternative ways to add authentication to Joux's three-party one-round protocol by employing a long-term key for each party. However, their analysis was only partially formal and when they did use a formal model they omitted reveal queries so that adversaries are assumed not to see old session keys. Hitchcock *et al.* [358] applied general compilers in the Canetti–Krawczyk model [178] with formal proofs. However, applying such compilers adds one round to the protocol so we end up with a two-round protocol, losing one of the main advantages of the Joux protocol.

Finally in 2009, Manulis *et al.* [518, 519] provided a proof in a strong model for a one-round authenticated version of the Joux protocol. Generalised versions of the protocol were later considered by the same authors plus Fujioka [285]. Protocol 9.17 shows the messages in the protocol of Manulis *et al.* Compared to their description, we have omitted the protocol identiifier for simplicity, but that should be included in

the computation of **K** if there is any possibility of confusion with messages of other protocols.

Information shared by principals: Constants D, E, F different from 0 or 1.

Phase 1. Message broadcast

U_1	U_2	U_3
$r_1 \in_R \mathbb{Z}_q$	$r_2 \in_R \mathbb{Z}_q$	$r_3 \in_R \mathbb{Z}_q$
$t_1 = g^{r_1}$	$t_2 = g^{r_2}$	$t_3 = g^{r_3}$
Broadcast U_1, t_1	Broadcast U_2, t_2	Broadcast U_3, t_3

Phase 2. Computation

Check $t_2, t_3 \in \mathbb{G}$	Check $t_1, t_3 \in \mathbb{G}$	Check $t_1, t_2 \in \mathbb{G}$
$\sigma_1 = \hat{e}(y_2 t_2, y_3 t_3)^{x_1 + D r_1}$	$\sigma_1 = \hat{e}(y_1 t_1^D, y_3 t_3)^{x_2 + r_2}$	$\sigma_1 = \hat{e}(y_1 t_1^D, y_2 t_2)^{x_3 + r_3}$
$\sigma_2 = \hat{e}(y_2 t_2^E, y_3 t_3)^{x_1 + r_1}$	$\sigma_2 = \hat{e}(y_1 t_1^D, y_3 t_3)^{x_2 + E r_2}$	$\sigma_2 = \hat{e}(y_1 t_1, y_2 t_2^E)^{x_3 + r_3}$
$\sigma_3 = \hat{e}(y_2 t_2, y_3 t_3^F)^{x_1 + r_1}$	$\sigma_3 = \hat{e}(y_1 t_1, y_3 t_3^F)^{x_2 + r_2}$	$\sigma_3 = \hat{e}(y_1 t_1, y_2 t_2)^{x_3 + F r_3}$
$\sigma_4 = \hat{e}(y_2 t_2^E, y_3 t_3^F)^{x_1 + D r_1}$	$\sigma_4 = \hat{e}(y_1 t_1^D, y_3 t_3^F)^{x_2 + E r_2}$	$\sigma_4 = \hat{e}(y_1 t_1^D, y_2 t_2^E)^{x_3 + F r_3}$

$$\mathbf{K} = H(\sigma_1, \sigma_2, \sigma_3, \sigma_4, U_1, y_1, t_1, U_2, y_2, t_2, U_3, y_3, t_3)$$

Protocol 9.17: Manulis–Suzuki–Ustaoglu authenticated Joux protocol

Protocol 9.17 makes use of three constants, D, E, F. In the earlier version of the protocol [518] these values were instead derived from the participant public keys. The computation required for each principal is relatively high due to the pairing and exponentiation required for computation of each σ_j value, for $1 \leq j \leq 4$. In compensation the protocol has a security proof in a strong model similar to the eCK model. In particular it is assumed that the adversary can obtain ephemeral secrets r_i or long-term secrets x_i where the combination of leaked secrets does not trivially reveal the key. Manulis *et al.* [518, 519] provided a security proof on the assumption that the bilinear Diffie–Hellman problem is hard and H acts as a random oracle. Later Li and Yang [490] proved the security of a related protocol in similar security model but without relying on random oracles.

9.3.8 Comparing Authenticated Group Diffie–Hellman

Performance of the authenticated protocols in this section is strongly related to the performance of the related unauthenticated versions compared in Table 9.2. In most cases signatures are added to at least one message from every participant which adds

significantly to the computational burden. Important differences between authenticated protocols can often be found in the security properties achieved. Some protocols aim to achieve mutual authentication while other do not. In addition the number of rounds can be affected by the authentication method. Table 9.5 is limited to these aspects.

In Table 9.5 the first two rows refer to the GDH variants which do not have concrete authenticated versions. The row named BDB-KY refers to the application of the Katz–Yung compiler to the BDB protocol. Note that the MSU protocol has only three principals. Insider security has a number of different flavours, for example it may or may not include KCI resistance; these are not shown in the table but are discussed in the descriptions of the individual protocols.

Table 9.5: Properties of authenticated Diffie–Hellman protocols

	Insider secure	Security proof	Rounds	Mutual authentication
A-GDH.2 (9.11)	No	Insecure	m	No
SA-GDH.2 (9.13)	No	No	m	No
BM (9.14)	Yes	Yes	3	Yes
BDB-KY	No	Yes	3	Yes
BGS (9.15)	Yes	Yes	2	Yes
GNS (9.16)	Yes	Yes	2	Yes
MSU (9.17) (three-party)	Yes	Yes	1	No

Armknecht and Furukawa [39] derived general bounds on the communication complexity of authenticated group key exchange. They placed a minimum requirement that the protocol should provide forward secrecy and mutual authentication with insider security, which rules out some of the protocols we have looked at. They also assumed that a unique session identifier is available to all parties (the so-called *common reference string* model) which is by no means always realistic and we have not generally assumed it in our descriptions. From their assumptions, Armknecht and Furukawa showed that any protocol with m principals must use at least two rounds and exchange $m^2/2 + m/2 - 3$ messages in total.

9.4 Identity-Based Group Key Establishment Protocols

In Chap. 7 we examined identity-based key agreement for two parties. This idea can be generalised to multi-party protocols. Of course identity-based group key establishment protocols do not have to use key agreement, but published protocols seem mainly to have considered this option. As is usual with identity-based protocols, the convenience of using only identity information in place of public keys comes at the

price of requiring a trusted authority to generate secret keys on behalf of principals. In the following, ID_i denotes the identifying string of entity U_i.

9.4.1 Koyama and Ohta Protocols

Koyama and Ohta [448] designed three protocols using different physical architectures. A succession of attacks and consequent countermeasures have been published, illustrating how difficult it is to consider all possibilities without a formal security statement. Indeed, it may be suggested that problems with these early protocols arose because at the time there was no clear notion of what are the desired security properties of such protocols.

Koyama and Ohta's first protocol type uses Diffie–Hellman in $\mathbb{Z}_n^*$ where $n = pq$ is an RSA modulus whose factorisation is known only to a trusted authority T. The authority chooses public and private keys e and d, with $ed \equiv 1 \bmod \phi(n)$. The principals are arranged in a ring where, as usual, we take U_{m+1} to mean U_1. The protocol can be viewed as a variant of Protocol 9.1 where authentication information is added to each message.

Each principal is issued with a secret key by the authority. Before issuing any secrets, the authority needs to choose a value M which will be the maximum number of users who can participate in a group (with the current parameter values). In our notation m is the size of the group so this means that $m \leq M$. Because the computational requirements for each principal increase with M it should not be chosen larger than necessary. Once M is fixed the authority can calculate the secret value S_i, using the following equation, and transfer it securely to principal U_i:

$$S_i = ID_i^{d^{M-1} \bmod \phi(n)}.$$

A successful protocol run is shown as Protocol 9.18. There are m rounds in a run. During each round, except for the last, each principal sends three values to the next principal in the ring. On receipt of a triple in round j the principal performs a check designed to verify that the triple has been processed by the previous j principals in the ring. If the check is successful the principal constructs and sends on its own triple. The first value in a triple, X_j, is a partial version of the final shared secret and the principal adds its random exponent to the received X_j value. The final shared secret $\mathbf{Z}$ is the following value.

$$\mathbf{Z} = g^{e^{m-1} r_1 r_2 \ldots r_m}.$$

Although not explicitly stated by Koyama and Ohta, it seems reasonable to say that Protocol 9.18 is designed to provide a strong form of key authentication. It is apparent that some group entity authentication is also intended. Indeed, Koyama and Ohta [448] state that if the check in round m succeeds then U_i can verify that the received messages came via U_{i-1}, U_{i-2}, ..., and U_{i-m+1} successively. However, there is nothing that any principal can use to verify that any received message is not replayed from an old protocol run. Therefore it is clear that no other principal need be present in any specific run.

Shared information: Identity information ID_i of principal U_i, public RSA key (n,e), element g of maximal order in $\mathbb{Z}_n^*$, prime c.
Information known only to U_i: Secret key S_i with $S_i^{e^{M-1}} \bmod n = ID_i$.

U_{i-1}	U_i	U_{i+1}
Round 1.		
	$r_i \in_R \mathbb{Z}_q$	$\xrightarrow{g^{er_i}, S_i g^{cr_i}, 1}$
Round 2.		
$\xrightarrow{X_1, Y_1, Z_1}$	$T_1 = X_1 Z_1^e$	
	$(Y_1^e / T_1^c)^{e^{M-2}} \stackrel{?}{=} ID_{i-1}$	$\xrightarrow{X_1^{er_i}, Y_1^e S_i^e X_1^{cr_i}, T_1}$
Round j.		
$\xrightarrow{X_{j-1}, Y_{j-1}, Z_{j-1}}$	$T_{j-1} = X_{j-1} Z_{j-1}^e$	
	$(Y_{j-1}^e / T_{j-1}^c)^{e^{M-j}} \stackrel{?}{=} ID_{i-1} ID_{i-2} \ldots ID_{i-j+1}$	
		$\xrightarrow{X_{j-1}^{er_i}, Y_{j-1}^e S_i^{e^{j-1}} X_{j-1}^{cr_i}, T_{j-1}}$
Round m.		
$\xrightarrow{X_{m-1}, Y_{m-1}, Z_{m-1}}$	$T_m = X_m Z_m^e$	
	$(Y_{m-1}^e / T_{m-1}^c)^{e^{M-m}} \stackrel{?}{=} ID_{i-1} ID_{i-2} \ldots ID_{i-m+1}$	
	$\mathbf{Z} = X_{m-1}^{r_i}$	

Protocol 9.18: Koyama–Ohta type 1 identity-based group key agreement protocol

Koyama and Ohta's second protocol type [448] allows every principal to communicate with each other principal, which they call a *complete graph* architecture. The original version of these protocols was attacked by Yacobi [747] and a new version was published by Koyama and Ohta the following year [449]. Koyama later refined these protocols again [447] due to a conspiracy attack by Shimbo and Kawamura [671]. We shall look only at the latest versions, which are anyway simpler than the originals.

Initialisation phase. The trusted authority chooses a modulus N to be the product of three distinct large primes: $N = pqr$. These primes must be large enough so that pq cannot be factorised. A random value e is chosen so that $N/2 < e < N$ and then d is calculated so that $de \bmod \phi(N) = 1$. An element g is chosen by the authority that is of maximal order in $\mathbb{Z}_N^*$. The values N, r, g, e are all made public, together with a hash function h.

Registration phase. Each principal U_i wishing to participate in a group must obtain a secret key S_i. This is generated by the authority from U_i's identifying informa-

tion ID_i as $S_i = ID_i^d \bmod N$. Notice that $ID_i = S_i^e \bmod N$. S_i must be transferred securely to U_i. If desired, the values d, p and q may be destroyed after all principals are registered.

Round 1. Each principal U_i chooses a value $P_i \in_R Z_{r-1}$ and generates a timestamp T_i. Principal U_i then calculates $X_i = g^{eP_i} \bmod N$ and $Y_i = S_i g^{h(X_i,T_i)P_i} \bmod N$. U_i broadcasts the triple (X_i, Y_i, T_i). On receipt of the triples (X_j, Y_j, T_j), for all $j \neq i$, U_i should verify that each timestamp T_j is fresh and, if so, check the following equation.

$$\frac{Y_j^e}{X_j^{h(X_j,T_j)}} \stackrel{?}{=} ID_j \bmod N.$$

If all $m-1$ checks are verified then U_i proceeds to round 2.

Round 2. Principal U_i chooses two values $r_i \in_R \mathbb{Z}_{r-1}$ and $Q_{i,j} \in_R \mathbb{Z}_{n-1}$ and generates a new timestamp T_i'. Then U_i calculates $A_{i,j} = (X_j + Q_{i,j}r)^{er_i} \bmod N$ and $B_{i,j} = S_i(X_j + Q_{i,j}r)^{h(A_{i,j},T_i')r_i} \bmod N$ for all $j \neq i$. U_i sends the triple $(A_{i,j}, B_{i,j}, T_i')$ to U_j. On receipt of its own $m-1$ triples, U_i verifies that the timestamps T_j' are fresh and if so performs the following check, for all $j \neq i$:

$$\frac{B_{j,i}^e}{A_{j,i}^{h(A_{j,i},T_j')}} \stackrel{?}{=} ID_j \bmod N.$$

If all checks pass then U_i calculates the shared secret $\mathbf{Z}$ as follows:

$$\mathbf{Z} = \left(\prod_{j=1}^{m} A_{j,i}\right)^{P_i^{-1} \bmod r-1} \bmod r.$$

If the protocol is run correctly then all principals obtain the same shared secret $\mathbf{Z} = g^{e^2(r_1+r_2+\ldots+r_m)}$. On the assumption that S_i is required in order to form (X_i, Y_i) pairs and $(A_{i,j}, B_{i,j})$ pairs, each principal does obtain implicit authentication of the shared key. There is no key confirmation provided. Forward secrecy is provided since knowledge of the S_i values does not appear to help in obtaining $\mathbf{Z}$. The computational cost per principal is, depending on the implementation, around $4m$ exponentiations, which is high compared with alternatives.

The third protocol type proposed by Koyama and Ohta uses a *star architecture*: messages are exchanged only between one principal, say U_1, and all other principals. The protocol is related to the previous example, but now the shared secret collapses to $\mathbf{Z} = g^{e^2 r_1}$ and so this becomes a key transport protocol in the sense that only U_1 contributes to the shared secret. Considerable computational savings result for all principals except U_1. Furthermore, forward secrecy of the key is still maintained. However, apart from U_1 no other principal now has any explicit assurance about the other members of the group.

The Koyama and Ohta protocols have evolved to take into account a number of attacks. However, there is no formal basis for their security and the problems in earlier versions do not inspire confidence. They also have high computational costs

in the versions that provide implicit key authentication. An implementation of these protocols was reported by Goscinski and Wang [331].

Chikazawa and Inoue [203] devised an identity-based group key establishment protocol that has similarities with Koyama and Ohta's type 3 protocol. One distinguished principal, say U_1, determines the key. The only messages sent are $m-1$ messages from U_1 to each other principal. Shimbo and Kawamura [671] found an attack on this protocol and it was then repaired by Chikazawa and Yamagishi [204]. Their protocol was recently found to be vulnerable to attacks by Shim [669], who proposed further modifications. Similar protocols were also published by Chen and Hwang [191, 369]. None of these protocols provides forward secrecy.

9.4.2 Protocols of Saeednia and Safavi-Naini

Saeednia and Safavi-Naini [642] proposed two identity-based group key establishment protocols based on the generalised Diffie–Hellman key agreement protocol of Burmester and Desmedt which was examined in Sect. 9.2.6. Their idea is to use identity information of adjacent principals to gain authentication of keying material. A composite modulus $n = pq$ is used to provide a trapdoor so that only the trusted authority is able to calculate user secrets related to identity information. Two bases are used, g and h, which both have order $p'q'$ where $p' = (p-1)/2$ and $q' = (q-1)/2$ are also prime. In an initialisation stage, all principals are issued with two secrets by the authority; these can only be found with knowledge of $p'q'$ (equivalent to knowing the factorisation of n).

Both of the proposed protocols use this same initialisation. The first protocol is computationally simpler but each party only uses the identity information of its adjacent principals to gain assurance that these are the correct entities. Thus there is no assurance as to who are the other protocol principals involved so this constitutes only a weak form of key authentication. In Saeednia and Safavi-Naini's second proposal a form of signature is broadcast by all principals, thereby allowing stronger authentication. This is shown in Protocol 9.19.

A close look at Protocol 9.19 reveals that it includes exactly Protocol 9.9 of Burmester and Desmedt. Of course this means that the shared secret,

$$\mathbf{Z} = g^{r_1 r_2 + r_2 r_3 + \ldots + r_m r_1}.$$

is exactly the same, and is calculated by each principal in the same way, as in that protocol. The additional fields used in Protocol 9.19 constitute the signature of each principal on the exchanged t_i and X_i values. However, it is also necessary for the t_i values to be broadcast so that all principals can sign them; in the Burmester–Desmedt protocol these need only be passed to adjacent principals.

The signature generation and verification requires $2m$ exponentiations for each principal; this dominates the computational requirements of the protocol except when m is very large. Knowledge of the long-term secrets S_i and T_i does not help in finding r_i and so forward secrecy is provided.

Information held by trusted authority: Primes p', q' with $p = 2p' + 1$ and $q = 2q' + 1$ both prime and modulus $n = pq$.
Shared information: Modulus n, elements g and h of order $p'q'$, integer $u < \min(p', q')$ and hash function H.
Information held by U_i: $S_i = g^{ID_i^{-1} \bmod p'q'}$, $T_i = h^{-ID_i^{-1} \bmod p'q'}$.

Phase 1.

U_i chooses $r_i \in_R \mathbb{Z}_{p'q'}$ and calculates $t_i = g^{r_i}$
U_i broadcasts t_i
U_i calculates $c = H(t_1, t_2, \ldots, t_m), X_i = (t_{i+1}/t_{i-1})^{r_i}, w_i = S_i^{H(X_i)r_i} T_i^c$

Phase 2.

U_i broadcasts X_i, w_i
U_i checks $w_j^{ID_j} h^c \stackrel{?}{=} t_j^{H(X_j)}$ for $j \neq i$
U_i calculates $\mathbf{Z} = t_{i-1}^{mr_i} X_i^{m-1} X_{i+1}^{m-2} \ldots X_{i-2}$

Protocol 9.19: Saeednia–Safavi-Naini identity-based group key agreement protocol

9.4.3 ID-Based Group Key Agreement and Pairings

In Chap. 7 we examined how elliptic curve pairings have been used widely to design protocols for identity-based key agreement. It may perhaps be expected that the same would be true of group key exchange, but in fact there are relatively few identity-based group key exchange protocols. A closer look shows that the two-party case is well suited to pairings because ephemeral and long-term keys from two participants can be paired together. We saw in Sect. 9.2.7 that pairings can be used to extend the Diffie–Hellman construction to three-party key exchange in a natural way, but this protocol is no longer identity-based. Extending to more parties with pairings can be done, for example by building up using trees as with tree-based Diffie–Hellman. However, there may be little reason for doing this since pairing computation is generally more expensive than exponentiation.

Most of the authenticated group protocols discussed in Sect. 9.3.1 use long-term keys only in the signatures used to authenticate other messages. Identity-based signatures, using pairings or not, can certainly be applied there. However, this is arguably not of special interest since conceptually identity-based signatures are not fundamentally different from ordinary PKI-based signatures.

Choi *et al.* [205] presented a bilinear version of the DBD protocol (Protocol 9.9) and then integrated that with an identity-based authentication scheme to provide a authenticated group key agreement protocol. They showed that this protocol is secure under the decision bilinear Diffie–Hellman assumption and using random oracles. However, no efficiency comparison was made with the ordinary DBD protocol authenticated with normal signatures. Moreover, Choi *et al.* [205] did not consider

insider attacks which were observed by Zhang and Chen [770]. Later authors have developed enhanced protocols with better efficiency [242, 446, 782] or additional properties such as anonymity [709, 726].

9.5 Group Key Agreement without Diffie–Hellman

Just as with the two-party case, almost all multi-party key agreement protocols are based on Diffie–Hellman key agreement. However, it is possible to use alternative techniques which can lead to advantages in efficiency. We will look at several examples in this section.

9.5.1 Pieprzyk and Li's Key Agreement Protocol

Secret sharing allows a group of users to cooperate to derive a secret value. Some researchers have shown how this can be exploited as a building block in group key establishment. The protocol described below employs the threshold scheme of Shamir [664] introduced in Sect. 1.3.6. We will not present the full detail, but note that recovery of secrets from a set of shares can be achieved using only additions and multiplications of the share values.

The idea to adapt secret sharing for *key broadcasting* seems to have been first proposed by Laih *et al.* [469] in a special case, and more generally by Berkovits [91]. The idea is to use an $(m+1, 2m)$ threshold scheme for which each receiving principal possesses one share; m further shares are broadcast when the key is established. Pieprzyk and Li [614] used the same basic idea to design a group key agreement protocol but now each principal sets up its own threshold scheme. Instead of recovering each principal's secret during a protocol run, an image of the secret is found. This allows the same secret sharing scheme to be used multiple times.

Initialisation phase. This phase must be run before any group key is established. Each principal U_i engages in an $(m+1, 2m)$ secret sharing scheme in $\mathbb{Z}_q$ with every other principal. Using Shamir's scheme this means that U_i chooses a polynomial

$$f_i(z) = a_{i,0} + a_{i,1}z + \ldots + a_{i,m}z^m$$

with coefficients randomly chosen in $\mathbb{Z}_q$. The shares are values $f_i(x)$ with $1 \leq x \leq 2m$. There are twice as many shares as principals, and they are divided into two sets: $s_{i,j}(1) = f_i(2j-1)$ and $s_{i,j}(2) = f_i(2j)$ with $1 \leq j \leq m$. The first set is kept secretly by U_i, while principal U_j is given one share, $s_{i,j}(2)$, from the second set. The share must be sent in a secure way.

The secret associated with U_i is $f_i(0)$ which could be recovered from any $m+1$ shares. The shared secret for the whole group will be $F(0)$ where $F(x) = f_1(x) + f_2(x) + \ldots + f_m(x)$. But neither $F(0)$ nor any $f_i(0)$ will be recovered during any protocol run. This allows multiple group keys to be found following one initialisation phase.

Key agreement. This phase is run each time a new group key is required. The same group may run this phase multiple times following one initialisation phase.

1. A new base value α is chosen for this session. A third party chooses this by taking a random value r and broadcasting the value $\alpha = g^r$. It is not necessary to keep the value of α (or even r) secret.
2. Principal U_i broadcasts the m values $\beta_{i,j} = \alpha^{s_{i,j}(1)}$, for $1 \leq j \leq m$, which are images of its first set of shares.
3. Each principal (indeed any eavesdropper too) can find the image of m shares of the sum of the secret shares by calculating $\alpha^{F(2j-1)} = \prod_{i=1}^{m} \beta_{i,j}$ for $1 \leq j \leq m$.
4. Principal U_j can also obtain $\alpha^{F(2j)} = \alpha^{\sum_{i=1}^{m} f_i(2j)}$. Thus U_i has the image of $m+1$ shares which allows calculation of the shared secret $\mathbf{Z} = \alpha^{F(0)}$ since addition and multiplication of the exponents can be done by multiplication and exponentiation of α.

Pieprzyk and Li provided an informal proof that the protocol maintains the confidentiality of the session key against an adversary who observes multiple runs of the protocol (with the same initialisation phase). They also proposed a final step in the key agreement protocol in which each principal broadcasts a hash of three values: the principal's identity, the shared secret $\mathbf{Z}$ and the value α. This gives a weak form of key confirmation; notice that any principal in the protocol can form all such values.

The computational requirements for each principal are mainly due to working with the exponents of α. Calculation of the $\beta_{i,j}$ values requires m exponentiations by each principal, and there is also a multi-exponentiation required to calculate the shared secret. A useful feature of the protocol is that the key agreement phase (excluding key confirmation) can be completed in one round. However, this is only true following some earlier initialisation, while there are a number of other drawbacks of this protocol.

- Confidence in freshness of the key depends on a random α being available. This means that the third party supplying this value must be trusted for this purpose.
- The initialisation phase must be run when a different group wishes to establish a key. Also there are $m-1$ secrets that need to be stored by each principal for each group that it belongs to.
- The secret shares are long-term keys whose compromise leads to compromise of any session key. Therefore forward secrecy is not provided.

Pieprzyk and Li also proposed an extended protocol that allows any subgroup to form a session key. A limitation of this extension is that users are not able to verify which members of the group take part in any particular protocol run. A related protocol has been proposed by Anzai *et al.* [36].

9.5.2 Tzeng–Tzeng Protocols

The protocols of Tzeng and Tzeng [717] have the same algebraic setting as Diffie–Hellman but the shared secret is $\mathbf{Z} = g^{r_1+r_2+\ldots+r_m}$ where r_i is the ephemeral private

input of principal U_i. Calculation of $\mathbf{Z}$ uses public information together with the long-term private key of any principal, and consequently the protocols do not provide forward secrecy.

In order to provide some element of fault tolerance, use is made of a *proof of knowledge* which allows any entity to check that the message elements sent by each party are properly constructed. We denote by $\mathtt{PEDL}(X_1, X_2, \ldots, X_m)$ a proof that the discrete logarithm of X_i to the base y_i is the same for each i with $1 \leq i \leq m$. The computational cost of generating each proof is m exponentiations with a similar cost to verify it.

Tzeng and Tzeng proposed two protocols, the first one of which is shown as Protocol 9.20. This protocol makes use of a signature scheme; they specify an ElGamal signature variant, but it is shown as a generic signature in Protocol 9.20. Their second protocol incorporates private key information into the proof instead of using a conventional signature.

Shared information: Session identifier *SID*.

Phase 1.

U_i chooses $r_i \in_R \mathbb{Z}_q$.

U_i calculates $t_i = g^{r_i}$ and broadcasts the following:

- $u_{i,j} = y_j^{r_i}$ for $1 \leq j \leq m$
- $\mathtt{PEDL}(u_{i,1}, u_{i,2}, \ldots, u_{i,m})$
- $\mathsf{Sig}_{U_i}(t_i, SID)$.

Phase 2.

U_i computes $t_j = u_{j,i}^{x_i^{-1}}$.

U_i verifies proof of knowledge from each U_j and signature on (t_j, SID).

U_i calculates $\mathbf{Z} = (u_{1,i} \cdot u_{2,i} \cdot \ldots \cdot u_{m,i})^{r_i^{-1}}$.

Protocol 9.20: Tzeng and Tzeng's group key agreement protocol

Protocol 9.20 has some interesting properties. The fault tolerance property ensures that if one or more of the proof verifications fails then the party concerned can be eliminated from the calculation of the shared secret. However, this mechanism only works on the strong assumption that the broadcast channel provides integrity of all messages; otherwise a malicious insider can send different proofs to different principals. The cost of verifying each of the proofs from the other $m-1$ principals is also high; about $m(m-1)$ exponentiations for each principal.

Tzeng and Tzeng [717] pointed out that their protocols can be completed in one (synchronous) round, which means that they meet the bound of Becker and Wille for contributory key agreement with broadcasts discussed in Sect. 9.2.9. However, their protocols require the session identifier *SID* to be known by all participating

principals. Unless this session identifier is agreed beforehand their protocols cannot be completed in one round. Tzeng and Tzeng also claimed a proof of security, but they provide no reduction proof for a powerful adversary, concentrating instead on the properties of the proof of knowledge. Boyd and González Nieto [143] have shown an explicit attack on the second of the protocols.

9.5.3 Boyd–González Nieto Group Key Agreement

Boyd [132] proposed a protocol using generic encryption and signature techniques together with a hash function possessing special properties. Its main virtue is its simplicity and efficiency; perhaps its main drawback is a lack of forward secrecy.

Protocol 9.21 shows the message flows. One principal, say U_1, is distinguished and sends its random value r_1 to each other user in an authenticated and confidential way. The other users only have to broadcast their messages so that all principals in $\mathbf{U}$ receive all the random r_i values. U_1 signs the value r_1 together with the identities of all principals in the group. Since this message is the same for every principal it only needs to be formed and sent once in a broadcast to all users. The value of r_1 is sent to user U_i encrypted with that principal's public key, K_i. The protocol then has two stages. In the first stage U_1 broadcasts one signature and $m-1$ encryptions. In the second stage each U_i broadcasts its r_i value.

Shared information: Hash function h. Signature verification with respect to U_1.
Information held by U_1: Public encryption key with respect to U_i for $2 \leq i \leq m$.

Phase 1:

U_1 broadcasts $\mathbf{U}, \mathsf{Sig}_{U_1}(\mathbf{U}, h(r_1))$.
U_1 broadcasts $\mathsf{Enc}_{U_2}(r_1), \mathsf{Enc}_{U_3}(r_1), \ldots, \mathsf{Enc}_{U_m}(r_1)$.

Phase 2:

U_i broadcasts r_i, for $2 \leq i \leq m$.
Each U_i calculates

$$\mathbf{Z} = \mathsf{MAC}_{r_1}(h(r_2) \oplus h(r_3) \oplus \ldots \oplus h(r_m)).$$

Protocol 9.21: Boyd–González Nieto group key agreement protocol

The security is based on the properties of the MAC function and h. It must not be possible to calculate $\mathsf{MAC}_{r_1}(.)$ without knowledge of r_1; since r_1 is sent confidentially to group members, no outsider can calculate $\mathbf{Z}$. Key freshness is provided by the one-wayness of h; no party is able to force an old value if at least one principal chooses its r_i freshly. Each principal gets an assurance from U_1 regarding the members of the group, and so U_1 must be trusted for this purpose. A generalisation of the protocol allows any principals to choose and encrypt a value for all other principals. This can reduce trust but increases the computation and communication.

The protocol can be completed with m broadcast messages. The computation required for U_1 is one signature and $m-1$ public key encryptions. Other principals use only one signature verification and one public key decryption. Compromise of any principal's decryption key results in compromise of $\mathbf{Z}$ so, as mentioned above, forward secrecy is not provided.

Protocol 9.21 is a contributory key agreement protocol in the sense of Becker and Wille [66]. Each of the broadcast messages can be sent simultaneously in one synchronous round. Therefore this protocol satisfies Becker and Wille's bound of one synchronous round for key agreement with broadcasts, mentioned in Sect. 9.2.9.

Boyd and González Nieto [143] claimed a security proof for an almost identical protocol in the Bellare–Rogaway model. The differences are that the MAC is replaced by a function f that is assumed to act as a random oracle, and the shared secret is calculated as $\mathbf{Z} = f(r_1, r_2, \ldots, r_m)$. Choo *et al.* [211] later demonstrated an unknown key-share attack on Protocol 9.21 and pointed out some oversights in the proof. As usual, such an attack can be prevented by including the identities of the group members in a key derivation function. Gorantla *et al.* [328] subsequently provided a new security proof which incorporates this change and avoids any use of random oracles.

Mailloux *et al.* [513] proposed to use a signcryption scheme to replace the separate encryption and signature functions in Phase 1 of Protocol 9.21. They also included a shared session identifier in all signed messages and used batch verification to improve efficiency. They prove security of their protocol in the random oracle model assuming secure signature and signcryption. Their protocol does not provide forward secrecy.

9.5.4 Generic One-Round Group Key Agreement from Multi-KEM

Gorantla *et al.* [329] proposed a simple generic construction for group key agreement using a *multi-KEM* (mKEM). An mKEM [680] is a generalisation of a key encapsulation mechanism which allows encapsulation of a new key for each of m parties in the set $\mathbf{P}$. The simple protocol has the following structure.

1. Each party U_i encapsulates a random key K_i for all other members of the group and broadcasts the resulting value C_i.
2. On receipt of each C_j, U_i decapsulates to obtain key K_j. U_i also computes the session identifier as $\mathsf{sid} = (C_1, C_2, \ldots, C_m, \mathbf{P})$. The session key is computed as

$$\mathbf{K} = f_{K_1}(\mathsf{sid}) \oplus f_{K_2}(\mathsf{sid}) \oplus \ldots \oplus f_{K_m}(\mathsf{sid})$$

   where $f_k(\cdot)$ is a pseudo-random function.

When instantiating the mKEM with any concrete CCA-secure version this construction was shown by Gorantla *et al.* [329] to be a secure group key agreement protocol without forward secrecy. With known instantiations of mKEM this is not a very efficient protocol but it does have the distinction of requiring only one round of communication.

9.5.5 Asymmetric Group Key Agreement

Wu *et al.* [739] introduced the notion of asymmetric group key agreement (AGKA). The motivation behind their protocol designs is to provide one of the common applications of group key agreement, namely the ability to send messages confidentially to a group of parties. However, in other ways the AGKA protocols are quite unlike the protocols which we have looked at in the rest of this chapter. In particular, no traditional shared session key is agreed during the protocol, but instead a shared public key is generated which can be used by any party, including outsiders to the group. Any group member can decrypt using a corresponding private key, which differs amongst different group members.

One of the attractive features of the AGKA protocols of Wu *et al.* [739] is that they only require one round of communication. In some situations, when the aim of the group key agreement is only to provide confidentiality of messages to group members, they can be more efficient than using proper group key agreement. Subsequently there have been constructions for AGKA protocols with additional properties, such as in the identity-based [771] and dynamic [776] settings.

9.6 Group Key Transport Protocols

The majority of published multi-party key establishment protocols rely on key agreement. In this section we look at a few group key establishment protocols using key transport. Many key transport protocols designed for two parties can be extended to the multi-party case in a straightforward fashion. We will first look at some simple examples based on different logical architectures. Then we look at a method of Mayer and Yung for systematically and formally extending two-party protocols for multiple parties. Following this we look briefly at proposals for key transport for dynamic groups using key hierarchies. Key transport protocols require a distinguished principal who generates the keys; we will sometimes call this principal the *group manager*.

Before looking at the specific protocols, it should also be mentioned that key pre-distribution schemes have been generalised to the multi-party case. The idea is that a trusted centre will distribute secrets to each member of a community in such a way that different subgroups can derive a shared secret known only to themselves. Such schemes may be regarded as ‘no-message’ protocols since each party is able to derive the correct key for a given group without any interaction. This means that additional measures must be taken in order to obtain a fresh session key. Stinson [697] has made a survey of such methods.

9.6.1 Burmester–Desmedt Star and Tree Protocols

As well as their key agreement protocol (Protocol 9.10) Burmester and Desmedt [168] proposed two simpler key transport protocols based on star and tree configurations. As with their key agreement protocol, these were initially proposed in unau-

thenticated versions. Authenticated versions can be achieved by applying a suitable compiler such as the Katz–Yung compiler variant proposed by Desmedt *et al.* [243].

Burmester–Desmedt Star Protocol

In the star protocol a distinguished principal U_1 acts as a manager and interacts with every other principal. First U_1 agrees a Diffie–Hellman key K_i with U_i and then sends the chosen session key $\mathbf{K}$ to U_i encrypted with K_i. The message flows are shown in Protocol 9.22. Note that only messages between U_1 and a single other principal, U_i, are shown; a similar exchange takes place between U_1 and all other principals.

U_1		U_i
$r_1 \in_R \mathbb{Z}_q$		
$t_1 = g^{r_1}$	$\xrightarrow{t_1}$	
		$r_i \in_R \mathbb{Z}_q$
	$\xleftarrow{t_i}$	$t_i = g^{r_i}$
Choose random $\mathbf{K}$		
$K_i = t_i^{r_1}$		
$c_i = \{\mathbf{K}\}_{K_i}$	$\xrightarrow{c_i}$	$K_i = t_1^{r_i}$
		Decrypt $\mathbf{K}$

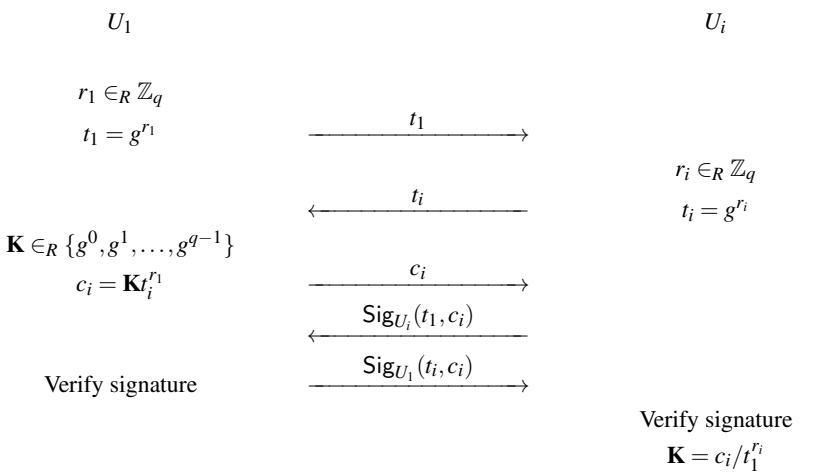
Protocol 9.22: Burmester–Desmedt star protocol

Burmester and Desmedt suggested that Protocol 9.22 may be extended to form an authenticated key transport protocol. To do so each principal U_i should authenticate the value t_i when sent to other principals. This authentication can be achieved using a generic method such as a digital signature.

Hirose and Yoshida [357] proposed a very similar group key establishment protocol employing their own signature scheme, which was also used in their key agreement scheme (Protocol 5.38). We show the message flows in Protocol 9.23 with a generic signature. Only messages between U_1 and a single other principal, U_i, are shown since the messages with all other principals are similar.

The confidentiality of $\mathbf{K}$ is protected only by the ephemeral keys and so forward secrecy is provided by this protocol. Apart from U_1, no principal gains explicit assurance about the other parties who obtain $\mathbf{K}$. The trust required in U_1 and the high computational load for this principal are the main drawbacks of this protocol. In an earlier protocol, Hirose and Ikeda [356] used a novel approach to key confirmation by allowing all users to contribute to a *multisignature* on the session key, using principal U_1 as a combiner. This multisignature can be verified by all principals using all of their public keys.

Shared information: Signature verification information for all principals.



Protocol 9.23: Hirose–Yoshida group key transport protocol

Burmester–Desmedt Tree Protocol

Burmester and Desmedt's tree protocol distributes the computational load more evenly amongst principals. Principals are arranged in a binary tree, each principal representing a node in the tree. (Leaf nodes may be left empty.) During the protocol each principal first agrees a Diffie–Hellman key with its parent node and its two child nodes. (The root of the tree has no parent so agrees a key only with its children, and the leaves of the tree have no children, so they agree a key only with their parent.) Once all keys are agreed, the root principal generates the session key $\mathbf{K}$ which is sent recursively by every parent to its children protected, by multiplication, with the shared key.

As with the star protocol, Burmester and Desmedt suggested to add authentication by having each party sign its t_i value for each of its recipients. The advantage of the tree protocol over the star protocol is that the computation is fairly even across all nodes: leaves require two exponentiations, internal nodes four exponentiations, and the root three exponentiations. Each principal needs to trust its parent node to correctly propagate the key. Once again there is no explicit assurance about other key recipients. Forward secrecy is provided because only ephemeral keys are used to protect $\mathbf{K}$.

The drawback of the original Burmester and Desmedt tree protocol [168] is that the number of rounds required increases as $1 + \lceil \log_2 m \rceil$ because $\mathbf{K}$ can only be propagated one level per round. However, Burmester and Desmedt later updated the protocol [169] to show that it can be reduced to two rounds at the cost of using broadcast communications. As in the original protocol, in the first round pairwise

keys are agreed between all nodes in the tree and their parent node (except for the root). However, for the second round these pairwise keys are propagated down to the leaves of the tree; each node, apart from the root, forms two ciphertexts consisting of the key shared with its parent encrypted with each of the keys shared with its two children. At the same time the root node encrypts the shared key K with the keys of its two children. Consequently every node in the tree can now recover K recursively. Of course this variant adds communication and decryption costs in comparison with the original protocol.

Desmedt and Lange [242] analysed a pairing-based variant of the Burmester and Desmedt tree protocol in which the binary tree is replaced by a tree of "triangles", each a set of three nodes. The Joux tripartite protocol (see Sect. 9.2.7) is run inside each triangle. Then the root node starts the propagation of the shared key down the tree of triangles. Desmedt and Lange showed, using assumptions regarding the relative cost of pairings and exponentiation, that their pairing-based version is more efficient than the original protocol. They also noted that increasing the branching factor in the tree still further can likely lead to further enhancements. Later, Desmedt and Mijayi [244] explored variants in which nodes can share an edge in different triangles, instead of sharing just a node as considered by Desmedt and Lange [242]. They showed that this leads to new protocols which can reduce the overall computation and also allow for tuning the protocol according to the computational power of different nodes.

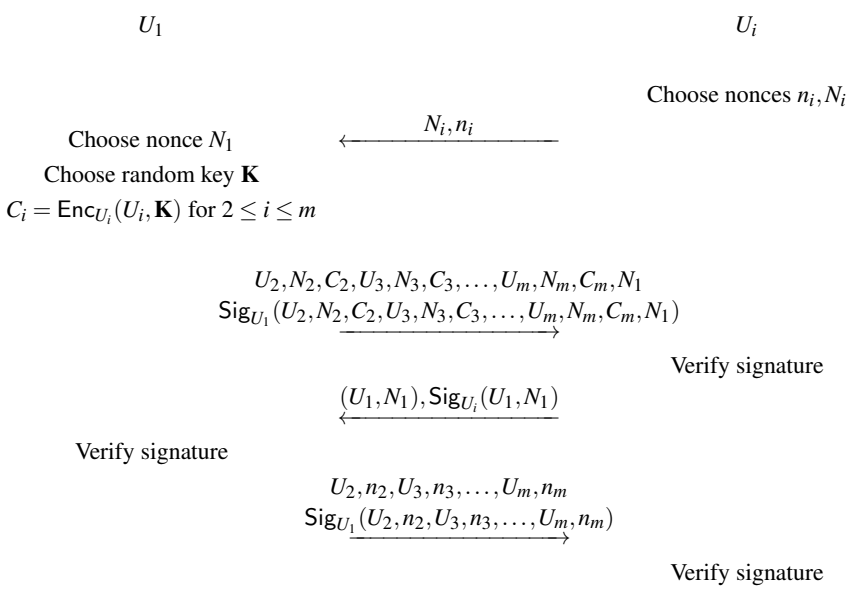
9.6.2 Mayer and Yung's Protocols

Mayer and Yung [530] made one of the early studies of provable security for multi-party key establishment. They proposed generic transformations that allow two-party key transport protocols to be extended to protocols for multi-party key transport in such a way that a proof for the two-party case automatically extends to the multi-party case. The transformations are quite straightforward, consisting basically of individual runs of the protocol between the group manager U_1 and the key recipients $U_2, \ldots, U_m$ but with the runs authenticated simultaneously. They use a formal model based closely on that of Bellare and Rogaway.

Mayer and Yung include two example protocols for which their method works. One is based on the shared key two-party protocol of Bellare and Rogaway (Protocol 3.2). This requires that each party already shares a secret with the key sender, which may not be realistic in many applications for authenticated key transport. Their second example is based on Blake-Wilson and Menezes' provable secure protocol (Protocol 4.19) examined in Chap. 4.

The message flows for the Mayer–Yung generalisation of the Blake-Wilson and Menezes protocol are shown in Protocol 9.24. Only flows between the server U_1 and U_i are shown, although in fact the messages from U_1 are broadcast to all other principals, and the messages from U_i to U_1 are gathered together for all i. The protocol is proven secure in the sense that if all parties accept the key then they must all agree on the messages exchanged and an adversary cannot obtain $\mathbf{K}$.

Shared information: Public encryption key and signature verification information for all principals.



Protocol 9.24: Mayer–Yung group key transport protocol

It may be observed that Protocol 9.24 uses one round more than Blake-Wilson and Menezes' Protocol 4.19 on which it is based. This is due to a deliberate reorganisation of the protocol for reasons that we now explain. In any protocol the adversary can always make one or more principals accept and complete the protocol, while others do not accept. This can be achieved by deleting the final protocol message to any recipient. Mayer and Yung called this situation an *inconsistency* and discuss two alternative consistency requirements.

Consistency type 1. If the key recipients $U_2, \ldots, U_m$ accept the session key then the group manager U_1 has also accepted it.

Consistency type 2. If the group manager U_1 accepts the session key then the recipients $U_2, \ldots, U_m$ have accepted it.

A protocol cannot provide both forms of consistency. Mayer and Yung argue that for group key transport protocols the first type of consistency is more useful than the second type. This is because denial of service attacks may become possible if U_1 is left in a state where it has not yet accepted, but the other principals believe that the protocol is complete. Since U_1 is usually a server, it is more vulnerable to this kind of problem. Mayer and Yung also provided a generic transformation that allows a protocol with one type of consistency to be transformed into a protocol

with the other type. Protocol 9.24 is the result of this transformation applied to the Blake-Wilson and Menezes protocol, which results in the additional message flow.

9.6.3 Key Hierarchies

Any group key transport protocol seems to require secure communication between the initiator and all other principals in order to initially establish the key. Solutions in which a single key is securely sent to each member, such as Protocols 9.22, 9.23 and 9.24, are appropriate in static groups. However, in dynamic groups it turns out that much may be gained from using a hierarchy of keys. When the group key is changed, in particular due to group members being added or deleted, different keys in the hierarchy can be used to save on both communications and computation.

To see the idea, consider the key hierarchy illustrated in Fig. 9.2 which is based on a binary tree. Each principal in the group shares a set of keys with the group manager. Each principal knows the key at one leaf of the tree and also every key that is the parent of a key that it knows. For example, in Fig. 9.2 principal U_5 knows the set of keys in bold, namely $\{k_5, k_{56}, k_{5678}, \mathbf{K}\}$. The key $\mathbf{K}$ at the root of the tree is the key shared by the group and can be used for secure communications.

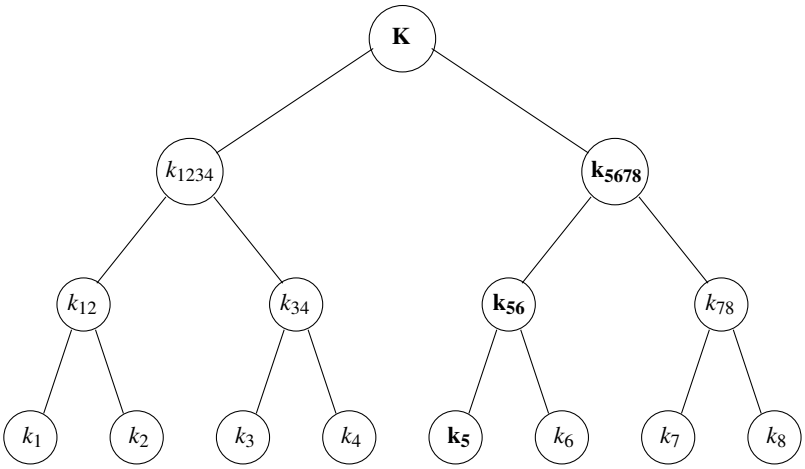


Fig. 9.2: Key hierarchy using binary tree

In this example each principal needs to store multiple keys, and indeed this is typical in key hierarchies. However, the potential advantage can immediately be seen if we consider what happens if it is required to remove user U_5 from the group. The set of keys known to U_5 needs to be changed, but this does not require the group manager to communicate directly with every principal. All principals on the left side

of the tree can know the new shared key $\mathbf{K}'$ if the manager encrypts $\mathbf{K}'$ with k_{1234} and sends it to these users (possibly by broadcasting it). Key k_{56} must be replaced by k'_{56} and can be sent to U_6 encrypted with k_6 (and k'_{56} could become the new individual key of U_6). Then key k_{5678} can be replaced by k'_{5678} and sent encrypted with k_{78} and with k'_{56}. In total the manager has to encrypt and send five new keys rather than having to encrypt a new key for all seven remaining principals which would occur with a flat hierarchy.

In general the manager needs to generate a new key for each level of the tree and encrypt each new key with the keys of the two child nodes (except for the deleted node). If $h = \lceil \log_2 n \rceil$ denotes the height of the tree then this means that the manager need only encrypt and send $2h - 1$ keys, which is a considerable saving over $n - 1$ when n is large. A similar saving can be made when adding new group members. (Recall that it is usually required that new members cannot know old group keys, so that $\mathbf{K}$ and all keys on the path from the new member to the root must be changed.)

The idea of using key hierarchies seems to have been published independently at about the same time by both Wong *et al.* [735] and Wallner *et al.* [725]. Both of these papers contain a detailed analysis of the various different related schemes. Wong *et al.* included experimental results on the comparative practical performance of different dynamic group operations depending on parameters such as group size and specific topology of the hierarchy. McGrew and Sherman [533] proposed key hierarchies based on one-way function trees in which the key at each node is defined by the keys at each of its child nodes. This can result in smaller message sizes. All these papers concentrate only on the dynamic operations and say little about establishing the initial group key.

9.7 Conclusion

Over the past 10–15 years, significant progress has been made in the topic of group key establishment protocols. Generic designs and improved concrete protocols have been developed, strong security models have been defined, and many protocols now have security proofs in the strong formal models. To a large extent, we can now say that design and analysis of group key agreement is at a similar state to that of two-party key agreement.

As in the two-party case, we have a number of alternative models and it is not always clear what is the best model to use, especially with regard to insider attacks. An integrated security model in which protocols can be fairly compared would be very beneficial. The three-party case has seen elegant solutions based on bilinear maps from elliptic curve pairings. At the time of writing the security status of multilinear maps remains unclear, but an efficient and secure instantiation would be a very significant tool for design of new group key agreement protocols, particularly if such maps can remain secure in the face of quantum computers.

A

Standards for Authentication and Key Establishment

Practitioners often look to standards bodies to recommend techniques that can be used with the assurance of independent verification of correctness and suitability. Several standards exist covering protocols of the type we have examined in this book. This appendix lists the main relevant standards and briefly summarises their contents. In many cases specific protocols have been examined in the body of the book and we refer to these where appropriate.

Standards are issued by many different bodies, both national and international. We have included mainly international standards; many national standards bodies issue their own versions of international standards with little or no alteration. Because of their international influence we also mention some US national standards. We additionally include some protocols which are not standardised by any organisation but which are widely deployed in certain settings.

A.1 ISO Standards

The International Organization for Standardization (`http://www.iso.ch`), also known as the ISO, has published numerous standards on cryptographic mechanisms and protocols.

A.1.1 ISO/IEC 9798

ISO issued the six-part standard ISO/IEC 9798 on the topic of entity authentication.

- Part 1: General (3rd edition, 2010).
- Part 2: Mechanisms using symmetric encipherment algorithms (3rd edition, 2008).
- Part 3: Mechanisms using digital. signature techniques (2nd edition, 1998; with amendment, 2010).
- Part 4: Mechanisms using a cryptographic check function (2nd edition, 1999).
- Part 5: Mechanisms using zero knowledge techniques (3rd edition, 2009).

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9

- Part 6: Mechanisms using manual data transfer (2nd edition, 2010).

The protocols in Part 2 of the standard were examined in Sect. 3.2.3. Some of the protocols in Part 3 of the standard were examined in Sect. 4.2.1. The protocols in Part 4 of the standard were examined in Sect. 3.2.4.

A.1.2 ISO/IEC 11770

ISO issued the six-part standard ISO/IEC 11770 on the topic of key management. Some of the protocols in Parts 2 and 3 of the standard are strongly related to protocols in Parts 2 and 3 of ISO/IEC 9798.

Part 1: Framework (2nd edition, 2010).

Part 2: Mechanisms using symmetric techniques (2nd edition, 2008). The protocols in this part of the standard were examined in Sects. 3.3.4 (the server-less protocols) and 3.4.4 (the server-based protocols).

Part 3: Mechanisms using asymmetric techniques (3rd edition, 2015). Protocols in this part of the standard include both key transport protocols which have been examined in Sect. 4.3.1 and key agreement protocols which were summarised in Sect. 5.6.

Part 4: Mechanisms based on weak secrets (2nd edition, 2017). This part of the standard is concerned with password-based protocols. It includes five of the password-based key agreement protocols described in Chap. 8. Two are 'balanced' protocols where both parties hold the shared password: SPEKE (Protocol 8.5) and J-PAKE (Protocol 8.8). Three are augmented protocols where a server holds only the image of the client password: SRP (Protocol 8.18), AMP (Protocol 8.19) and AugPAKE (Protocol 8.20). There is also one *password-authenticated key retrieval* mechanism which allows a server to furnish a user with a strong secret while the user only stores a password. This is based on a scheme of Ford and Kaliski [281].

Part 5: Group key management (2011). This part of the standard is devoted to group key establishment using a key distribution centre. (This means that it excludes group key agreement which was covered extensively in Chapter 9.) Much of the document is taken up with describing key management structures for groups, particularly various types of trees including the octopus structure shown in Protocol 9.8. There are two specific mechanisms included in the standard, but both are given abstract descriptions. Protocols based on key chains are also described. With regard to security properties the standard only considers what are termed *forward secrecy* and *backward secrecy* for dynamic groups: the former states that leaving group members should not learn future keys[1] while the latter states that new group members should not learn past keys.

[1] Note that this definition of forward secrecy is different from what we use in most of this book.

Part 6: Key derivation (2016). Part 6 of ISO/IEC 11770 standardises key derivation functions in two classes called one-step and two-step functions. The one-steps functions compute one or more keys as the output of a hash function or, in one case, a MAC algorithm. The two-step functions apply a key extraction function followed by a key expansion function, the latter of which can be repeated to obtain further keys.

A related standard is ISO/IEC 15946 (Part 1, 3rd edition, 2016; Part 5, 2nd edition, 2017) which covers cryptography based on elliptic curves. This standard includes details for implementing elliptic curve mechanisms defined in Part 3 of ISO/IEC 11770.

A.1.3 ISO 9594-8/ITU X.509

A series of standards for directory systems was first issued in 1988 jointly by ISO and CCITT (which was later re-formed as ITU). The section of the standard numbered 9594-8 (ISO version) or X.509 (ITU version) was known as the *Authentication Framework*. This section of the standard provides information on how to use a directory to store public key certificates, including the format of certificates. It also includes examples of how to use the certificates to provide authentication and key establishment. In the most recent version of the standard (8th edition, 2017) the *Authentication Framework* has been renamed *Public-Key and Attribute Certificate Frameworks*. Portions of the X.509 specification have also been published by the Internet Engineering Task Force, which we note below.

Under the heading 'Strong authentication' three key establishment protocols were presented. Unfortunately there were some problems with the protocols in the first version of the standard and they were subsequently updated. The protocols have been examined in Sect. 4.3.5.

A.2 IETF Standards

The Internet Engineering Task Force (IETF) (`http://www.ietf.org`) produces standards concerned with development of Internet technology. In contrast to many other standards bodies, the IETF works in an open way and all its documents are freely available on the Internet. Documents are first proposed as 'Internet-Drafts'; after a series of revisions and discussion, some become standardized in documents that are known (for historical reasons) as 'Requests for Comments' (RFCs). There are different types of RFCs, including informational, historical, proposed standards, and Internet standards.

Some RFCs are also developed by the Crypto Forum Research Group (CFRG), a working group of the IETF's sister organisation Internet Research Task Force (IRTF), which aims to bridge theory and practice by bringing new cryptographic techniques to the Internet community and serving as an 'expert crypto consultant' to IETF working groups.

Table A.1 summarises some prominent RFCs that specify or use prominent authenticated key exchange protocols. Some of these were examined earlier and we provide pointers to where they are described in this book. For many of these protocols there are additional RFCs available which cover related information or extensions of these protocols for different applications. We also include Internet-Drafts in a few areas which are under development but as of writing have not yet progressed to RFCs.

A.3 IEEE P1363 Standards

The Institute of Electrical and Electronics Engineers (IEEE) is a US-based institution with a worldwide membership. The IEEE Standards Association (`http://standards.ieee.org`) issues standards in a wide range of electronics and communications areas. The IEEE P1363 Working Group has developed a number of standards under the heading Standard Specifications for Public-Key Cryptography. At the time of writing there are five active standards in force from the P1363 group.

1363-2000: Public-Key Cryptography. The original P1363 standard includes specifications for public key algorithms and key agreement protocols based on discrete logarithms. The only key establishment protocols included in P1363-2000 are basic Diffie–Hellman key agreement and authenticated versions using the Unified Model and MQV. These were examined in Sects. 5.4.4 and 5.4.5.

1363a-2004: Amendment 1: Additional Techniques. This update to the original standard incorporates implementation details for elliptic curves.

1363.1-2008: Public Key Cryptographic Techniques Based on Hard Problems over Lattices. This includes encryption and signature algorithms but no authentication or key establishment protocols.

1363.2-2008: Password-Based Public-Key Cryptographic Techniques. There is much overlap with the ISO/IEC 11770-4 standard (see Sect. A.1.2). The 1363.2-2008 standard includes versions of PAK (Protocol 8.3), PPK (Protocol 8.4), SPEKE (Protocol 8.5), AMP (Protocol 8.19), B-SPEKE (Protocol 8.16) and a variant known as W-SPEKE, PAK-Z (Protocol 8.15) and SRP (Protocol 8.17). Most of the protocol specifications include versions for both the elliptic curve setting and for discrete logarithms in finite fields.

1363.3-2013: Identity-Based Cryptographic Techniques Using Pairings. There are many different identity-based cryptographic primitives included in this standard but only two key agreement protocols, namely Protocol 7.12 of Wang and the variant of Smart's protocol discussed in Sect. 7.3.2.

A.4 NIST Standards

The US National Institute of Standards and Technology (NIST) (`https://www.nist.gov`) issues a range of standards and guidelines covering cryptographic techniques. Federal Information Processing Standards (FIPS) are standards developed by

Table A.1: Some RFCs for key establishment protocols

RFC	Year	Description	Status	Section
Kerberos				
1510	1993	Kerberos v5	Historic	3.4.3
4120	2005	Kerberos v5	Proposed standard	3.4.3
Transport Layer Security (TLS) protocol				
2246	1999	TLS v1.0	Proposed standard	6.3
4346	2006	TLS v1.1	Proposed standard	6.3
5246	2008	TLS v1.2	Proposed standard	6.3
6347	2012	Datagram TLS v1.2	Proposed standard	6.5
8446	2018	TLS v1.3	Proposed standard	6.3
Internet Key Exchange (IKE) protocol				
2409	1998	IKEv1	Proposed standard	5.5.5
4306	2005	IKEv2	Proposed standard	5.5.6
7296	2014	IKEv2	Internet standard	5.5.6
Secure Shell (SSH) protocol				
4252	2006	SSHv2 Authentication protocol	Proposed standard	
4253	2006	SSHv2 Transport Layer protocol	Proposed standard	
Other protocols				
2412	1998	Oakley protocol	Informational	5.5.3
3830	2004	Multimedia Internet Keying (MIKEY)	Proposed standard	
6189	2011	ZRTP (for real-time streaming)	Informational	
Other protocols – password-based				
5683	2010	PAK protocol	Informational	8.3.1
6628	2012	AugPAKE protocol	Experimental	8.4.5
7664	2015	Dragonfly protocol	Experimental	8.3.3
Public key management				
5280	2008	X.509v3 Public Key Infrastructure	Proposed standard	
6962	2013	Certificate Transparency	Experimental	
–	2018	Automatic Certificate Management Environment (ACME)	Internet-Draft	

NIST for use in non-military government computing systems. NIST Special Publications (SPs) provide additional guidelines for certain techniques.

The following list describes FIPS and Special Publications that NIST has issued that address key exchange and authentication:

FIPS 196: Entity Authentication Using Public Key Cryptography (1997). This document includes two of the protocols contained in ISO/IEC 9798-3. FIPS 196 was withdrawn in 2015 as being obsolete.

FIPS 140-2 Annex D: Approved Key Establishment Techniques for FIPS PUB 140-2, Security Requirements for Cryptographic Modules (draft 2017). FIPS 140-2 specifies how cryptographic hardware and software modules should be assessed for security. Annex D provides a list of approved key establishment algorithms that can be used in FIPS 140-2-certified modules. It is a very short document that simply consists of references to other documents, the most important of which are the following two Special Publications.

SP-800-56A revision 2: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (2013). This document defines two fundamental key agreement primitives: Diffie–Hellman (DH), and Menezes–Qu–Vanstone (Sect. 5.4.5), which can be instantiated over either finite fields or elliptic curves. The Special Publication then shows a variety of key agreement protocols built from these primitives, categorized according to the number of static and ephemeral keys used in the protocol. Using the notation (xE,yS) to denote a protocol with x ephemeral keys and y static keys, NIST SP-800-56A rev. 2 gives protocols for (2E,2S), (2E,0S), (1E,2S), (1E,1S), and (0E,2S). Variants also included which provide key confirmation. Several of the protocols in NIST SP-800-56A are based on protocols in the ANSI X9.42 and X9.63 standards.

SP-800-56B revision 1: Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography (2014). This document focuses on public key encryption using the RSA algorithm, and includes three key exchange protocols based on key transport using RSA public key encryption.

NIST's Post-Quantum Crypto Project (http://nist.gov/pqcrypto), running from 2016 through to 2023–2025, aims to standardise one or more key encapsulation mechanisms believed to be resistant to attacks by quantum computers.

A.5 Other Standards and Protocols

A.5.1 ANSI

ANSI, the American National Standards Institute, is a non-profit organisation that develops a range of voluntary standards. The ANSI X9 committee (http://www.x9.org) provides standards for financial services industries. It has published numerous standards covering cryptographic algorithms and authentication mechanisms. Two standards, X9.42 and X9.63, are devoted to key agreement protocols.

X9.42 Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography (2001). This covers key agreement for protocols based on conventional discrete logarithms. It includes Diffie–Hellman in static, ephemeral and hybrid (one-pass) versions, as well as the Unified Model and MQV protocols in full and one-pass versions.

X9.63 Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography (2001, revised in 2011). This includes elliptic curve versions of all but one of the protocols in X9.42. In addition it includes versions with key confirmation, an elliptic curve STS protocol (see Sect. 5.5.2) and two elliptic curve key transport protocols.

A.5.2 Widely Deployed Protocols

There are a variety of other purpose-specific standards and protocols, maintained by industry consortia, non-profit organizations, or the original creator of the protocol.

Bluetooth (https://www.bluetooth.com) versions 3.0 (2009) and higher support elliptic curve Diffie–Hellman key exchange. Authentication can be provided using the Secure Simple Pairing (SSP) protocol, which can operate in a variety of modes. The numeric comparison and passkey entry modes involve comparing or entering a 6-digit PIN (derived using a message authentication code) from one device to another, thereby authenticating the key exchange. The 'just works' mode does not involve a PIN entry or comparison, and instead works with no user interaction (or just the user pressing a single button to confirm pairing); this mode does not protect the key exchange from man-in-the-middle attacks.

EMV 'Chip-and-PIN'. (https://www.emvco.com) The EMV chip-and-PIN system is used in credit and bank cards to secure physical card transactions; there are more than 6 billion EMV cards deployed worldwide. Chip-and-PIN cards establish a secure channel with a (point-of-sale) terminal using a channel establishment protocol involving elliptic Diffie–Hellman key exchange, signatures and certificates, and an authenticated encryption scheme. The protocol aims to achieve standard security notions for security channels (authentication, key establishment, confidentiality and integrity of messages) as well as an unlinkability property. See [165] for an academic analysis of the EMV protocol.

Mobile phones. The GSM protocol was one of the first digital mobile phone protocols. It provides authentication of a mobile device to a cell tower, and encryption for that communication link. Session keys are established using symmetric cryptography techniques based on a long-term shared secret key between the mobile device and the carrier. The design of the original GSM authentication and key establishment protocol, as well as the proprietary cryptographic functions used therein, contains a variety of weaknesses which can be readily exploited. A new security protocol called AKA (Authentication and Key Agreement, not to be confused with the AKA protocol described in Sect. 4.3.9) was developed as part

of the 3G and LTE standards. AKA depends on symmetric cryptography based on long-term shared secret keys. The core AKA protocol design eliminates some of the design weaknesses in the GSM protocol, and the new proprietary ciphers used in AKA are somewhat better (though still have some weaknesses), but a security risk remains when 3G phones downgrade to GSM when no 3G connection can be established.

Tor (https://www.torproject.org). The Tor anonymity network allows users to transmit their communications through a series of relays and obscure the source and destination of their communication. Links between relays are encrypted using keys established by an elliptic curve Diffie–Hellman-based protocol called ntor, discussed in Sect. 1.5.11.

Off-the-Record Messaging (OTR) (https://otr.cypherpunks.ca). The OTR protocol allows secure instant messaging between two parties with confidentiality and integrity of communication and mutual authentication of parties, as well as subsequent deniability of communications. The key exchange component of OTR is a variant of the SIGMA protocol (Sect. 5.5.6) in a finite-field Diffie–Hellman group. Parties can authenticate to each other either by checking the fingerprint (hash) of their long-term signature keys, or by checking that they both know a shared secret passphrase; the latter is checked using a zero knowledge protocol called the Socialist Millionaires' Protocol, which can be viewed as a form of password-authenticated key exchange protocol. Every time either party sends an instant message, they send along a fresh Diffie–Hellman public key and a new shared secret is established; this construction is called a 'ratchet' and yields an aggressive form of forward secrecy.

Signal (https://whispersystems.org/docs). The Signal protocol was first introduced in a messaging app called TextSecure, later renamed Signal, and has since been adopted by WhatsApp, Facebook Messenger, and Google's Allo instant messaging app. Signal allows secure messaging between two parties with confidentiality and integrity of communication. Signal is designed to work in an asynchronous scenario where one of the parties is offline for a period of time. In Signal each party has a variety of long-term, medium-term, and ephemeral public keys. A session is initially established using a 'triple Diffie–Hellman' sub-protocol where the initial session key is the hash of three (or four) Diffie–Hellman shared secrets. Like OTR, parties send fresh Diffie–Hellman public keys along with subsequent messages so new shared secrets can be established. However, Signal also includes a 'symmetric ratchet': if the same party sends two messages in a row without receiving a reply from the peer, it applies a key derivation function to the session key to derive a new one. This results in the 'double ratchet' protocol: an asymmetric (Diffie–Hellman) ratchet when fresh Diffie–Hellman keys have been exchanged, and a symmetric (KDF) ratchet when fresh Diffie–Hellman keys have not been exchanged. See [222] for an academic analysis of the Signal protocol.

B

Tutorial: Building a Key Establishment Protocol

This appendix is a tutorial introduction to the topic of key establishment. It is intended to lead the beginner (who may already be familiar with cryptographic algorithms and communications protocols) through the fundamentals of the subject by following common mistakes in a hypothetical protocol design. At the same time it enables us to start establishing some common concepts and notation used throughout this book. The procedure we will use to explain the ideas is to try to design a protocol for key establishment from first principles. Problems with the protocol will be revealed in stages through presentation of legitimate attacks so that each may be solved in turn. Eventually a good protocol is achieved.

Before we start designing any protocol the communications architecture must be established. We choose one common scenario, but the reader should be aware that there is a wide variety of alternatives (these are explored in Sect. 1.2). Our scenario has a set of users, any two of whom may wish to establish a new key for use in securing their subsequent communications through cryptography. Such a key is known as a *session key*. It is important to understand that successful completion of key establishment (and entity authentication) is only the beginning of a secure communications session: once an appropriate key has been established its use comes in protecting the real data to be communicated with whatever cryptographic mechanisms are chosen.

In order to achieve their aim the users interact with an entity called the *server* which will also engage in the protocol. All users trust the server to execute the protocol faithfully and not to engage in any other activity that will deliberately compromise their security. Furthermore, the server is trusted to generate the new key and to do so in such a way that it is sufficiently random to prevent an attacker gaining any useful information about it.

Our protocols thus involve three entities (often called *principals* or *parties* in the literature). These are two users whom we denote A and B (often expanded to *Alice* and *Bob*) and the trusted server S. The aim of the protocol is for A and B to establish a new secret key K_{AB} which they can use for subsequent secure communications. The role of S is to generate K_{AB} and transport it to A and B. The aims of the protocol can be summarised as follows.

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9

- At the end of the protocol the value of K_{AB} should be known to both A and B, but to no other parties with the possible exception of S.
- A and B should know that K_{AB} is newly generated.

Let us begin with a completely naïve outlook. A protocol to achieve transport of a new session key K_{AB} is shown in Fig. B.1. The protocol consists of three messages. Firstly, user A contacts S by sending the identities of the two parties who are going to share the new session key; secondly, S returns the key K_{AB} to A; finally, A passes K_{AB} on to B. Before we examine the (lack of) security of this protocol we should acknowledge that, as a specification of the protocol, Fig. B.1 is significantly incomplete. Note the following features.

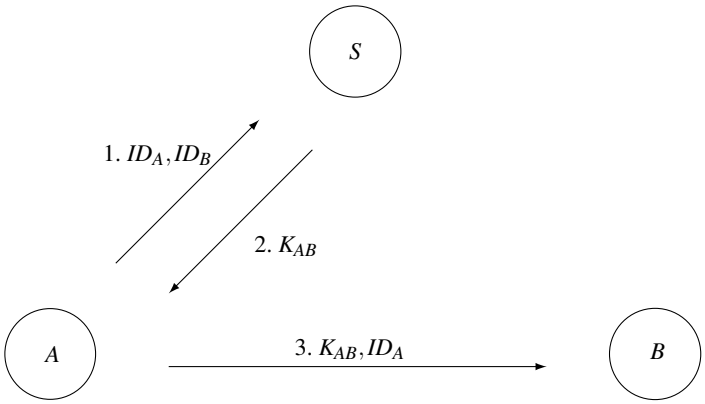


Fig. B.1: First protocol attempt

- Only the messages passed in a successful run of the protocol are specified. In particular there is no description of what happens in the case that a message of the wrong format is received or that no message is received at all. The inclusion of all such information is standard in specifying ordinary communications protocols, and is essential to prove basic functional properties [636]. It is unfortunate that it is commonplace to omit such information from specifications of cryptographic protocols in the academic literature.
- There is no specification of internal actions of principals. In many protocols the internal actions are fairly obvious; for example, they may be simply to calculate what is required to output the next message. But in others there are numerous alternatives and the choice can have security-critical relevance.
- It is implicitly assumed that A and B 'know' that the received messages are part of the protocol. It is common practice to omit such details which would be required for a networked computer to be able to track the progress of a particular protocol run. This may include details of which key should be used to decrypt a received message which has been encrypted.

Despite the obvious limitations associated with specifying protocols by showing only the messages of a successful run, it remains the most popular method of describing cryptographic protocols in the literature. An equivalent representation of the protocol of Fig. B.1 is to show the messages sent in a successful run, each preceded by the principals between whom it is intended to pass. For example, the protocol in Fig. B.1 could be specified as follows.

1. $A \rightarrow S: ID_A, ID_B$
2. $S \rightarrow A: K_{AB}$
3. $A \rightarrow B: K_{AB}, ID_A$

Protocol B.1: First protocol attempt in conventional notation

This notation is often preferred when a more compact description is desired. However, it does make the protocol harder to visualise. Furthermore, it may be argued that it tends to reinforce, even more than the diagrammatic version, an assumption that messages automatically reach their destination securely. In the authors' experience it is usually helpful when trying to understand a protocol for the first time to draw a figure showing the message flows between the principals involved.

In this book we generally use two different formats for protocol descriptions. One is the format of Protocol B.1, showing only the protocol messages. However, we usually describe any internal actions explicitly in the commentary. The second format is to show the protocol flows between principals and include, where appropriate, any internal checking that is required. Although the second format is more complete, it is more cumbersome and we generally use it when the details are less obvious. For both formats we try to be as precise as possible about the properties required of cryptographic mechanisms. (Such properties are discussed in Sect. 1.3.)

B.1 Confidentiality

The reader is probably already aware of the obvious problem with our first attempt. Nevertheless it is our purpose here to be explicit about our assumptions. The problem is that the session key K_{AB} must be transported to A and B but to *no other entities*. It is an assumption that the adversary, against whose attacks we are implementing our security, can eavesdrop on all messages that are sent or received. This is a realistic assumption in typical communications systems such as the Internet and corporate networks. Indeed, if this possibility can be discounted then there is probably no need to apply security at all.

Security Assumption 1 *The adversary is able to eavesdrop on all messages sent in a cryptographic protocol.*

In order to provide confidentiality it is necessary to use a cryptographic algorithm and associated keys. For now we will simply make the assumption that the server S initially shares a secret key with each user of the system. The keys K_{AS} and K_{BS} are shared between A and S, and between B and S, respectively. The need to keep K_{AB} confidential leads immediately to our second protocol attempt shown in Fig. B.2.

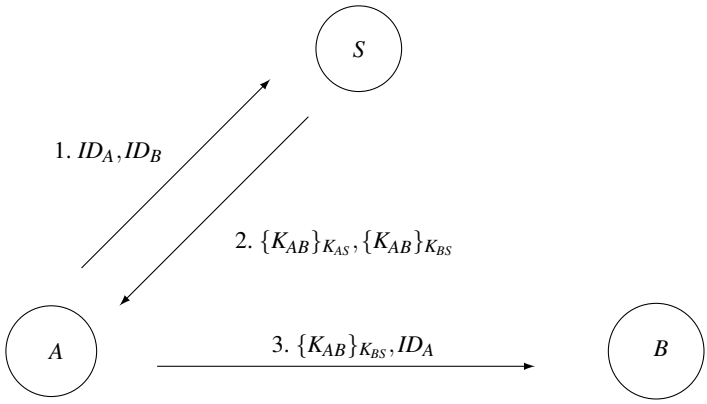


Fig. B.2: Second protocol attempt

This protocol starts, as our first attempt, with A sending to S the identities of the two parties who are to share the session key. S generates the session key K_{AB}, then encrypts it with each of the keys K_{AS} and K_{BS} and sends the result back to A. Principal A then relays the encrypted key to B along with her identity so that B knows who else has this key.

This protocol is just as insecure in an open environment as our first attempt, but for a completely different reason. A passive eavesdropper cannot see K_{AB} since encrypted messages may only be read by the legitimate recipients who have the keys required to decrypt. This brings us to the question of cryptographic algorithms. Fundamental cryptographic properties and their use in cryptographic protocols are reviewed in Sect. 1.3. An understanding of cryptographic algorithms is essential for the correct design of protocols. However, the details of the exact cryptographic algorithm used are often irrelevant and such details are frequently avoided during both protocol design and analysis. All the attacks described in this section are independent of the cryptographic algorithms used.

In many analyses an assumption of 'perfect cryptography' is made, which means that there is nothing gained whatsoever by the adversary in observing an encrypted message. Naturally this convenient assumption brings with it certain responsibilities for the protocol designer. In order to make a design practical there must be suitable cryptographic algorithms available that satisfy the security requirements. Furthermore, these requirements must be made explicit as part of the protocol specification. An alternative approach is to include an abstract model of the cryptographic algo-

rithms as part of the protocol specification. The protocol analysis can then aim to verify that any advantage gained by the adversary from eavesdropping on the protocol is sufficiently small. A comparison of these two different views has been made by Abadi and Rogaway [7].

B.2 Authentication

The problem with the protocol in Fig. B.2 is not that it gives away the secret key K_{AB}. The difficulty is that the information about who else has the key is not protected. We need to take into account that the adversary is not only able to eavesdrop on messages sent, but can also capture messages and alter them.

Security Assumption 2 *The adversary is able to alter all messages sent in a cryptographic protocol using any information available. In addition the adversary can re-route any message to any other principal. This includes the ability to generate and insert completely new messages.*

We may summarise the situation by saying that the adversary has complete control of the channel over which protocol messages flow. The reason for the difficulty in designing authentication protocols now begins to clarify. In contrast to ordinary communications protocols, there is an unknown and unpredictable principal involved. Although there may be no more than four or five messages involved in a legitimate run of the protocol, there are an infinite number of variations in which the adversary participates. These variations have an unbounded number of messages and each must satisfy the protocol's security requirements. Over the past 10 years there have been various methods devised to gain confidence in the security of protocols; some of these are discussed in Chap. 2.

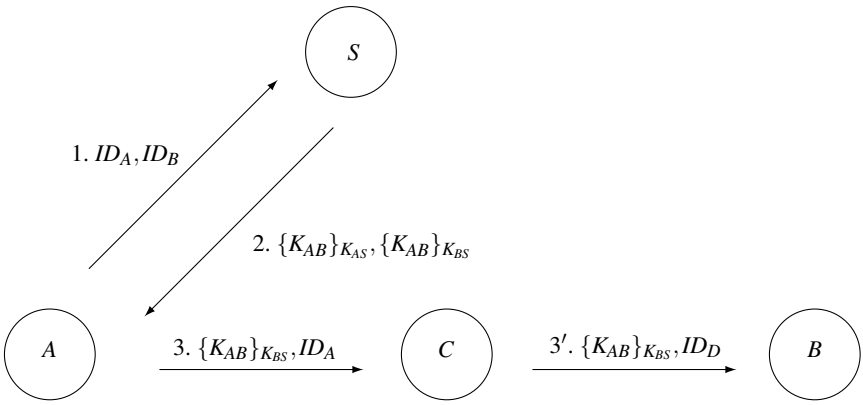


Fig. B.3: Attack on the second protocol attempt

One attack on our second protocol is shown in Fig. B.3. The attack is very simple. The adversary C simply intercepts the message from A to B and substitutes D's identity for A's (where D could be any identity including C's own). The consequence is that B believes that he is sharing the key with D whereas he is in fact sharing it with A. The subsequent results of this attack will depend on the scenario in which the protocol is used, but may include such actions as B giving away information to A which should have been shared only with D. Although C does not obtain K_{AB} we still regard the protocol as broken since it does not satisfy our requirement that the users should know who else knows the session key. However, another attack on the protocol does allow C to obtain the session key as shown in Fig. B.4.

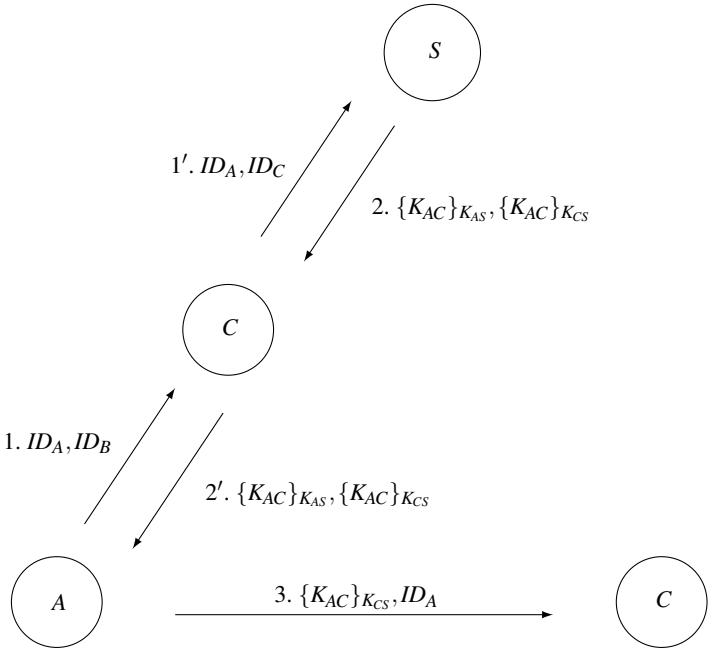


Fig. B.4: Alternative attack on second protocol attempt

In this second attack C alters the message from A to S so that S encrypts the key K_{AC} with C's key, K_{CS}, instead of with B's key. Since A cannot distinguish between encrypted messages meant for other principals she will not detect the alteration. Notice that K_{AC} is simply a formal name for the bitstring representing the session key so will be accepted by A. Also C collects the message from A intended for B so that B will not detect any anomaly. The result of this attack is that A will believe that the protocol has been successfully completed with B whereas in fact C knows K_{AC} and so can masquerade as B as well as learn all the information that A sends to B. Notice

that, in contrast to the previous attack, this one will only succeed if C is a legitimate user known to S. This, again, is a quite realistic assumption – it is widely agreed that insiders are often more of a threat than outsiders.

Security Assumption 3 *The adversary may be a legitimate protocol participant (an insider), or an external party (an outsider), or a combination of both.*

To overcome the attack, the names of the users who are to share K_{AB} need to be bound cryptographically to the value of K_{AB}. This leads to the protocol shown in Fig. B.5 where the names of A and B are included in the encrypted messages received from S. It can easily be checked that in this protocol neither of the two attacks on the protocol of Fig. B.2 will succeed. It is a necessary property of the encryption algorithm used by S that it is not possible to alter the value of the encrypted messages. The importance of distinguishing between this *integrity* property and the *confidentiality* property of cryptographic algorithms is discussed further in Sect. 1.3.

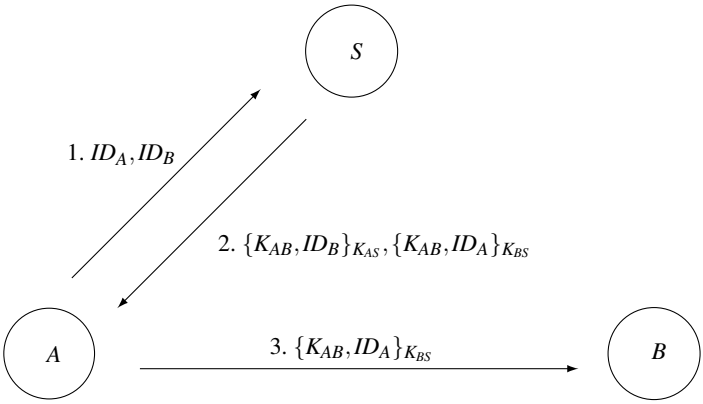


Fig. B.5: Third protocol attempt

B.3 Replay

So far our protocol has improved to the point where an adversary is unable to attack it by either eavesdropping or altering the messages sent between the legitimate users. However, even now the protocol is not good enough to provide security in normal operating conditions. The problem stems from the difference in quality between the long-term key-encrypting keys shared initially with S, and the session keys K_{AB} generated for each protocol run.

One reason that a new key is generated for each session is that session keys are expected to be vulnerable to attack. They are likely to be used with a variety

of data of regular formats, making them targets for cryptanalysis; also they may be placed in relatively insecure storage and could easily be discarded carelessly after the session is closed. A second reason for using new session keys is that communications in different sessions should be separated. In particular, it should not be possible to replay messages from previous sessions.

For these reasons a whole class of attacks becomes possible based on the notion that old keys may be *replayed* in a subsequent session. Notice that even if A is careful in the management of session keys used by her, compromise of a session key by B may still allow replay attacks when A communicates with B.

Security Assumption 4 *An adversary is able to obtain the value of the session key K_{AB} used in any sufficiently old previous run of the protocol.*

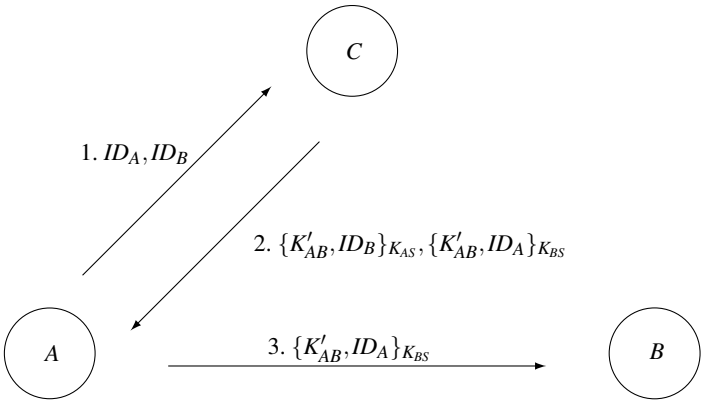


Fig. B.6: Attack on third protocol attempt

Figure B.6 shows a replay attack on our third protocol attempt. This time C intercepts the message from A to S – indeed S plays no part in the protocol. The key K'_{AB} is an old session key used by A and B in a previous session; by Security Assumption 1, C can be expected to know the encrypted messages via which K'_{AB} was transported to A and B. By Security Assumption 4, C can be expected to know the value of K'_{AB}. Thus when A completes the protocol with B, C is able to decrypt subsequent information encrypted with K'_{AB} or insert or alter messages whose integrity is protected by K'_{AB}.

Notice that the replay attack in Fig. B.6 can still be regarded as successful even if C has not obtained the value of K'_{AB}. This is because C has succeeded in making A and B accept an old session key. Such an attack may be useful to C because it allows C to replay messages protected by K'_{AB} which were sent in the previous session. In addition it enables C to obtain more ciphertext with the same key which might aid in cryptanalysis.

There are various mechanisms that may be employed to allow users to check that session keys have not been replayed. These are considered in detail in Sect. 1.3.7, but for now we will improve our protocol using the popular method called *challenge–response*. In this method, A will generate a new random value N_A commonly known as a *nonce* (a number used only once).

Definition 40. *A* nonce *is a random value generated by one party and returned to that party to show that a message is newly generated.*

Principal A sends her nonce N_A to S at the start of the protocol together with the request for a new key. If this same value is received with the session key then A can deduce that the key has not been replayed. This deduction will be valid as long as the session key and nonce are bound together cryptographically in such a way that only S could have formed such a message.

Since B does not directly contact the server S, it is inconvenient for him to send his own nonce to S to be returned with K_{AB}. How, then, is he able to gain the same assurance as A that K_{AB} has not been replayed?

If the encrypted key for B is included in the encrypted part of A's message, then A can gain assurance that it is fresh. It is tempting to believe that A may pass this assurance on to B in an extra handshake: B will generate a nonce N_B and send this to A protected by K_{AB} itself. Then A can use the session key to send a related reply to B. This leads to a fourth protocol attempt shown in Fig. B.7.

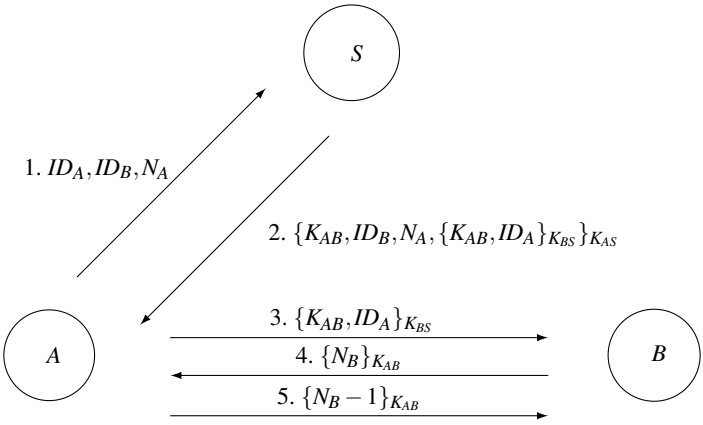


Fig. B.7: Fourth protocol attempt (Needham–Schroeder)

The protocol in Fig. B.7, which we have reached by a series of steps, is one of the most celebrated in the subject of cryptographic protocols. It was published by Needham and Schroeder in 1978 [581] and has been the basis for a whole class of related protocols. Unfortunately the original Needham–Schroeder protocol is vulnerable to an almost equally celebrated attack due to Denning and Sacco [240]. Their

attack illustrates that there was a flaw in the above argument used to justify the protocol design. This can be pinpointed to an assumption that only A will be able to form a correct reply to message 4 from B. Since the adversary C can be expected to know the value of an old session key, this assumption is unrealistic. In the attack in Fig. B.8, C masquerades as A and is thus able to persuade B to use the old key K'_{AB}.

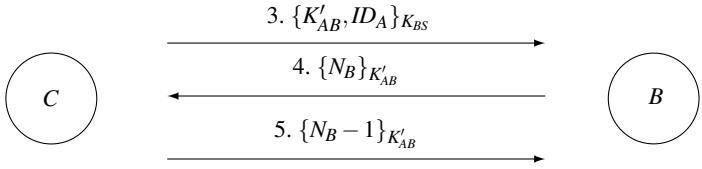


Fig. B.8: Attack on fourth protocol attempt

As usual, once an attack has been spotted, it is relatively easy to suggest ways of overcoming it. The method we choose here is to throw away the assumption that it is inconvenient for both B and A to send their challenges to S. This leads to our final protocol shown in Fig. B.9.

It would be rash to claim that this protocol is secure before giving a precise meaning to that term. Yet we can say that it avoids all the attacks that we have met so far, as long as the cryptographic algorithm used provides the properties of both confidentiality and integrity, and the server S acts correctly. The security of a protocol must always be considered relative to its goals; the different possible goals are considered in detail in Chap. 2, as well as ways to gain greater assurance that they are met.

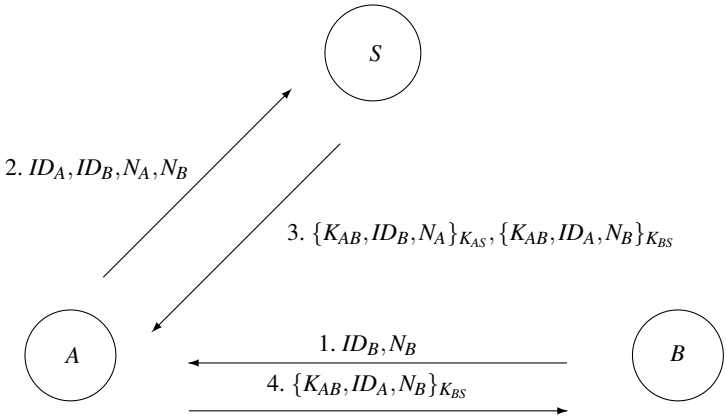


Fig. B.9: Fifth protocol attempt

To enable both users to send their nonces to S, the protocol is now initiated by B who sends his nonce, N_B, first to A. A adds her nonce, N_A, and sends both to S who is now able to return K_{AB} in separate messages for A and B, which can each be verified as fresh by their respective recipients. Although it may seem that we have achieved more than the protocol in Fig. B.7 using fewer messages, A has in fact achieved less in this protocol. This is because A in Fig. B.7 could verify not only that the key is new and known only by A, B and S, but also that B has in fact received the key. This property of *key confirmation* is achieved due to B's use of the key in message 4, assuming that $\{N_B\}_{K_{AB}}$ cannot be formed without knowledge of K_{AB}. In the protocol of Fig. B.9, neither A nor B can deduce at the end of a successful protocol run that the other has actually received K_{AB}. We leave it as an exercise for the reader to construct variant protocol runs showing why this is the case.

It is worth noting that it has been a very common pattern for published protocols to be subsequently found to be flawed. Each time a new protocol is designed and an attack is found our understanding of protocol design improves. The frequent occurrence of such attacks should be a caution, particularly for implementers of security protocols. Much of the recent research in cryptographic protocols has been devoted to remedying the situation.

B.4 Design Principles for Cryptographic Protocols

Abadi and Needham [6] proposed a set of principles intended to act as 'rules of thumb' for protocol designers. They were derived from observation of the most common errors that have been found in published protocols. By following these principles designers are less likely to make errors, but it must be emphasised that there can be no guarantee that this will result in a good protocol. Furthermore, there are many examples of protocols that ignore one or more of the principles and yet are (believed to be) secure.

The principles are paraphrased in Table B.1. Many of them can be related to discussions and examples earlier in this tutorial. For example, Principle 9 could refer directly to the attack shown in Fig. B.8. We believe that most of the principles are self-explanatory. Abadi and Needham discussed each principle in detail with examples to illustrate their application.

Table B.1: Abadi and Needham's principles for design of cryptographic protocols

1. Every message should say what it means: the interpretation of the message should depend only on its content.
2. The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.
3. If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.
4. Be clear about why encryption is being done.
5. When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message.
6. Be clear about what properties you are assuming about nonces.
7. If a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later replay a response.
8. If timestamps are used as freshness guarantees, then the difference between local clocks at various machines must be much less than the allowable age of a message.
9. A key may have been used recently, for example to encrypt a nonce, and yet be old and possibly compromised.
10. It should be possible to deduce which protocol, and which run of that protocol, a message belongs to, and to know its number in the protocol.
11. The trust relations in a protocol should be explicit and there should be good reasons for the necessity of these relations.

C

Summary of Notation

Notation is described in each chapter as it is introduced. In this appendix the main notational conventions are summarised.

A and B	Two users who wish to share a new session key
S	A trusted server
N_P	Random nonce value chosen by principal P
T_P	Timestamp chosen by principal P
K_{AB}	Key shared by A and B
C_P	Adversary C masquerading as principal P
$\mathsf{Enc}_P(M)$	Public key encryption of message M with public key of principal P
$\mathsf{Encap}_A(\cdot)$	Public key encapsulation of a shared secret with public key of party A.
$\mathsf{MAC}_K(M)$	Message authentication code tag of M using shared key K
$\mathsf{Sig}_P(M)$	Digital signature with appendix of message M by principal P
$\{M\}_K$	Symmetric encryption of message M with shared key K to provide confidentiality and integrity
$[\![M]\!]_K$	Encryption of message M with key K to provide confidentiality
$[M]_K$	One-way transformation of message M with key K to provide integrity
p	A large prime (usually at least 2048 bits)
q	A prime (typically of 256 bits) with $q \mid p-1$
$\mathbb{Z}_p$	The field of integers (under addition and multiplication) modulo p
$\mathbb{Z}_p^*$	The multiplicative group of non-zero integers modulo p
$\mathbb{G}$	A subgroup of $\mathbb{Z}_p^*$. Often a subgroup of order q, but sometimes equal to $\mathbb{Z}_p^*$
g	A generator of G
r_P	Random integer chosen by principal P
t_P	Ephemeral public keys: $t_P = g^{r_P}$
x_P	The private long-term key of principal P
y_P	The public key of principal P: $y_P = g^{x_P}$
$\mathbf{Z}$	The shared secret calculated by the protocol principals
$\mathbf{K}$	The session key calculated by the protocol principals
S_{AB}	The static Diffie–Hellman key of P and Q: $S_{AB} = g^{x_A x_B}$

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9

$H(.)$	A one-way hash function
$x \in_R X$	The element x is chosen uniformly at random from the set X
$F \stackrel{?}{=} G$	Verify that F and G evaluate to the same value
$\hat{e}$	Elliptic curve pairing: $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$
$\mathbf{U}$	The set of principals intended to share a conference session key
U_i	The i'th principal in $\mathscr{U}$, where $1 \leq i \leq m$
π	A key of short length, such as a password

References

1. Abadi, M.: Explicit communication revisited: Two new attacks on authentication protocols. IEEE Transactions on Software Engineering **23**(3), 185–186 (1997)
2. Abadi, M.: Two facets of authentication. In: 11th IEEE Computer Security Foundations Workshop, pp. 27–32. IEEE Computer Society Press (1998)
3. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the Pi calculus. In: D.A. Schmidt (ed.) Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, *Lecture Notes in Computer Science*, vol. 2986, pp. 340–354. Springer (2004). DOI 10.1007/978-3-540-24725-8_24
4. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the Pi calculus. ACM Trans. Inf. Syst. Secur. **10**(3) (2007). DOI 10.1145/1266977.1266978
5. Abadi, M., Lomas, T.M.A., Needham, R.: Strengthening passwords. Tech. Rep. 1997-033, Digital Systems Research Center, Palo Alto, California (1997)
6. Abadi, M., Needham, R.: Prudent engineering practice for cryptographic protocols. In: IEEE Symposium on Research in Security and Privacy, pp. 122–136. IEEE Computer Society Press (1994)
7. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology **15**(2), 103–127 (2002)
8. Abdalla, M.: Password-based authenticated key exchange: An overview. In: S.S.M. Chow, et al. (eds.) Provable Security - 8th International Conference, ProvSec 2014, *Lecture Notes in Computer Science*, vol. 8782, pp. 1–9. Springer (2014). DOI 10.1007/978-3-319-12475-9_1
9. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE password-authenticated key exchange protocol. In: 2015 IEEE Symposium on Security and Privacy, pp. 571–587. IEEE Computer Society (2015). DOI 10.1109/SP.2015.41
10. Abdalla, M., Benhamouda, F., Pointcheval, D.: Public-key encryption indistinguishable under plaintext-checkable attacks. In: J. Katz (ed.) Public-Key Cryptography - PKC 2015, *Lecture Notes in Computer Science*, vol. 9020, pp. 332–352. Springer (2015). DOI 10.1007/978-3-662-46447-2_15
11. Abdalla, M., Bohli, J., Vasco, M.I.G., Steinwandt, R.: (Password) authenticated key establishment: From 2-party to group. In: S.P. Vadhan (ed.) Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, *Lecture Notes in Computer Science*, vol. 4392, pp. 499–514. Springer (2007). DOI 10.1007/978-3-540-70936-7_27
12. Abdalla, M., Bresson, E., Chevassut, O., Pointcheval, D.: Password-based group key exchange in a constant number of rounds. In: M. Yung, et al. (eds.) Public Key Cryptog-

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9

raphy - PKC 2006, *Lecture Notes in Computer Science*, vol. 3958, pp. 427–442. Springer (2006). DOI 10.1007/11745853_28
13. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Password-authenticated group key agreement with adaptive security and contributiveness. In: B. Preneel (ed.) Progress in Cryptology - AFRICACRYPT 2009, *Lecture Notes in Computer Science*, vol. 5580, pp. 254–271. Springer (2009). DOI 10.1007/978-3-642-02384-2_16
14. Abdalla, M., Chevalier, C., Granboulan, L., Pointcheval, D.: Contributory password-authenticated group key exchange with join capability. In: A. Kiayias (ed.) Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, *Lecture Notes in Computer Science*, vol. 6558, pp. 142–160. Springer (2011). DOI 10.1007/978-3-642-19074-2_11
15. Abdalla, M., Chevalier, C., Manulis, M., Pointcheval, D.: Flexible group key exchange with on-demand computation of subgroup keys. In: D.J. Bernstein, T. Lange (eds.) Progress in Cryptology - AFRICACRYPT 2010, *Lecture Notes in Computer Science*, vol. 6055, pp. 351–368. Springer (2010)
16. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: S. Vaudenay (ed.) Public Key Cryptography - PKC 2005, *Lecture Notes in Computer Science*, vol. 3386, pp. 65–84. Springer (2005)
17. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. IEE Proceedings - Information Security **153**, 27–39 (2006)
18. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: A. Menezes (ed.) Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, *Lecture Notes in Computer Science*, vol. 3376, pp. 191–208. Springer (2005). DOI 10.1007/978-3-540-30574-3_14
19. Abdalla, M., Pointcheval, D.: A scalable password-based group key exchange protocol in the standard model. In: X. Lai, K. Chen (eds.) Advances in Cryptology - ASIACRYPT 2006, *Lecture Notes in Computer Science*, vol. 4284, pp. 332–347. Springer (2006). DOI 10.1007/11935230_22
20. Acıiçmez, O., Gueron, S., Seifert, J.P.: New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. In: S.D. Galbraith (ed.) Cryptography and Coding, *Lecture Notes in Computer Science*, vol. 4887, pp. 185–203. Springer (2007). DOI 10.1007/978-3-540-77272-9_12
21. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Béguelin, S.Z., Zimmermann, P.: Imperfect forward secrecy: How Diffie–Hellman fails in practice. In: I. Ray, N. Li, C. Kruegel (eds.) 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 5–17. ACM (2015)
22. Agnew, G., Mullin, R., Vanstone, S.: An interactive data exchange protocol based on discrete exponentiation. In: C.G. Günther (ed.) Advances in Cryptology – EUROCRYPT '88, *Lecture Notes in Computer Science*, vol. 330, pp. 159–166. Springer (1988)
23. Aiello, W., Bellovin, S.M., Blaze, M., Canetti, R., Ioannidis, J., Keromytis, A.D., Reingold, O.: Just fast keying: Key agreement in a hostile internet. ACM Trans. Inf. Syst. Secur. **7**(2), 242–273 (2004). DOI 10.1145/996943.996946
24. Al-Riyami, S.S., Paterson, K.G.: Certificateless public key cryptography. In: C.S. Laih (ed.) Advances in Cryptology - ASIACRYPT 2003, *Lecture Notes in Computer Science*, vol. 2894, pp. 452–473. Springer (2003)
25. Al-Riyami, S.S., Paterson, K.G.: Tripartite authenticated key agreement protocols from pairings. In: K.G. Paterson (ed.) Cryptography and Coding, 9th IMA International Conference, *Lecture Notes in Computer Science*, vol. 2898, pp. 332–359. Springer (2003)

26. Alawatugoda, J.: Generic construction of an eCK-secure key exchange protocol in the standard model. Int. J. Inf. Sec. **16**(5), 541–557 (2016). DOI 10.1007/s10207-016-0346-9
27. Alawatugoda, J., Stebila, D., Boyd, C.: Continuous after-the-fact leakage-resilient eCK-secure key exchange. In: IMA Int. Conf., *Lecture Notes in Computer Science*, vol. 9496, pp. 277–294. Springer (2015)
28. AlFardan, N.J., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.N.: On the security of RC4 in TLS. In: S.T. King (ed.) 22th USENIX Security Symposium, pp. 305–320. USENIX Association (2013)
29. AlFardan, N.J., Paterson, K.G.: Lucky thirteen: Breaking the TLS and DTLS record protocols. In: 34th IEEE Symposium on Security and Privacy, pp. 526–540. IEEE Computer Society (2013)
30. Alves-Foss, J.: Provably insecure mutual authentication protocols: The two-party symmetric-encryption case. In: 22nd National Information Systems Security Conference (1999). URL http://csrc.nist.gov/nissc/1999/proceeding/papers/p25.pdf
31. Amir, Y., Kim, Y., Nita-Rotaru, C., Tsudik, G.: On the performance of group key agreement protocols. Tech. Rep. CNDS-2001-5, The Center for Networking and Distributed Systems (CNDS), Johns Hopkins University (2001). URL http://www.cnds.jhu.edu/pub/papers/cnds-2001-5.ps
32. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: L.R. Knudsen (ed.) Advances in Cryptology - EUROCRYPT 2002, *Lecture Notes in Computer Science*, vol. 2332, pp. 83–107. Springer (2002)
33. Anderson, R., Lomas, T.M.A.: Fortifying key negotiation schemes with poorly chosen passwords. Electronics Letters **30**(13), 1040–1041 (1994)
34. Anderson, R., Needham, R.: Programming Satan's computer. In: J. van Leeuwen (ed.) Computer Science Today: Recent Trends and Developments, *Lecture Notes in Computer Science*, vol. 1000, pp. 426–440. Springer (1995)
35. Anderson, R., Needham, R.: Robustness principles for public key protocols. In: D. Coppersmith (ed.) Advances in Cryptology – Crypto '95, *Lecture Notes in Computer Science*, vol. 963, pp. 236–247. Springer (1995)
36. Anzai, J., Matsuzaki, N., Matsumoto, T.: A quick group key distribution scheme with 'entity revocation'. In: K.Y. Lam, et al. (eds.) Advances in Cryptology – ASIACRYPT '99, *Lecture Notes in Computer Science*, vol. 1716, pp. 333–347. Springer (1999)
37. Arazi, B.: Integrating a key distribution procedure into the digital signature standard. Electronics Letters **29**(11), 966–967 (1993)
38. Armando, A., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: K. Etessami, S.K. Rajamani (eds.) Computer Aided Verification, 17th International Conference, CAV 2005, *Lecture Notes in Computer Science*, vol. 3576, pp. 281–285. Springer (2005). DOI 10.1007/11513988_27
39. Armknecht, F., Furukawa, J.: On the minimum communication effort for secure group key exchange. In: A. Biryukov, et al. (eds.) Selected Areas in Cryptography - 17th International Workshop, SAC 2010, *Lecture Notes in Computer Science*, vol. 6544, pp. 320–337. Springer (2010)
40. Ars Technica: Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections (2015). URL http://arstechnica.com/security/2015/02/lenovo-pcs-ship-with-man-in-the-middle-adware-that-breaks-https-connections/

41. Ateniese, G., Steiner, M., Tsudik, G.: Authenticated group key agreement and friends. In: 5th ACM Conference on Computer and Communications Security, pp. 17–26. ACM Press (1998)
42. Ateniese, G., Steiner, M., Tsudik, G.: New multiparty authentication services and key agreement protocols. IEEE Journal on Selected Areas in Communications **18**(4), 628–639 (2000)
43. Aumann, Y., Rabin, M.O.: Authentication, enhanced security and error correcting codes. In: Advances in Cryptology - CRYPTO '98, *Lecture Notes in Computer Science*, vol. 1462, pp. 299–303. Springer (1998)
44. Aura, T.: Strategies against replay attacks. In: 10th IEEE Computer Security Foundations Workshop, pp. 59–68. IEEE Computer Society Press (1997)
45. Aura, T., Nikander, P.: Stateless connections. In: Y. Han, et al. (eds.) Information and Computer Security, First International Conference, *Lecture Notes in Computer Science*, vol. 1334, pp. 87–97. Springer, Beijing (1997)
46. Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J.A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., , Shavitt, Y.: DROWN: Breaking TLS using SSLv2. In: Proc. 25th USENIX Security Symposium (2016). URL https://drownattack.com/
47. Backes, M., Kate, A., Mohammadi, E.: Ace: an efficient key-exchange protocol for onion routing. In: T. Yu, N. Borisov (eds.) Proceedings of the 11th annual ACM Workshop on Privacy in the Electronic Society, WPES 2012, pp. 55–64. ACM (2012). DOI 10.1145/2381966.2381974
48. Backes, M., Mohammadi, E., Ruffing, T.: Computational soundness results for ProVerif - bridging the gap from trace properties to uniformity. In: M. Abadi, S. Kremer (eds.) Principles of Security and Trust - Third International Conference, POST 2014, *Lecture Notes in Computer Science*, vol. 8414, pp. 42–62. Springer (2014). DOI 10.1007/978-3-642-54792-8_3
49. Baek, J., Kim, K.: Remarks on the unknown key share attacks. IEICE Transactions Fundamentals **E83-A**(12), 2766–2769 (2000)
50. Bakhtiari, S., Safavi-Naini, R., Pieprzyk, J.: On password-based authenticated key exchange using collisionful hash functions. In: J. Pieprzyk, et al. (eds.) Information Security and Privacy, First Australasian Conference, *Lecture Notes in Computer Science*, vol. 1172, pp. 299–310. Springer (1996)
51. Balfanz, D., Durfee, G., Shankar, N., Smetters, D.K., Staddon, J., Wong, H.: Secret handshakes from pairing-based key agreements. In: 2003 IEEE Symposium on Security and Privacy (S&P 2003), pp. 180–196. IEEE Computer Society (2003). DOI 10.1109/SECPRI.2003.1199336
52. Bana, G., Adão, P., Sakurada, H.: Computationally complete symbolic attacker in action. In: D. D'Souza, T. Kavitha, J. Radhakrishnan (eds.) 32nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, Leibniz International Proceedings in Informatics, pp. 546–560 (2012)
53. Bard, G.V.: The vulnerability of SSL to chosen plaintext attack. Cryptology ePrint Archive, Report 2004/111 (2004). URL https://eprint.iacr.org/2004/111
54. Bardou, R., Focardi, R., Kawamoto, Y., Simionato, L., Steel, G., Tsay, J.: Efficient padding oracle attacks on cryptographic hardware. In: R. Safavi-Naini, R. Canetti (eds.) Advances in Cryptology - CRYPTO 2012, *Lecture Notes in Computer Science*, vol. 7417, pp. 608–625. Springer (2012)

55. Barnes, R., Thomson, M., Pironti, A., Langley, A.: Deprecating Secure Sockets Layer Version 3.0. RFC 7568 (Proposed Standard) (2015). DOI 10.17487/RFC7568. URL https://www.rfc-editor.org/rfc/rfc7568.txt
56. Barthe, G.: High-assurance cryptography: Cryptographic software we can trust. IEEE Security & Privacy **13**(5), 86–89 (2015). DOI 10.1109/MSP.2015.112
57. Barthe, G., Crespo, J.M., Lakhnech, Y., Schmidt, B.: Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In: E. Oswald, M. Fischlin (eds.) Advances in Cryptology - EUROCRYPT 2015, *Lecture Notes in Computer Science*, vol. 9057, pp. 689–718. Springer (2015)
58. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.: EasyCrypt: A tutorial. In: Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures, pp. 146–166 (2013)
59. Basin, D., Cremers, C., Horvat, M.: Actor key compromise: Consequences and countermeasures. In: 27th IEEE Computer Security Foundations Symposium, CSF 2014, pp. 244–258. IEEE Computer Society (2014)
60. Basin, D., Cremers, C., Meadows, C.: Model checking security protocols. In: E. Clarke, T. Henzinger, H. Veith, R. Bloem (eds.) Handbook of Model Checking. Springer (2017)
61. Basin, D.A., Cremers, C., Meier, S.: Provably repairing the ISO/IEC 9798 standard for entity authentication. Journal of Computer Security **21**(6), 817–846 (2013). DOI 10.3233/JCS-130472
62. Basin, D.A., Cremers, C.J.F.: Degrees of security: Protocol guarantees in the face of compromising adversaries. In: A. Dawar, H. Veith (eds.) Computer Science Logic, 24th International Workshop, CSL 2010, *Lecture Notes in Computer Science*, vol. 6247, pp. 1–18. Springer (2010). DOI 10.1007/978-3-642-15205-4_1
63. Basin, D.A., Cremers, C.J.F.: Modeling and analyzing security in the presence of compromising adversaries. In: D. Gritzalis, et al. (eds.) 15th European Symposium on Research in Computer Security, ESORICS 2010, *Lecture Notes in Computer Science*, vol. 6345, pp. 340–356. Springer (2010). DOI 10.1007/978-3-642-15497-3_21
64. Basin, D.A., Cremers, C.J.F., Meier, S.: Provably repairing the ISO/IEC 9798 standard for entity authentication. In: P. Degano, J.D. Guttman (eds.) Principles of Security and Trust - First International Conference, POST 2012, *Lecture Notes in Computer Science*, vol. 7215, pp. 129–148. Springer (2012). DOI 10.1007/978-3-642-28641-4_8
65. Bauer, R.K., Berson, T.A., Feiertag, R.J.: A key distribution protocol using event markers. ACM Transactions on Computer Systems **1**(3), 249–255 (1983)
66. Becker, K., Wille, U.: Communication complexity of group key distribution. In: 5th ACM Conference on Computer and Communications Security, pp. 1–6. ACM Press (1998)
67. Be'ery, T., Shulman, A.: A perfect CRIME? only TIME will tell. In: Black Hat Europe 2013 (2013). URL https://www.blackhat.com/eu-13/briefings.html#Beery
68. Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: C. Dwork (ed.) Advances in Cryptology - CRYPTO 2006, *Lecture Notes in Computer Science*, vol. 4117. Springer (2006)
69. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: C. Boyd (ed.) Advances in Cryptology - ASIACRYPT 2001, *Lecture Notes in Computer Science*, vol. 2248, pp. 566–582. Springer (2001). DOI 10.1007/3-540-45682-1_33
70. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: B. Preneel (ed.) Advances in Cryptology – EUROCRYPT 2000, *Lecture Notes in Computer Science*, vol. 1807, pp. 259–274. Springer

(2000). Full version at http://www-cse.ucsd.edu/users/mihir/papers/musu.html
71. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: N. Koblitz (ed.) Advances in Cryptology – CRYPTO '96, *Lecture Notes in Computer Science*, vol. 1109, pp. 1–15. Springer (1996)
72. Bellare, M., Canetti, R., Krawczyk, H.: A modular approach to the design and analysis of authentication and key exchange protocols. In: 30th ACM Symposium on Theory of Computing, pp. 419–428. ACM Press (1998). Full version at http://cseweb.ucsd.edu/~mihir/papers/modular.pdf
73. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: H. Krawczyk (ed.) Advances in Cryptology – CRYPTO '98, pp. 26–45. Springer (1998). Lecture Notes in Computer Science Volume 1462
74. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: B. Preneel (ed.) Advances in Cryptology – EUROCRYPT 2000, *Lecture Notes in Computer Science*, vol. 1807, pp. 139–155. Springer (2000)
75. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: D.R. Stinson (ed.) Advances in Cryptology – Crypto '93, *Lecture Notes in Computer Science*, vol. 773, pp. 232–249. Springer (1993). URL http://cseweb.ucsd.edu/~mihir/papers/eakd.pdf
76. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: 1st ACM Conference on Computer and Communications Security, pp. 62–73. ACM Press (1993)
77. Bellare, M., Rogaway, P.: Optimal asymmetric encryption - how to encrypt with RSA. In: A.D. Santis (ed.) Advances in Cryptology – EUROCRYPT '94, *Lecture Notes in Computer Science*, vol. 950, pp. 92–111. Springer (1995)
78. Bellare, M., Rogaway, P.: Provably secure session key distribution – the three party case. In: 27th ACM Symposium on Theory of Computing, pp. 57–66. ACM Press (1995)
79. Bellare, M., Rogaway, P.: The AuthA protocol for password-based authenticated key exchange (2000). URL http://www.cs.ucdavis.edu/~rogaway/papers/autha.pdf
80. Beller, M.J., Chang, L.F., Yacobi, Y.: Privacy and authentication on a portable communications system. In: GLOBECOM'91, pp. 1922–1927. IEEE Press (1991)
81. Beller, M.J., Chang, L.F., Yacobi, Y.: Security for personal communication services: Public-key vs. private key approaches. In: 3rd IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'92), pp. 26–31. IEEE Press (1992)
82. Beller, M.J., Chang, L.F., Yacobi, Y.: Privacy and authentication on a portable communications system. IEEE Journal on Selected Areas in Communications **11**(6), 821–829 (1993)
83. Beller, M.J., Yacobi, Y.: Fully-fledged two-way public key authentication and key agreement for low-cost terminals. Electronics Letters **29**(11), 999–1001 (1993)
84. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: IEEE Symposium on Research in Security and Privacy, pp. 72–84. IEEE Computer Society Press (1992)
85. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In: 1st ACM Conference on Computer and Communications Security, pp. 244–250. ACM Press (1993)

86. Belshe, M., Peon, R.: SPDY Protocol. The Internet Society (2012). Internet-Draft
87. Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions (2013). URL https://eprint.iacr.org/2013/833. Cryptology ePrint Archive, Report 2013/833
88. Berbecaru, D., Lioy, A.: On the robustness of applications based on the SSL and TLS security protocols. In: J. Lopez, P. Samarati, J.L. Ferrer (eds.) 4th European PKI Workshop (EUROPKI), *Lecture Notes in Computer Science*, vol. 4582, pp. 248–264. Springer (2007)
89. Bergsma, F., Dowling, B., Kohlar, F., Schwenk, J., Stebila, D.: Multi-ciphersuite security of the Secure Shell (SSH) protocol. In: M. Yung, N. Li (eds.) Proc. 21st ACM Conference on Computer and Communications Security (CCS) 2014. ACM (2014)
90. Bergsma, F., Jager, T., Schwenk, J.: One-round key exchange with strong security: An efficient and generic construction in the standard model. In: J. Katz (ed.) Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, *Lecture Notes in Computer Science*, vol. 9020, pp. 477–494. Springer (2015). DOI 10.1007/978-3-662-46447-2_21
91. Berkovits, S.: How to broadcast a secret. In: D.W. Davies (ed.) Advances in Cryptology – EUROCRYPT '91, *Lecture Notes in Computer Science*, vol. 547, pp. 535–541. Springer (1991)
92. Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P., Zinzindohoue, J.K.: A messy state of the union: Taming the composite state machines of TLS. In: 36th IEEE Symposium on Security and Privacy, pp. 535–552. IEEE Computer Society (2015)
93. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: 2017 IEEE Symposium on Security and Privacy, pp. 483–502. IEEE Computer Society (2017). DOI 10.1109/SP.2017.26
94. Bhargavan, K., Brzuska, C., Fournet, C., Green, M., Kohlweiss, M., Zanella-Béguelin, S.: Downgrade resilience in key-exchange protocols. In: 37th IEEE Symposium on Security and Privacy, pp. 506–525. IEEE Computer Society (2016)
95. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pan, J., Protzenko, J., Rastogi, A., Swamy, N., Béguelin, S.Z., Zinzindohoue, J.K.: Implementing and proving the TLS 1.3 record layer. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, pp. 463–482. IEEE Computer Society (2017). DOI 10.1109/SP.2017.58
96. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: 35th IEEE Symposium on Security and Privacy. IEEE Computer Society (2014). URL https://www.mitls.org/pages/attacks/3SHAKE
97. Bhargavan, K., Fournet, C., Corin, R., Zălinescu, E.: Verified cryptographic implementations for TLS. ACM Transactions on Information and System Security (TISSEC) **15**(1), 3 (2012)
98. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y.: Implementing TLS with verified cryptographic security. In: IEEE Symposium on Security & Privacy, pp. 445–459. IEEE (2013). DOI 10.1109/SP.2013.37. URL https://www.mitls.org/pages/publications
99. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zanella-Béguelin, S.: Proving the TLS handshake secure (as it is). In: Advances in Cryptology – CRYPTO 2014, Lecture Notes in Computer Science, pp. 235–255. Springer (2014)
100. Bhargavan, K., Leurent, G.: On the practical (in-)security of 64-bit block ciphers – collision attacks on HTTP over TLS and OpenVPN. In: 23nd ACM SIGSAC Conference on Computer and Communications Security. ACM (2016)

101. Bhargavan, K., Leurent, G.: Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In: Network and Distributed System Security Symposium. Internet Society (2016). URL https://www.mitls.org/downloads/transcript-collisions.pdf
102. Bhargavan (Ed.), K., Delignat-Lavaud, A., Pironti, A., Langley, A., Ray, M.: Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension. RFC 7627 (Proposed Standard) (2015). DOI 10.17487/RFC7627. URL https://www.rfc-editor.org/rfc/rfc7627.txt
103. Bird, R., Gopal, I., Herzberg, A., Janson, P.A., Kutten, S., Molva, R., Yung, M.: Systematic design of a family of attack-resistant authentication protocols. IEEE Journal on Selected Areas in Communications **11**(5), 679–693 (1993)
104. Birkett, J., Stebila, D.: Predicate-based key exchange. In: R. Steinfeld, P. Hawkes (eds.) Information Security and Privacy, ACISP 2010, *Lecture Notes in Computer Science*, vol. 6168, pp. 282–299. Springer (2010)
105. Blake, I.F., Seroussi, G., Smart, N.P.: Elliptic Curves in Cryptography. Cambridge University Press (1999)
106. Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Moeller, B.: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational) (2006). DOI 10.17487/RFC4492. URL https://www.rfc-editor.org/rfc/rfc4492.txt
107. Blake-Wilson, S., Johnson, D., Menezes, A.: Key agreement protocols and their security analysis. In: M. Darnell (ed.) Crypography and Coding - 6th IMA Conference, *Lecture Notes in Computer Science*, vol. 1355, pp. 30–45. Springer-Verlag (1997). URL http://www.math.uwaterloo.ca/~ajmeneze/publications/agreement.ps
108. Blake-Wilson, S., Menezes, A.: Entity authentication and authenticated key transport protocols employing asymmetric techniques. In: B. Christianson, et al. (eds.) Security Protocols – 5th International Workshop, *Lecture Notes in Computer Science*, vol. 1361, pp. 137–158. Springer (1998)
109. Blake-Wilson, S., Menezes, A.: Authenticated Diffie-Hellman key agreement protocols. In: S. Tavares, et al. (eds.) Selected Areas in Cryptography, 5th International Workshop, *Lecture Notes in Computer Science*, vol. 1556, pp. 339–361. Springer (1999)
110. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the Station-to-Station (STS) protocol. In: H. Imai, et al. (eds.) Public Key Cryptography, *Lecture Notes in Computer Science*, vol. 1560, pp. 154–170. Springer (1999)
111. Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T.: Transport Layer Security (TLS) Extensions. RFC 3546 (Proposed Standard) (2003). DOI 10.17487/RFC3546. URL https://www.rfc-editor.org/rfc/rfc3546.txt
112. Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., Wright, T.: Transport Layer Security (TLS) Extensions. RFC 4366 (Proposed Standard) (2006). DOI 10.17487/RFC4366. URL https://www.rfc-editor.org/rfc/rfc4366.txt
113. Blanchet, B.: A computationally sound mechanized prover for security protocols. In: 2006 IEEE Symposium on Security and Privacy (S&P 2006), pp. 140–154. IEEE Computer Society (2006). DOI 10.1109/SP.2006.1
114. Blanchet, B.: Mechanizing game-based proofs of security protocols. In: T. Nipkow, et al. (eds.) Software Safety and Security - Tools for Analysis and Verification, *NATO Science for Peace and Security Series - D: Information and Communication Security*, vol. 33, pp. 1–25. IOS Press (2012). URL http://prosecco.gforge.inria.fr/personal/bblanche/publications/BlanchetMOD11.pdf

115. Blanchet, B.: Modeling and verifying security protocols with the applied Pi calculus and ProVerif. Foundations and Trends in Privacy and Security **1**(1-2), 1–135 (2016). DOI 10.1561/3300000004
116. Blanchet, B., Jaggard, A.D., Scedrov, A., Tsay, J.: Computationally sound mechanized proofs for basic and public-key Kerberos. In: M. Abe, V.D. Gligor (eds.) Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, pp. 87–99. ACM (2008). DOI 10.1145/1368310.1368326
117. Blanchet, B., Pointcheval, D.: Automated security proofs with sequences of games. In: C. Dwork (ed.) Advances in Cryptology - CRYPTO 2006, *Lecture Notes in Computer Science*, vol. 4117, pp. 537–554. Springer (2006)
118. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: H. Krawczyk (ed.) Advances in Cryptology – Proc. CRYPTO '98, *Lecture Notes in Computer Science*, vol. 1462, pp. 1–12. Springer (1998). DOI 10.1007/BFb0055716
119. Bleichenbacher, D.: Personal Communication (2001)
120. Bohli, J., Steinwandt, R.: Deniable group key agreement. In: Progress in Cryptology-VIETCRYPT 2006, pp. 298–311. Springer (2006)
121. Bohli, J., Vasco, M.I.G., Steinwandt, R.: Secure group key establishment revisited. Int. J. Inf. Sec. **6**(4), 243–254 (2007)
122. Bohli, J.M., Vasco, M.I.G., Steinwandt, R.: Password-authenticated constant-round group key establishment with a common reference string (2006). URL https://eprint.iacr.org/2006/214. Cryptology ePrint Archive, Report 2006/214
123. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. SIAM Journal of Computing **32**(3), 585–615 (2003)
124. Boneh, D., Sahai, A., Waters, B.: Functional encryption: a new vision for public-key cryptography. Communications of the ACM **55**(11), 56–64 (2012)
125. Boneh, D., Silverberg, A.: Applications of multilinear forms to cryptography. IACR Cryptology ePrint Archive (2002). URL https://eprint.iacr.org/2002/080
126. Bonneau, J.: The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: 33rd IEEE Symposium on Security and Privacy. IEEE Computer Society (2012)
127. Boyarsky, M.K.: Public-key cryptography and password protocols: The multi-user case. In: 6th ACM Conference on Computer and Communications Security, pp. 63–72. ACM Press (1999)
128. Boyd, C.: Hidden assumptions in cryptographic protocols. IEE Proceedings - Computers and Digital Techniques **137**(6), 433–436 (1990)
129. Boyd, C.: Security architectures using formal methods. IEEE Journal on Selected Areas in Communications **11**(5), 694–701 (1993)
130. Boyd, C.: Towards a classification of key agreement protocols. In: 8th IEEE Computer Security Foundations Workshop, pp. 38–43. IEEE Computer Society Press (1995)
131. Boyd, C.: A class of flexible and efficient key management protocols. In: 9th IEEE Computer Security Foundations Workshop, pp. 2–8. IEEE Computer Society Press (1996)
132. Boyd, C.: On key agreement and conference key agreement. In: V. Varadharajan, et al. (eds.) Security and Privacy – Proceedings of ACISP'97, *Lecture Notes in Computer Science*, vol. 1270, pp. 294–302. Springer (1997)
133. Boyd, C., Choo, K.K.R.: Security of two-party identity-based key agreement. In: E. Dawson, S. Vaudenay (eds.) Progress in Cryptology - Mycrypt 2005, *Lecture Notes in Computer Science*, vol. 3715, pp. 229–243. Springer (2005)

134. Boyd, C., Cliff, Y., Nieto, J.M.G., Paterson, K.G.: One-round key exchange in the standard model. IJACT **1**(3), 181–199 (2009). DOI 10.1504/IJACT.2009.023466
135. Boyd, C., Cremers, C., Feltz, M., Paterson, K.G., Poettering, B., Stebila, D.: ASICS: authenticated key exchange security incorporating certification systems. In: J. Crampton, et al. (eds.) 18th European Symposium on Research in Computer Security, ESORICS 2013, *Lecture Notes in Computer Science*, vol. 8134, pp. 381–399. Springer (2013)
136. Boyd, C., Cremers, C., Feltz, M., Paterson, K.G., Poettering, B., Stebila, D.: ASICS: authenticated key exchange security incorporating certification systems. Int. J. Inf. Sec. **16**(2), 151–171 (2017). DOI 10.1007/s10207-015-0312-y
137. Boyd, C., Mao, W.: Limitations of logical analysis of cryptographic protocols. In: Preproceedings – EUROCRYPT '93 (1993)
138. Boyd, C., Mao, W.: On a limitation of BAN logic. In: T. Helleseth (ed.) Advances in Cryptology – EUROCRYPT '93, *Lecture Notes in Computer Science*, vol. 765, pp. 240–247. Springer (1994)
139. Boyd, C., Mao, W., Paterson, K.G.: Key agreement using statically keyed authenticators. In: M. Jakobsson, et al. (eds.) Applied Cryptography and Network Security, ACNS 2004, *Lecture Notes in Computer Science*, vol. 3089, pp. 248–262. Springer (2004)
140. Boyd, C., Mathuria, A.: Systematic design of key establishment protocols based on one-way functions. IEE Proceedings - Computers and Digital Techniques **144**(2), 93–99 (1997)
141. Boyd, C., Mathuria, A.: Key establishment protocols for secure mobile communications: A critical survey. Computer Communications **23**, 575–587 (2000)
142. Boyd, C., Nieto, J.G.: On forward secrecy in one-round key exchange. In: L. Chen (ed.) Cryptography and Coding - 13th IMA International Conference, *Lecture Notes in Computer Science*, vol. 7089, pp. 451–468. Springer (2011). DOI 10.1007/978-3-642-25516-8_27
143. Boyd, C., Nieto, J.M.G.: Round-optimal contributory conference key agreement. In: Y. Desmedt (ed.) Public Key Cryptography – PKC 2003, *Lecture Notes in Computer Science*, vol. 2567, pp. 161–174. Springer (2003)
144. Boyen, X.: The uber-assumption family. In: S.D. Galbraith, K.G. Paterson (eds.) Pairing-Based Cryptography - Pairing 2008, *Lecture Notes in Computer Science*, vol. 5209, pp. 39–56. Springer (2008)
145. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: B. Preneel (ed.) Advances in Cryptology – EUROCRYPT 2000, *Lecture Notes in Computer Science*, vol. 1807, pp. 156–171. Springer (2000)
146. Brainard, J., Juels, A., Kaliski Jr., B.S., Szydlo, M.: A new two-server approach for authentication with short secrets. In: Proceedings of the 12th USENIX Workshop on Security, pp. 201–213. USENIX Association (2003)
147. Brandt, J., Damgård, I., Landrock, P., Pedersen, T.: Zero knowledge authentication scheme with secret key exchange. In: S. Goldwasser (ed.) Advances in Cryptology – Crypto '88, *Lecture Notes in Computer Science*, vol. 403, pp. 583–588. Springer (1989)
148. Bresson, E., Chevassut, O., Pointcheval, D.: Provably authenticated group Diffie-Hellman key exchange – the dynamic case. In: C. Boyd (ed.) Advances in Cryptology – ASIACRYPT 2001, *Lecture Notes in Computer Science*, vol. 2248, pp. 290–309. Springer (2001)
149. Bresson, E., Chevassut, O., Pointcheval, D.: Dynamic group Diffie-Hellman key exchange under standard assumptions. In: L. Knudsen (ed.) Advances in Cryptology – EUROCRYPT 2002, *Lecture Notes in Computer Science*, vol. 2332, pp. 321–336. Springer (2002)

150. Bresson, E., Chevassut, O., Pointcheval, D.: Group Diffie–Hellman key exchange secure against dictionary attacks. In: Y. Zheng (ed.) Advances in Cryptology - ASIACRYPT 2002, *Lecture Notes in Computer Science*, vol. 2501, pp. 497–514. Springer (2002). DOI 10.1007/3-540-36178-2_31
151. Bresson, E., Chevassut, O., Pointcheval, D.: The group Diffie–Hellman problems. In: K. Nyberg, H.M. Heys (eds.) Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, *Lecture Notes in Computer Science*, vol. 2595, pp. 325–338. Springer (2002)
152. Bresson, E., Chevassut, O., Pointcheval, D.: New security results on encrypted key exchange. In: F. Bao, et al. (eds.) Public Key Cryptography - PKC 2004, *Lecture Notes in Computer Science*, vol. 2947, pp. 145–158. Springer (2004). DOI 10.1007/978-3-540-24632-9_11
153. Bresson, E., Chevassut, O., Pointcheval, D.: Provably secure authenticated group Diffie–Hellman key exchange. ACM Trans. Inf. Syst. Secur. **10**(3) (2007)
154. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.J.: Provably authenticated group Diffie-Hellman key exchange. In: 8th ACM Conference on Computer and Communications Security, pp. 255–264. ACM Press (2001)
155. Bresson, E., Manulis, M.: Contributory group key exchange in the presence of malicious participants. IET Information Security **2**(3), 85–93 (2008)
156. Bresson, E., Manulis, M.: Securing group key exchange against strong corruptions. In: M. Abe, V.D. Gligor (eds.) Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, pp. 249–260. ACM (2008)
157. Bresson, E., Manulis, M., Schwenk, J.: On security models and compilers for group key exchange protocols. In: A. Miyaji, et al. (eds.) Advances in Information and Computer Security, Second International Workshop on Security, IWSEC 2007, *Lecture Notes in Computer Science*, vol. 4752, pp. 292–307. Springer (2007)
158. Brouwer, A.E., Pellikaan, R., Verheul, E.R.: Doing more with fewer bits. In: K.Y. Lam, et al. (eds.) Advances in Cryptology – ASIACRYPT '99, *Lecture Notes in Computer Science*, vol. 1716, pp. 321–332. Springer (1999)
159. Brown, D., Menezes, A.: A small subgroup attack on Arazi's key agreement protocol. Bulletin of the ICA **37**, 45–50 (2003)
160. Brubaker, C., Jana, S., Ray, B., Khurshid, S., Shmatikov, V.: Using Frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In: 35th IEEE Symposium on Security and Privacy, pp. 114–129. IEEE Computer Society (2014)
161. Brumley, B.B., Barbosa, M., Page, D., Vercauteren, F.: Practical realisation and elimination of an ECC-related software bug attack. In: O. Dunkelman (ed.) Topics in Cryptology – CT-RSA 2012, *Lecture Notes in Computer Science*, vol. 7178, pp. 171–186. Springer (2012). DOI 10.1007/978-3-642-27954-6_11
162. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: V. Atluri, C. Diaz (eds.) 16th European Symposium on Research in Computer Security, ESORICS 2011, *Lecture Notes in Computer Science*, vol. 6879, pp. 355–371. Springer (2011). DOI 10.1007/978-3-642-23822-2_20
163. Brusilovsky, A., Faynberg, I., Zeltsan, Z., Patel, S.: Password-Authenticated Key (PAK) Diffie-Hellman Exchange. RFC 5683 (Informational) (2010). DOI 10.17487/RFC5683. URL https://www.rfc-editor.org/rfc/rfc5683.txt
164. Brzuska, C., Fischlin, M., Smart, N.P., Warinschi, B., Williams, S.C.: Less is more: Relaxed yet composable security notions for key exchange. International Journal of Information Security **12**(4), 267–297 (2013). DOI 10.1007/s10207-013-0192-y

165. Brzuska, C., Smart, N.P., Warinschi, B., Watson, G.J.: An analysis of the EMV channel establishment protocol. Cryptology ePrint Archive, Report 2013/031 (2013). URL https://eprint.iacr.org/2013/031
166. Buchholtz, M.: Automated analysis of infinite scenarios. In: R. De Nicola, D. Sangiorgi (eds.) Trustworthy Global Computing - International Symposium, TGC 2005, *Lecture Notes in Computer Science*, vol. 3705, pp. 334–352. Springer (2005)
167. Burmester, M.: On the risk of opening distributed keys. In: Y. Desmedt (ed.) Advances in Cryptology – CRYPTO '94, *Lecture Notes in Computer Science*, vol. 839, pp. 308–317. Springer (1994)
168. Burmester, M., Desmedt, Y.: A secure and efficient conference key distribution system. In: A.D. Santis (ed.) Advances in Cryptology – EUROCRYPT '94, *Lecture Notes in Computer Science*, vol. 950, pp. 275–286. Springer (1995)
169. Burmester, M., Desmedt, Y.: Efficient and secure conference-key distribution. In: T.M.A. Lomas (ed.) Security Protocols, *Lecture Notes in Computer Science*, vol. 1189, pp. 119–129. Springer (1996)
170. Burmester, M., Desmedt, Y.: A secure and scalable group key exchange system. Inf. Process. Lett. **94**(3), 137–143 (2005)
171. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. Proceedings of the Royal Society of London **A426**, 233–271 (1989)
172. Byun, J.W., Lee, D.H.: N-party encrypted Diffie–Hellman key exchange using different passwords. In: J. Ioannidis, et al. (eds.) Applied Cryptography and Network Security, Third International Conference, ACNS 2005, *Lecture Notes in Computer Science*, vol. 3531, pp. 75–90 (2005). DOI 10.1007/11496137_6
173. Cadé, D., Blanchet, B.: From computationally-proved protocol specifications to implementations and application to SSH. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) **4**(1), 4–31 (2013). URL http://isyou.info/jowua/papers/jowua-v4n1-1.pdf
174. Camenisch, J., Casati, N., Groß, T., Shoup, V.: Credential authenticated identification and key exchange. In: T. Rabin (ed.) Advances in Cryptology - CRYPTO 2010, *Lecture Notes in Computer Science*, vol. 6223, pp. 255–276. Springer (2010)
175. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Advances in Cryptology – EUROCRYPT '99, *Lecture Notes in Computer Science*, vol. 1592, pp. 107–122. Springer (1999)
176. Canetti, R., Dwork, C., Naor, M., Ostrovsky, R.: Deniable encryption. In: B.S. Kaliski Jr. (ed.) Advances in Cryptology - CRYPTO '97, *Lecture Notes in Computer Science*, vol. 1294, pp. 90–104. Springer (1997)
177. Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast security: A taxonomy and some efficient constructions. In: INFOCOM'99, vol. 2, pp. 708–716. IEEE Press (1999)
178. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: B. Pfitzmann (ed.) Advances in Cryptology – EUROCRYPT 2001, *Lecture Notes in Computer Science*, vol. 2045, pp. 453–474. Springer (2001). URL https://eprint.iacr.org/2001/040
179. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. Cryptology ePrint Archive, Report 2002/120 (2002). URL https://eprint.iacr.org/2002/120
180. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: M. Yung (ed.) Advances in Cryptology - CRYPTO 2002, *Lecture Notes in Computer Science*, vol. 2442, pp. 143–161. Springer (2002). DOI 10.1007/3-540-45708-9_10

181. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: L.R. Knudsen (ed.) Advances in Cryptology - EUROCRYPT 2002, *Lecture Notes in Computer Science*, vol. 2332, pp. 337–351. Springer (2002)
182. Carlsen, U.: Cryptographic protocol flaws - know your enemy. In: 7th IEEE Computer Security Foundations Workshop, pp. 192–200. IEEE Computer Society Press (1994)
183. Carlsen, U.: Optimal privacy and authentication on a portable communications system. ACM Operating Systems Review **28**(3), 16–23 (1994)
184. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: N.P. Smart (ed.) Advances in Cryptology - EUROCRYPT 2008, *Lecture Notes in Computer Science*, vol. 4965, pp. 127–145. Springer (2008)
185. Cervesato, I., Jaggard, A., Scedrov, A., Tsay, J.K., Walstad, C.: Breaking and fixing public-key Kerberos. In: M. Okada, I. Satoh (eds.) Advances in Computer Science - ASIAN 2006, *Lecture Notes in Computer Science*, vol. 4435, pp. 167–181. Springer (2008)
186. Chaki, S., Datta, A.: ASPIER: An automated framework for verifying security protocol implementations. In: Proc. 22nd IEEE Computer Security Foundations Symposium (CSF) 2009, pp. 172–185 (2009)
187. Chalkias, K., Halkidis, S.T., Hristu-Varsakelis, D., Stephanides, G., Alexiadis, A.: A provably secure one-pass two-party key establishment protocol. In: D. Pei, et al. (eds.) Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, *Lecture Notes in Computer Science*, vol. 4990, pp. 108–122. Springer (2007)
188. Chatterjee, S., Menezes, A., Ustaoglu, B.: Combined security analysis of the one- and three-pass unified model key agreement protocols. In: G. Gong, K.C. Gupta (eds.) Progress in Cryptology - INDOCRYPT 2010, *Lecture Notes in Computer Science*, vol. 6498, pp. 49–68. Springer (2010)
189. Chatterjee, S., Menezes, A., Ustaoglu, B.: A generic variant of NIST's KAS2 key agreement protocol. In: ACISP, *Lecture Notes in Computer Science*, vol. 6812, pp. 353–370. Springer (2011)
190. Chaum, D., van Heyst, E.: Group signatures. In: D.W. Davies (ed.) Advances in Cryptology - EUROCRYPT '91, *Lecture Notes in Computer Science*, vol. 547, pp. 257–265. Springer (1991). DOI 10.1007/3-540-46416-6_22
191. Chen, J.L., Hwang, T.: Identity-based conference key broadcast schemes with user authentication. Computers and Security **13**, 53–57 (1994)
192. Chen, L., Cheng, Z., Smart, N.P.: Identity-based key agreement protocols from pairings. International Journal of Information Security **6**, 213–241 (2007)
193. Chen, L., Gollmann, D., Mitchell, C.J.: Key distribution without individual trusted authentication servers. In: 8th IEEE Computer Security Foundations Workshop, pp. 30–36. IEEE Computer Society Press (1995)
194. Chen, L., Kudla, C.: Identity based authenticated key agreement protocols from pairings. In: 16th IEEE Computer Security Foundations Workshop - CSFW 2003, pp. 219–233. IEEE Computer Society Press (2003)
195. Chen, L., Kudla, C.: Identity based authenticated key agreement protocols from pairings. Cryptology ePrint Archive, Report 2002/184 (2004). URL https://eprint.iacr.org/2002/184
196. Chen, L., Lim, H.W., Yang, G.: Cross-domain password-based authenticated key exchange revisited. ACM Trans. Inf. Syst. Secur. **16**(4), 15:1–15:32 (2014). DOI 10.1145/2584681
197. Chen, L., Mitchell, C.J.: Parsing ambiguities in authentication and key establishment protocols. IJESDF **3**(1), 82–94 (2010)

198. Cheng, Q., Ma, C.: Ephemeral key compromise attack on the IB-KA protocol. Cryptology ePrint Archive, Report 2009/568 (2009). URL https://eprint.iacr.org/2009/568
199. Cheng, Z., Chen, L.: On security proof of McCullagh–Barreto's key agreement protocol and its variants. IJSN **2**(3/4), 251–259 (2007)
200. Cheng, Z., Comley, R.: Attacks on an ISO/IEC 11770-2 key establishment protocol. I. J. Network Security **3**(3), 290–295 (2006)
201. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. In: E. Oswald, M. Fischlin (eds.) Advances in Cryptology - EUROCRYPT 2015, Part I, *Lecture Notes in Computer Science*, vol. 9056, pp. 3–12. Springer (2015)
202. Chevalier, Y., Vigneron, L.: Automated unbounded verification of security protocols. In: E. Brinksma, K.G. Larsen (eds.) Computer Aided Verification - 14th International Conference, CAV 2002, *Lecture Notes in Computer Science*, vol. 2404, pp. 324–337. Springer (2002)
203. Chikazawa, T., Inoue, T.: A new key sharing system for global telecommunications. In: GLOBECOM '90, pp. 1069–1072. IEEE Press (1990)
204. Chikazawa, T., Yamagishi, A.: An improved identity-based one-way conference key sharing system. In: Proceedings of ICCS/ISITA '92, pp. 270–273. IEEE Press, Singapore (1992)
205. Choi, K.Y., Hwang, J.Y., Lee, D.H.: Efficient ID-based group key agreement with bilinear maps. In: F. Bao, et al. (eds.) Public Key Cryptography - PKC 2004, *Lecture Notes in Computer Science*, vol. 2947, pp. 130–144. Springer (2004)
206. Choi, K.Y., Hwang, J.Y., Lee, D.H., Seo, I.S.: ID-based authenticated key agreement for low-power mobile devices. In: C. Boyd, J.M.G. Nieto (eds.) Information Security and Privacy, 10th Australasian Conference, ACISP 2005, *Lecture Notes in Computer Science*, vol. 3574, pp. 494–505. Springer (2005)
207. Choie, Y.J., Jeong, E., Lee, E.: Efficient identity-based authenticated key agreement protocol from pairings. Applied Mathematics and Computation pp. 179–188 (2005)
208. Choo, K.K.R., Boyd, C., Hitchcock, Y.: Examining indistinguishability-based proof models for key establishment protocols. In: B.K. Roy (ed.) Advances in Cryptology - ASIACRYPT 2005, *Lecture Notes in Computer Science*, vol. 3788, pp. 585–604. Springer (2005)
209. Choo, K.K.R., Boyd, C., Hitchcock, Y., Maitland, G.: On session identifiers in provably secure protocols: The Bellare-Rogaway three-party key distribution protocol revisited. In: C. Blundo, S. Cimato (eds.) Security in Communication Networks, SCN 2004, *Lecture Notes in Computer Science*, vol. 3352, pp. 351–366. Springer (2005)
210. Choo, K.R.: Revisit of McCullagh-Barreto two-party ID-based authenticated key agreement protocols. I. J. Network Security **1**(3), 154–160 (2005)
211. Choo, K.R., Boyd, C., Hitchcock, Y.: Errors in computational complexity proofs for protocols. In: B.K. Roy (ed.) Advances in Cryptology - ASIACRYPT 2005, *Lecture Notes in Computer Science*, vol. 3788, pp. 624–643. Springer (2005)
212. Chow, S.S.M., Choo, K.K.R.: Strongly-secure identity-based key agreement and anonymous extension. In: Information Security, pp. 203–220. Springer (2007). URL https://eprint.iacr.org/2007/018
213. Chown, P.: Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS). RFC 3268 (Proposed Standard) (2002). DOI 10.17487/RFC3268. URL https://www.rfc-editor.org/rfc/rfc3268.txt

214. Christianson, B., Roe, M., Wheeler, D.: Secure sessions from weak secrets. In: B. Christianson, et al. (eds.) Security Protocols, 11th International Workshop, *Lecture Notes in Computer Science*, vol. 3364, pp. 190–205. Springer (2003). DOI 10.1007/11542322_24
215. Clark, J., Jacob, J.: On the security of recent protocols. Information Processing Letters **56**(3), 151–155 (1995)
216. Clark, J., Jacob, J.: Attacking authentication protocols. High Integrity Systems **1**(5), 465–473 (1996)
217. Clark, J., Jacob, J.: A survey of authentication protocol literature: Version 1.0 (1997). URL https://www-users.cs.york.ac.uk/~jac/PublishedPapers/reviewV1_1997.pdf
218. Clark, J., van Oorschot, P.C.: SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In: 34th IEEE Symposium on Security and Privacy, pp. 511–525. IEEE Computer Society (2013)
219. Clarke, D., Hao, F.: Cryptanalysis of the Dragonfly key exchange protocol. IET Information Security **8**(6), 283–289 (2014). URL https://eprint.iacr.org/2013/058.pdf
220. Codenomicon: Heartbleed bug (2014). URL http://heartbleed.com/
221. Cohen, H., Frey, G. (eds.): Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman and Hall/CRC (2005)
222. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the Signal messaging protocol. In: Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 2017. IEEE (2017)
223. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (2008). DOI 10.17487/RFC5280. URL https://www.rfc-editor.org/rfc/rfc5280.txt
224. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: H. Krawczyk (ed.) Advances in Cryptology – CRYPTO '98, *Lecture Notes in Computer Science*, vol. 1462, pp. 13–25. Springer (1998)
225. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: L.R. Knudsen (ed.) Advances in Cryptology - EUROCRYPT 2002, *Lecture Notes in Computer Science*, vol. 2332, pp. 45–64. Springer (2002). DOI 10.1007/3-540-46035-7_4
226. Cremers, C.: Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In: B.S.N. Cheung, et al. (eds.) Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, pp. 80–91. ACM (2011)
227. Cremers, C., Feltz, M.: One-round strongly secure key exchange with perfect forward secrecy and deniability. Cryptology ePrint Archive, Report 2011/300 (2011). URL https://eprint.iacr.org/2011/300
228. Cremers, C., Horvat, M.: Improving the ISO/IEC 11770 standard for key management techniques. In: L. Chen, C.J. Mitchell (eds.) Security Standardisation Research - First International Conference, SSR 2014, *Lecture Notes in Computer Science*, vol. 8893, pp. 215–235. Springer (2014). DOI 10.1007/978-3-319-14054-4_13
229. Cremers, C., Horvat, M.: Improving the ISO/IEC 11770 standard for key management techniques. Int. J. Inf. Sec. **15**(6), 659–673 (2016). DOI 10.1007/s10207-015-0306-9
230. Cremers, C., Horvat, M., Scott, S., van der Merwe, T.: Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In: IEEE Symposium on Security and Privacy, SP 2016, pp. 470–485. IEEE Computer Society (2016). DOI 10.1109/SP.2016.35

231. Cremers, C.J.F.: The Scyther tool: Verification, falsification, and analysis of security protocols. In: A. Gupta, S. Malik (eds.) Computer Aided Verification, 20th International Conference, CAV 2008, *Lecture Notes in Computer Science*, vol. 5123, pp. 414–418. Springer (2008). DOI 10.1007/978-3-540-70545-1_38
232. Cremers, C.J.F.: Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In: M. Abdalla, et al. (eds.) Applied Cryptography and Network Security, ACNS 2009, *Lecture Notes in Computer Science*, vol. 5536, pp. 20–33 (2009)
233. Cremers, C.J.F.: Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In: V. Atluri, C. Díaz (eds.) 16th European Symposium on Research in Computer Security, ESORICS 2011, *Lecture Notes in Computer Science*, vol. 6879, pp. 315–334. Springer (2011). DOI 10.1007/978-3-642-23822-2_18
234. Cremers, C.J.F., Feltz, M.: Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In: S. Foresti, et al. (eds.) 17th European Symposium on Research in Computer Security, ESORICS 2012, *Lecture Notes in Computer Science*, vol. 7459, pp. 734–751. Springer (2012)
235. Cremers, C.J.F., Feltz, M.: Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal. Des. Codes Cryptography **74**(1), 183–218 (2015). DOI 10.1007/s10623-013-9852-1
236. Cremers, C.J.F., Lafourcade, P., Nadeau, P.: Comparing state spaces in automatic security protocol analysis. In: V. Cortier, et al. (eds.) Formal to Practical Security, *Lecture Notes in Computer Science*, vol. 5458, pp. 70–94. Springer (2009). DOI 10.1007/978-3-642-02002-5_5
237. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer (2002)
238. Dagdelen, Ö., Fischlin, M., Gagliardoni, T., Marson, G.A., Mittelbach, A., Onete, C.: A cryptographic analysis of OPACITY - (extended abstract). In: J. Crampton, et al. (eds.) 18th European Symposium on Research in Computer Security, ESORICS 2013, *Lecture Notes in Computer Science*, vol. 8134, pp. 345–362. Springer (2013)
239. Delignat-Lavaud, A., Bhargavan, K.: Virtual host confusion: Weaknesses and exploits. In: Black Hat 2014 (2014). URL https://bh.ht.vc/vhost_confusion.pdf
240. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Communications of the ACM **24**(8), 533–536 (1981)
241. Desmedt, Y., Burmester, M.: Towards practical 'proven secure' authenticated key distribution. In: 1st ACM Conference on Computer and Communications Security, pp. 228–231. ACM Press (1993)
242. Desmedt, Y., Lange, T.: Revisiting pairing based group key exchange. In: G. Tsudik (ed.) Financial Cryptography and Data Security, 12th International Conference, FC 2008, *Lecture Notes in Computer Science*, vol. 5143, pp. 53–68. Springer (2008)
243. Desmedt, Y., Lange, T., Burmester, M.: Scalable authenticated tree based group key exchange for ad-hoc groups. In: S. Dietrich, R. Dhamija (eds.) Financial Cryptography and Data Security, 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, *Lecture Notes in Computer Science*, vol. 4886, pp. 104–118. Springer (2007)
244. Desmedt, Y., Miyaji, A.: Redesigning group key exchange protocol based on bilinear pairing suitable for various environments. In: X. Lai, et al. (eds.) Information Security and Cryptology - 6th International Conference, Inscrypt 2010, *Lecture Notes in Computer Science*, vol. 6584, pp. 236–254. Springer (2010)
245. Desmedt, Y., Pieprzyk, J., Steinfeld, R., Wang, H.: A non-malleable group key exchange protocol robust against active insiders. In: S.K. Katsikas, et al. (eds.) Information Se-

curity, 9th International Conference, ISC 2006, *Lecture Notes in Computer Science*, vol. 4176, pp. 459–475. Springer (2006)
246. Deutsch, P.: DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational) (1996). DOI 10.17487/RFC1951. URL https://www.rfc-editor.org/rfc/rfc1951.txt
247. Di Raimondo, M., Gennaro, R.: Provably secure threshold password-authenticated key exchange. J. Comput. Syst. Sci. **72**(6), 978–1001 (2006). DOI 10.1016/j.jcss.2006.02.002
248. Di Raimondo, M., Gennaro, R., Krawczyk, H.: Deniable authentication and key exchange. In: A. Juels, et al. (eds.) Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, pp. 400–409. ACM (2006)
249. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard) (1999). DOI 10.17487/RFC2246. URL https://www.rfc-editor.org/rfc/rfc2246.txt
250. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard) (2006). DOI 10.17487/RFC4346. URL https://www.rfc-editor.org/rfc/rfc4346.txt
251. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (2008). DOI 10.17487/RFC5246. URL https://www.rfc-editor.org/rfc/rfc5246.txt
252. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Information Theory **22**(6), 644–654 (1976)
253. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptography **2**(2), 107–125 (1992)
254. Ding, Y., Horster, P.: Undetectable on-line password guessing attacks. ACM Operating Systems Review **29**(4), 77–86 (1995)
255. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: M. Blaze (ed.) Proceedings of the 13th USENIX Security Symposium, pp. 303–320. USENIX (2004). URL http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html
256. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Information Theory **29**(2), 198–207 (1983). DOI 10.1109/TIT.1983.1056650
257. Dowling, B., Günther, F., Fischlin, M., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: 22nd ACM Conference on Computer and Communications Security (CCS) 2015, pp. 1197–1210. ACM (2015)
258. Dowling, B., Stebila, D.: Modelling ciphersuite and version negotiation in the TLS protocol. In: E. Foo, D. Stebila (eds.) Information Security and Privacy, 20th Australasian Conference, *Lecture Notes in Computer Science*, vol. 9144, pp. 270–288. Springer (2015)
259. Dreier, J., Duménil, C., Kremer, S., Sasse, R.: Beyond subterm-convergent equational theories in automated verification of stateful protocols. In: M. Maffei, M. Ryan (eds.) Principles of Security and Trust - 6th International Conference, POST 2017, *Lecture Notes in Computer Science*, vol. 10204, pp. 117–140. Springer (2017). DOI 10.1007/978-3-662-54455-6_6
260. Duong, T.: BEAST (2011). URL http://vnhacker.blogspot.com.au/2011/09/beast.html
261. Dupont, R., Enge, A.: Practical non-interactive key distribution based on pairings. Cryptology ePrint Archive, Report 2002/136 (2002). URL https://eprint.iacr.org/2002/136

262. Durumeric, Z., Wustrow, E., Halderman, J.A.: ZMap: Fast Internet-wide scanning and its security applications. In: Proc. 22nd USENIX Security Symposium (2013). URL https://zmap.io/paper.pdf
263. Dutta, R., Barua, R.: Provably secure constant round contributory group key agreement in dynamic setting. IEEE Trans. on Information Theory **54**(5), 2007–2025 (2008)
264. Eastlake 3rd, D.: Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066 (Proposed Standard) (2011). DOI 10.17487/RFC6066. URL https://www.rfc-editor.org/rfc/rfc6066.txt
265. Electronic Frontier Foundation: Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design. O'Reilly Media (1998)
266. Electronic Frontier Foundation: The EFF SSL Observatory (2010). URL https://www.eff.org/observatory
267. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Information Theory **IT-31**(4), 469–472 (1985)
268. Eronen (Ed.), P.: DES and IDEA Cipher Suites for Transport Layer Security (TLS). RFC 5469 (Informational) (2009). DOI 10.17487/RFC5469. URL https://www.rfc-editor.org/rfc/rfc5469.txt
269. Eronen (Ed.), P., Tschofenig (Ed.), H.: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard) (2005). DOI 10.17487/RFC4279. URL https://www.rfc-editor.org/rfc/rfc4279.txt
270. Escobar, S., Kapur, D., Lynch, C., Meadows, C.A., Meseguer, J., Narendran, P., Sasse, R.: Protocol analysis in Maude-NPA using unification modulo homomorphic encryption. In: P. Schneider-Kamp, M. Hanus (eds.) Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, pp. 65–76. ACM (2011). DOI 10.1145/2003476.2003488
271. Escobar, S., Meadows, C.A., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: A. Aldini, et al. (eds.) Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures, *Lecture Notes in Computer Science*, vol. 5705, pp. 1–50. Springer (2007). DOI 10.1007/978-3-642-03829-7_1
272. Evans, C., Palmer, C., Sleevi, R.: Public Key Pinning Extension for HTTP. RFC 7469 (Proposed Standard) (2015). DOI 10.17487/RFC7469. URL https://www.rfc-editor.org/rfc/rfc7469.txt
273. Feldmeier, D.C., Karn, P.R.: UNIX password security—ten years later (invited). In: G. Brassard (ed.) Advances in Cryptology – CRYPTO '89, *Lecture Notes in Computer Science*, vol. 435, pp. 44–63. Springer (1990)
274. Ferguson, N., Schneier, B.: A cryptographic evaluation of IPsec (2000). URL https://www.schneier.com/academic/paperfiles/paper-ipsec.pdf
275. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: A.M. Odlyzko (ed.) Advances in Cryptology – Crypto '86, *Lecture Notes in Computer Science*, vol. 263, pp. 186–194. Springer (1987)
276. Finney, H.: Bleichenbacher's RSA signature forgery based on implementation error (2006). URL https://www.ietf.org/mail-archive/web/openpgp/current/msg00999.html
277. Fiore, D., Gennaro, R.: Identity-based key exchange protocols without pairings. Trans. Computational Science **10**, 42–77 (2010). DOI 10.1007/978-3-642-17499-5_3
278. Fiore, D., Gennaro, R.: Making the Diffie-Hellman protocol identity-based. In: J. Pieprzyk (ed.) Topics in Cryptology - CT-RSA 2010, *Lecture Notes in Computer Science*, vol. 5985, pp. 165–178. Springer (2010). URL https://eprint.iacr.org/2009/174

279. Fleischhacker, N., Manulis, M., Azodi, A.: A modular framework for multi-factor authentication and key exchange. In: 1st International Conference on Security Standardisation Research (SSR 2014), *Lecture Notes in Computer Science*, vol. 8893, pp. 190–214. Springer (2014)
280. Fluhrer, S.R., McGrew, D.A.: Statistical analysis of the alleged RC4 keystream generator. In: B. Schneier (ed.) Fast Software Encryption, 7th International Workshop, *Lecture Notes in Computer Science*, vol. 1978, pp. 19–30. Springer (2000)
281. Ford, W., Kaliski Jr., B.S.: Server-assisted generation of a strong secret from a password. In: 9th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2000, pp. 176–180. IEEE Press (2000)
282. Fox-IT: DigiNotar certificate authority breach "operation black tulip" (2011). URL https://www.rijksoverheid.nl/binaries/rijksoverheid/documenten/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf
283. Freier, A., Karlton, P., Kocher, P.: The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic) (2011). DOI 10.17487/RFC6101. URL https://www.rfc-editor.org/rfc/rfc6101.txt
284. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: K. Kurosawa, G. Hanaoka (eds.) Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, *Lecture Notes in Computer Science*, vol. 7778, pp. 254–271. Springer (2013). DOI 10.1007/978-3-642-36362-7_17
285. Fujioka, A., Manulis, M., Suzuki, K., Ustaoglu, B.: Sufficient condition for ephemeral key-leakage resilient tripartite key exchange. In: W. Susilo, et al. (eds.) Information Security and Privacy, ACISP 2012, *Lecture Notes in Computer Science*, vol. 7372, pp. 15–28. Springer (2012)
286. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. Des. Codes Cryptography **76**(3), 469–504 (2015). DOI 10.1007/s10623-014-9972-2
287. Fujioka, A., Suzuki, K., Yoneyama, K.: Hierarchical ID-based authenticated key exchange resilient to ephemeral key leakage. In: I. Echizen, et al. (eds.) Advances in Information and Computer Security, IWSEC 2010, *Lecture Notes in Computer Science*, vol. 6434, pp. 164–180. Springer (2010)
288. Gajek, S., Manulis, M., Pereira, O., Sadeghi, A.R., Schwenk, J.: Universally composable security analysis of TLS. In: 2nd International Conference on Provable Security (ProvSec) 2008, *Lecture Notes in Computer Science*, vol. 5324, pp. 313–327. Springer (2008). DOI 10.1007/978-3-540-88733-1_22. URL https://eprint.iacr.org/2008/251
289. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**, 3113–3121 (2008)
290. Gao, W., Neupane, K., Steinwandt, R.: Tuning a two-round group key agreement. Int. J. Inf. Sec. **13**(5), 467–476 (2014)
291. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: T. Johansson, P.Q. Nguyen (eds.) Advances in Cryptology - EUROCRYPT 2013, *Lecture Notes in Computer Science*, vol. 7881, pp. 1–17. Springer (2013)
292. Garman, C., Paterson, K.G., der Merwe, T.V.: Attacks only get better: Password recovery attacks against RC4 in TLS. In: 24th USENIX Security Symposium, pp. 113–128. USENIX Association (2015)
293. Gennaro, R.: Faster and shorter password-authenticated key exchange. In: R. Canetti (ed.) Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008,

Lecture Notes in Computer Science, vol. 4948, pp. 589–606. Springer (2008). DOI 10.1007/978-3-540-78524-8_32
294. Gennaro, R., Krawczyk, H., Rabin, T.: Okamoto–Tanaka revisited: Fully authenticated Diffie–Hellman with minimal overhead. Cryptology ePrint Archive, Report 2010/068 (2010). URL https://eprint.iacr.org/2010/068
295. Gennaro, R., Krawczyk, H., Rabin, T.: Okamoto-Tanaka revisited: Fully authenticated Diffie–Hellman with minimal overhead. In: J. Zhou, M. Yung (eds.) Applied Cryptography and Network Security, ACNS 2010, *Lecture Notes in Computer Science*, vol. 6123, pp. 309–328 (2010)
296. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: E. Biham (ed.) Advances in Cryptology - EUROCRYPT 2003, *Lecture Notes in Computer Science*, vol. 2656, pp. 524–543. Springer (2003). DOI 10.1007/3-540-39200-9_33
297. Gentry, C.: Practical identity-based encryption without random oracles. In: Advances in Cryptology – EUROCRYPT 2006, *Lecture Notes in Computer Science*, vol. 4004, pp. 445–464. Springer (2006)
298. Gentry, C., MacKenzie, P., Ramzan, Z.: PAK-Z+. Tech. rep., Submissions to IEEE P1363.2 (2005). URL http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/27952.pdf
299. Gentry, C., MacKenzie, P.D., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: C. Dwork (ed.) Advances in Cryptology - CRYPTO 2006, *Lecture Notes in Computer Science*, vol. 4117, pp. 142–159. Springer (2006). DOI 10.1007/11818175_9
300. Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: Y. Zheng (ed.) Advances in Cryptology - ASIACRYPT 2002, *Lecture Notes in Computer Science*, vol. 2501, pp. 548–566. Springer (2002)
301. George, W., Rackoff, C.: Rethinking definitions of security for session key agreement. Cryptology ePrint Archive: Report 2013/139 (2013). URL https://eprint.iacr.org/2013/139
302. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: T. Yu, G. Danezis, V.D. Gligor (eds.) 19th ACM SIGSAC Conference on Computer and Communications Security, pp. 38–49. ACM (2012)
303. Ghosh, S., Kate, A.: Post-quantum forward-secure onion routing - (future anonymity in today's budget). In: T. Malkin, et al. (eds.) Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, *Lecture Notes in Computer Science*, vol. 9092, pp. 263–286. Springer (2015). DOI 10.1007/978-3-319-28166-7_13
304. Giesen, F., Kohlar, F., Stebila, D.: On the security of TLS renegotiation. In: V. Gligor, M. Yung (eds.) Proc. 20th ACM Conference on Computer and Communications Security (CCS) 2013, pp. 387–398. ACM (2013). DOI 10.1145/2508859.2516694. URL https://eprint.iacr.org/2012/630
305. Girault, M.: Self-certified public keys. In: D.W. Davies (ed.) Advances in Cryptology – EUROCRYPT '91, *Lecture Notes in Computer Science*, vol. 547, pp. 490–497. Springer (1991)
306. Girault, M., Paillès, J.C.: An identity-based scheme providing zero-knowledge authentication and authenticated key exchange. In: European Symposium on Research in Computer Security, ESORICS 1990, pp. 173–184. AFCET, Toulouse (1990)
307. Girault, M., Poupard, G., Stern, J.: On the fly authentication and signature schemes based on groups of unknown order. J. Cryptology **19**(4), 463–487 (2006). DOI 10.1007/s00145-006-0224-0

308. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Des. Codes Cryptography **67**(2), 245–269 (2013)
309. Goldberg, I., Wagner, D.: Randomness and the Netscape browser: How secure is the World Wide Web? Dr. Dobb's Journal (1996). URL https://people.eecs.berkeley.edu/~daw/papers/ddj-netscape.html
310. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. In: J. Kilian (ed.) Advances in Cryptology - CRYPTO 2001, *Lecture Notes in Computer Science*, vol. 2139, pp. 408–432. Springer (2001). DOI 10.1007/3-540-44647-8_24
311. Gollmann, D.: What do we mean by entity authentication? In: IEEE Symposium on Security and Privacy, pp. 46–54. IEEE Computer Society Press (1996)
312. Gollmann, D.: Authentication by correspondence. IEEE Journal on Selected Areas in Communications **21**(1), 88–95 (2003)
313. Gong, L.: Using one-way functions for authentication. ACM Computer Communication Review **19**(5), 8–11 (1989)
314. Gong, L.: A security risk of depending on synchronized clocks. ACM Operating Systems Review **26**(1), 49–53 (1992)
315. Gong, L.: Increasing availability and security of an authentication service. IEEE Journal on Selected Areas in Communications **11**(5), 657–662 (1993)
316. Gong, L.: Lower bounds on messages and rounds for network authentication protocols. In: 1st ACM Conference on Computer and Communications Security, pp. 26–37. ACM Press (1993)
317. Gong, L.: Variations on the themes of message freshness and replay. In: 6th IEEE Computer Security Foundations Workshop, pp. 131–136. IEEE Computer Society Press (1993)
318. Gong, L.: Optimal authentication protocols resistant to password guessing attacks. In: 8th IEEE Computer Security Foundations Workshop, pp. 24–29. IEEE Computer Society Press (1995)
319. Gong, L., Lomas, M.A., Needham, R., Saltzer, J.H.: Protecting poorly chosen secrets from guessing attacks. IEEE Journal on Selected Areas in Communications **11**(5), 648–656 (1993)
320. Gong, L., Syverson, P.: Fail-stop protocols: An approach to designing secure protocols. In: Dependable Computing for Critical Applications 5. IEEE Computer Society (1998). URL http://www.csl.sri.com/papers/sri-csl-tr94-14/sri-csl-tr94-14.ps.gz
321. González-Burgueño, A., Santiago, S., Escobar, S., Meadows, C.A., Meseguer, J.: Analysis of the IBM CCA security API protocols in Maude-NPA. In: L. Chen, C.J. Mitchell (eds.) Security Standardisation Research - First International Conference, SSR 2014, *Lecture Notes in Computer Science*, vol. 8893, pp. 111–130. Springer (2014). DOI 10.1007/978-3-319-14054-4_8
322. González-Burgueño, A., Santiago, S., Escobar, S., Meadows, C.A., Meseguer, J.: Analysis of the PKCS#11 API using the Maude-NPA tool. In: L. Chen, S. Matsuo (eds.) Security Standardisation Research - Second International Conference, SSR 2015, *Lecture Notes in Computer Science*, vol. 9497, pp. 86–106. Springer (2015). DOI 10.1007/978-3-319-27152-1_5
323. Google: SPDY: An experimental protocol for a faster web (2009). URL http://dev.chromium.org/spdy/spdy-whitepaper
324. Gorantla, M.C., Boyd, C., Nieto, J.M.G.: On the connection between signcryption and one-pass key establishment. In: S.D. Galbraith (ed.) Cryptography and Coding, *Lecture Notes in Computer Science*, vol. 4887, pp. 277–301. Springer (2007)

325. Gorantla, M.C., Boyd, C., Nieto, J.M.G.: ID-based one-pass authenticated key establishment. In: L. Brankovic, M. Miller (eds.) Sixth Australasian Information Security Conference, AISC 2008, *CRPIT*, vol. 81, pp. 39–46. Australian Computer Society (2008)
326. Gorantla, M.C., Boyd, C., Nieto, J.M.G.: Universally composable contributory group key exchange. In: W. Li, et al. (eds.) Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, pp. 146–156. ACM (2009)
327. Gorantla, M.C., Boyd, C., Nieto, J.M.G.: Attribute-based authenticated key exchange. In: R. Steinfeld, P. Hawkes (eds.) Information Security and Privacy, ACISP 2010, *Lecture Notes in Computer Science*, vol. 6168, pp. 300–317. Springer (2010)
328. Gorantla, M.C., Boyd, C., Nieto, J.M.G.: One round group key exchange with forward security in the standard model. Cryptology ePrint Archive, Report 2010/083 (2010). URL https://eprint.iacr.org/2010/083
329. Gorantla, M.C., Boyd, C., Nieto, J.M.G., Manulis, M.: Generic one round group key exchange in the standard model. In: D. Lee, S. Hong (eds.) Information, Security and Cryptology – ICISC 2009, pp. 1–15. Springer (2010)
330. Gorantla, M.C., Boyd, C., Nieto, J.M.G., Manulis, M.: Modeling key compromise impersonation attacks on group key exchange protocols. ACM Trans. Inf. Syst. Secur. **14**(4), 28 (2011)
331. Goscinski, A., Wang, M.: Conference authentication and key distribution service in the RHODOS distributed system. In: Communications on the Move, ICCS/ISITA '92, pp. 284–289. IEEE Press, Singapore (1992)
332. Goss, K.C.: Cryptographic Method and Apparatus for Public Key Exchange with Authentication. US Patent 4,956,863 (1990)
333. Gray III, J.W.: On the Clark–Jacob version of SPLICE/AS. Inf. Process. Lett. **62**(5), 251–254 (1997)
334. Groza, B., Warinschi, B.: Cryptographic puzzles and DoS resilience, revisited. Des. Codes Cryptography **73**(1), 177–207 (2014). DOI 10.1007/s10623-013-9816-5
335. Guillou, L., Quisquater, J.J.: A practical zero knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: C.G. Günther (ed.) Advances in Cryptology – EUROCRYPT '88, *Lecture Notes in Computer Science*, vol. 330, pp. 123–128. Springer (1988)
336. Günther, C.G.: An identity-based key exchange protocol. In: J.J. Quisquater, et al. (eds.) Advances in Cryptology – EUROCRYPT '89, *Lecture Notes in Computer Science*, vol. 434, pp. 29–37. Springer (1989)
337. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-RTT key exchange with full forward secrecy. In: J. Coron, J.B. Nielsen (eds.) Advances in Cryptology - EUROCRYPT 2017, *Lecture Notes in Computer Science*, vol. 10212, pp. 519–548 (2017). DOI 10.1007/978-3-319-56617-7_18
338. Guo, Y., Zhang, Z.: Authenticated key exchange with entities from different settings and varied groups. In: Provable Security, ProveSec 2012, pp. 276–287. Springer (2012)
339. Gutmann, P.: Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7366 (Proposed Standard) (2014). DOI 10.17487/RFC7366. URL https://www.rfc-editor.org/rfc/rfc7366.txt
340. Halevi, S.: A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181 (2005). URL https://eprint.iacr.org/2005/181
341. Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. In: 5th ACM Conference on Computer and Communications Security, pp. 122–131. ACM Press (1998)

342. Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. ACM Transactions on Information and Systems Security **2**(3), 230–268 (1999)
343. Halevi, S., Krawczyk, H.: One-pass HMQV and asymmetric key-wrapping. In: D. Catalano, et al. (eds.) Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, *Lecture Notes in Computer Science*, vol. 6571, pp. 317–334. Springer (2011)
344. Hao, F.: On robust key agreement based on public key authentication. In: R. Sion (ed.) Financial Cryptography, *Lecture Notes in Computer Science*, vol. 6052, pp. 383–390. Springer (2010)
345. Hao, F.: On robust key agreement based on public key authentication. Security and Communication Networks **7**(1), 77–87 (2014)
346. Hao, F., Ryan, P.: J-PAKE: authenticated key exchange without PKI. Trans. Computational Science **11**, 192–206 (2010). DOI 10.1007/978-3-642-17697-5_10
347. Hao, F., Shahandashti, S.F.: The SPEKE protocol revisited. In: L. Chen, C.J. Mitchell (eds.) Security Standardisation Research - First International Conference, SSR 2014, *Lecture Notes in Computer Science*, vol. 8893, pp. 26–38. Springer (2014). DOI 10.1007/978-3-319-14054-4_2
348. Hao, F., Yi, X., Chen, L., Shahandashti, S.F.: The fairy-ring dance: Password authenticated key exchange in a group. In: R. Chow, G. Saldamli (eds.) Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security, IoTPTS@AsiaCCS 2015, pp. 27–34. ACM (2015). DOI 10.1145/2732209.2732212
349. Hardjono, T., Tsudik, G.: IP multicast security: Issues and directions. Annales de Telecom pp. 324–340 (2000)
350. Harkins, D.: Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In: J. Lopez, C.J. Mitchell (eds.) Second International Conference on Sensor Technologies and Applications, pp. 839–844. IEEE (2008)
351. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). The Internet Society (1998). RFC 2409
352. Harkins (Ed.), D.: Dragonfly Key Exchange. RFC 7664 (Informational) (2015). DOI 10.17487/RFC7664. URL https://www.rfc-editor.org/rfc/rfc7664.txt
353. He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.C.: A modular correctness proof of IEEE 802.11i and TLS. In: Proc. 12th ACM Conference on Computer and Communications Security (CCS) 2005, pp. 2–15. ACM (2005). DOI 10.1145/1102120.1102124
354. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: Detection of widespread weak keys in network devices. In: Proc. 21st USENIX Security Symposium (2012). URL https://factorable.net/paper.html
355. Hickman, K.E.B.: The SSL protocol (1995). URL http://www-archive.mozilla.org/projects/security/pki/nss/ssl/draft02.html. Internet-Draft
356. Hirose, S., Ikeda, K.: A conference key distribution system for the star configuration based on the discrete logarithm problem. Information Processing Letters **62**, 189–192 (1997)
357. Hirose, S., Yoshida, S.: An authenticated Diffie-Hellman key agreement protocol secure against active attacks. In: H. Imai, et al. (eds.) Public Key Cryptography, *Lecture Notes in Computer Science*, vol. 1431, pp. 135–148. Springer (1998)
358. Hitchcock, Y., Boyd, C., Nieto, J.M.G.: Tripartite key exchange in the Canetti-Krawczyk proof model. In: A. Canteaut, K. Viswanathan (eds.) Progress in Cryptology - INDOCRYPT 2004, *Lecture Notes in Computer Science*, vol. 3348, pp. 17–32. Springer (2004)

359. Hitchcock, Y., Boyd, C., Nieto, J.M.G.: Modular proofs for key exchange: rigorous optimizations in the Canetti–Krawczyk model. Appl. Algebra Eng. Commun. Comput. **16**(6), 405–438 (2006). DOI 10.1007/s00200-005-0185-9
360. Hodges, J., Jackson, C., Barth, A.: HTTP Strict Transport Security (HSTS). RFC 6797 (Proposed Standard) (2012). DOI 10.17487/RFC6797. URL https://www.rfc-editor.org/rfc/rfc6797.txt
361. Hoeper, K., Gong, G.: Integrated DH-like key exchange protocols from LUC, GH and XTR. In: 2006 IEEE International Symposium on Information Theory, pp. 922–926. IEEE (2006)
362. Hollenbeck, S.: Transport Layer Security Protocol Compression Methods. RFC 3749 (Proposed Standard) (2004). DOI 10.17487/RFC3749. URL https://www.rfc-editor.org/rfc/rfc3749.txt
363. Horng, G.: An efficient and secure protocol for multi-party key establishment. The Computer Journal **44**(5), 463–470 (2001)
364. Horng, G., Hsu, C.K.: Weaknesses in the Helsinki protocol. Electronics Letters **34**(4), 354–355 (1998)
365. Horster, P., Michels, M., Petersen, H.: Authenticated encryption schemes with low communications costs. Electronics Letters **30**(15), 1212–1213 (1994)
366. Huang, H.: Strongly secure one round authenticated key exchange protocol with perfect forward security. In: X. Boyen, X. Chen (eds.) Provable Security - 5th International Conference, ProvSec 2011, *Lecture Notes in Computer Science*, vol. 6980, pp. 389–397. Springer (2011). DOI 10.1007/978-3-642-24316-5_28
367. Huang, H., Cao, Z.: Strongly secure authenticated key exchange protocol based on computational Diffie-Hellman problem. Cryptology ePrint Archive, Report 2008/500 (2008). URL https://eprint.iacr.org/2008/500
368. Huang, H., Cao, Z.: An ID-based authenticated key exchange protocol based on bilinear Diffie–Hellman problem. In: R. Safavi-Naini, V. Varadharajan (eds.) ACM Symposium on Information, Computer and Communications Security – ASIACCS 2009, pp. 333–342. ACM (2009)
369. Hwang, T., Chen, J.L.: Identity-based conference key broadcast systems. IEE Proceedings - Computers and Digital Techniques **141**(1), 57–60 (1994)
370. Hwang, T., Chen, Y.H.: On the security of SPLICE/AS – the authentication system in WIDE internet. Inf. Process. Lett. **53**(2), 97–101 (1995)
371. I'Anson, C., Mitchell, C.J.: Security defects in CCITT recommendation X.509 – the directory authentication framework. ACM Computer Communication Review **20**(2), 30–34 (1990)
372. IEEE: P1363 Standard Specifications for Public-Key Cryptography (2000). IEEE Std 1363-2000
373. IEEE: P1363.2 Standard Specifications for Password-based Public-Key Cryptographic Techniques (2009). Https://doi.org/10.1109/IEEESTD.2009.4773330
374. Ingemarsson, I., Tang, D.T., Wong, C.K.: A conference key distribution system. IEEE Transactions on Information Theory **IT-28**(5), 714–720 (1982)
375. ISO: Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture ISO 7498-2 (1989). International Standard
376. ISO: Information Technology – Security Techniques – Key Management – Part 2: Mechanisms Using Symmetric Techniques ISO/IEC 11770-2 (1996). International Standard
377. ISO: Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 3: Entity Authentication Using a Public Key Algorithm ISO/IEC 9798-3, 2nd edn. (1998). International Standard

378. ISO: Information Technology – Security Techniques – Entity Authentication – Part 4: Mechanisms Using a Cryptographic Check Function ISO/IEC 9798-4, 2nd edn. (1999). International Standard
379. ISO: Information Technology – Security Techniques – Key Management – Part 4: Mechanisms based on weak secrets ISO/IEC 11770-4, 1st edn. (2006). International Standard
380. ISO: Information Technology – Security Techniques – Entity Authentication – Part 2: Mechanisms Using Symmetric Encipherment Algorithms ISO/IEC 9798-2, 3rd edn. (2008). International Standard
381. ISO: Information Technology – Security Techniques – Key Management – Part 2: Mechanisms Using Symmetric Techniques ISO/IEC 11770-2, 2nd edn. (2008). International Standard
382. ISO: Information Technology – Security Techniques – Entity Authentication Mechanisms – Part 5: Mechanisms using Zero Knowledge Techniques ISO/IEC 9798-5, 3rd edn. (2009). International Standard
383. ISO: Information Technology – Security Techniques – Key Management – Part 3: Mechanisms Using Asymmetric Techniques ISO/IEC 11770-3, 3rd edn. (2015). International Standard
384. ISO: Information Technology – Security Techniques – Key Management – Part 6: Key derivation ISO/IEC 11770-4, 1st edn. (2016). International Standard
385. ISO: Information Technology – Security Techniques – Key Management – Part 4: Mechanisms based on weak secrets ISO/IEC 11770-4, 2nd edn. (2017). International Standard
386. ITU: Password-authenticated key exchange (PAK) protocol, ITU-T Rec. X.1035 (2007). ITU-T Recommendation
387. ITU/ISO: Information Technology – Open Systems Interconnection – The Directory – Part 8: Authentication Framework, ITU-T Rec. X.509 – ISO/IEC 9594-8 (1995). International Standard
388. Jablon, D.P.: Strong password-only authenticated key exchange. ACM Computer Communication Review **26**(5), 5–26 (1996)
389. Jablon, D.P.: Extended password key exchange protocols immune to dictionary attack. In: 6th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 248–255. IEEE Press (1997)
390. Jablon, D.P.: Password authentication using multiple servers. In: D. Naccache (ed.) Topics in Cryptology – CT-RSA 2001, *Lecture Notes in Computer Science*, vol. 2020, pp. 344–360. Springer (2001)
391. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: Generic compilers for authenticated key exchange. In: M. Abe (ed.) Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, *Lecture Notes in Computer Science*, vol. 6477, pp. 232–249. Springer (2010). DOI 10.1007/978-3-642-17373-8_14
392. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: R. Safavi-Naini, R. Canetti (eds.) Advances in Cryptology – CRYPTO 2012, *Lecture Notes in Computer Science*, vol. 7417, pp. 273–293. Springer (2012). DOI 10.1007/978-3-642-32009-5_17. URL https://eprint.iacr.org/2011/219
393. Jager, T., Schwenk, J., Somorovsky, J.: On the security of TLS 1.3 and QUIC against weaknesses in PKCS#1 v1.5 Encryption. In: I. Ray, et al. (eds.) 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1185–1196. ACM (2015)
394. Janson, P., Tsudik, G.: Secure and minimal protocols for authenticated key distribution. Computer Communications **18**(9), 645–653 (1995)

395. Jao, D., Feo, L.D.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: B. Yang (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, *Lecture Notes in Computer Science*, vol. 7071, pp. 19–34. Springer (2011). DOI 10.1007/978-3-642-25405-5_2
396. Jaspan, B.: Dual-workfactor encrypted key exchange: Efficiently preventing password chaining and dictionary attacks. In: 6th USENIX Security Symposium. San Jose, California (1996). URL https://www.usenix.org/legacy/publications/library/proceedings/sec96/jaspan.html
397. Jeong, I.R., Katz, J., Lee, D.H.: One-round protocols for two-party authenticated key exchange. In: M. Jakobsson, et al. (eds.) Applied Cryptography and Network Security, ACNS 2004, *Lecture Notes in Computer Science*, vol. 3089, pp. 220–232. Springer (2004). URL https://www.cs.umd.edu/~jkatz/papers/1round_AKE.pdf
398. Jeong, I.R., Kwon, J.O., Lee, D.H.: Strong Diffie–Hellman-DSA key exchange. IEEE Communications Letters **11**(5), 432–433 (2007). DOI 10.1109/LCOMM.2007.070004
399. Jiang, S., Gong, G.: Password based key exchange with mutual authentication. In: H. Handschuh, M.A. Hasan (eds.) Selected Areas in Cryptography, 11th International Workshop, SAC 2004, *Lecture Notes in Computer Science*, vol. 3357, pp. 267–279. Springer (2004). DOI 10.1007/978-3-540-30564-4_19
400. Jiang, S., Safavi-Naini, R.: An efficient deniable key exchange protocol. In: G. Tsudik (ed.) Financial Cryptography and Data Security, 12th International Conference, FC 2008, *Lecture Notes in Computer Science*, vol. 5143, pp. 47–52 (2008)
401. Jonsson, J., Kaliski Jr., B.S.: On the security of RSA encryption in TLS. In: M. Yung (ed.) Advances in Cryptology - CRYPTO 2002, *Lecture Notes in Computer Science*, vol. 2442, pp. 127–142. Springer (2002)
402. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In: W. Bosma (ed.) Algorithmic Number Theory, 4th International Symposium, ANTS-IV, *Lecture Notes in Computer Science*, vol. 1838, pp. 385–393. Springer (2000)
403. Joux, A., Nguyen, K.: Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups. J. Cryptology **16**(4), 239–247 (2003). DOI 10.1007/s00145-003-0052-4
404. Juels, A., Brainard, J.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: Network and Distributed System Security Symposium. Internet Society (1999). URL https://www.ndss-symposium.org/ndss1999/cryptographic-defense-against-connection-depletion-attacks/
405. Jürjens, J.: Security analysis of crypto-based Java programs using automated theorem provers. In: Proc. 21st IEEE/ACM International Conf. on Automated Software Engineering (ASE) 2006., pp. 167–176 (2006)
406. Just, M., van Oorschot, P.C.: Addressing the problem of undetected signature key compromise. In: Network and Distributed System Security Symposium. Internet Society (1999). URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/Addressing-the-Problem-of-Undetected-Signature-Key-Compromise-Mike-Just.pdf
407. Just, M., Vaudenay, S.: Authenticated multi-party key agreement. In: K. Kim, et al. (eds.) Advances in Cryptology – ASIACRYPT '96, *Lecture Notes in Computer Science*, vol. 1163, pp. 36–49. Springer (1996)
408. Kaliski Jr., B.S.: PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational) (1998). DOI 10.17487/RFC2313. URL https://www.rfc-editor.org/rfc/rfc2313.txt

409. Kaliski Jr., B.S.: An unknown key-share attack on the MQV key agreement protocol. ACM Trans. Inf. Syst. Secur. **4**(3), 275–288 (2001)
410. Kamil, A., Lowe, G.: Analysing TLS in the strand spaces model. Journal of Computer Security **19**(5), 975–1025 (2011)
411. Karn, P., Simpson, W.: Photuris: Session-Key Management Protocol. RFC 2522 (Experimental) (1999). DOI 10.17487/RFC2522. URL https://www.rfc-editor.org/rfc/rfc2522.txt
412. Katz, J., Lindell, Y.: Modern Cryptography, 2nd edn. CRC Press (2015)
413. Katz, J., MacKenzie, P.D., Taban, G., Gligor, V.D.: Two-server password-only authenticated key exchange. In: J. Ioannidis, et al. (eds.) Applied Cryptography and Network Security, Third International Conference, ACNS 2005, *Lecture Notes in Computer Science*, vol. 3531, pp. 1–16 (2005). DOI 10.1007/11496137_1
414. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: B. Pfitzmann (ed.) Advances in Cryptology – EUROCRYPT 2001, *Lecture Notes in Computer Science*, vol. 2045, pp. 475–494. Springer (2001)
415. Katz, J., Shin, J.S.: Modeling insider attacks on group key-exchange protocols. In: V. Atluri, et al. (eds.) Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, pp. 180–189. ACM (2005)
416. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: M. Matsui (ed.) Advances in Cryptology - ASIACRYPT 2009, *Lecture Notes in Computer Science*, vol. 5912, pp. 636–652. Springer (2009). DOI 10.1007/978-3-642-10366-7_37
417. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Y. Ishai (ed.) Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, *Lecture Notes in Computer Science*, vol. 6597, pp. 293–310. Springer (2011). DOI 10.1007/978-3-642-19571-6_18
418. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In: D. Boneh (ed.) Advances in Cryptology - CRYPTO 2003, *Lecture Notes in Computer Science*, vol. 2729, pp. 110–125. Springer (2003)
419. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. Journal of Cryptology **20**(1), 85–113 (2007)
420. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (Internet Standard) (2014). DOI 10.17487/RFC7296. URL https://www.rfc-editor.org/rfc/rfc7296.txt
421. Kaufman, C., Perlman, R.: PDM: A new strong password-based protocol. In: 10th USENIX Security Symposium (2001). URL https://www.usenix.org/legacy/publications/library/proceedings/sec01/kaufman.html
422. Kelsey, J.: Compression and information leakage of plaintext. In: J. Daemen, V. Rijmen (eds.) Fast Software Encryption, 9th International Workshop, FSE 2002, *Lecture Notes in Computer Science*, vol. 2365, pp. 263–276. Springer (2002). DOI 10.1007/3-540-45661-9_21. URL http://www.iacr.org/cryptodb/archive/2002/FSE/3091/3091.pdf
423. Kelsey, J., Schneier, B., Wagner, D.: Protocol interactions and the chosen protocol attack. In: B. Christianson, et al. (eds.) Security Protocols – 5th International Workshop, *Lecture Notes in Computer Science*, vol. 1361, pp. 91–104. Springer (1998)
424. Kemmerer, R., Meadows, C., Millen, J.: Three systems for cryptographic protocol analysis. Journal of Cryptology **7**(2), 79–130 (1994)

425. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard) (2005). DOI 10.17487/RFC4301. URL https://www.rfc-editor.org/rfc/rfc4301.txt
426. Kiefer, F., Manulis, M.: Blind password registration for verifier-based PAKE. In: K. Emura, et al. (eds.) Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, pp. 39–48. ACM (2016). DOI 10.1145/2898420.2898424
427. Kiefer, F., Manulis, M.: Universally composable two-server PAKE. In: M. Bishop, A.C.A. Nascimento (eds.) Information Security - 19th International Conference, ISC 2016, *Lecture Notes in Computer Science*, vol. 9866, pp. 147–166. Springer (2016). DOI 10.1007/978-3-319-45871-7_10
428. Kim, H., Lee, S., Lee, D.H.: Constant-round authenticated group key exchange for dynamic groups. In: P.J. Lee (ed.) Advances in Cryptology - ASIACRYPT 2004, *Lecture Notes in Computer Science*, vol. 3329, pp. 245–259. Springer (2004)
429. Kim, M., Fujioka, A., Ustaoglu, B.: Strongly secure authenticated key exchange without NAXOS' approach. In: T. Takagi, M. Mambo (eds.) Advances in Information and Computer Security, 4th International Workshop on Security, IWSEC 2009, *Lecture Notes in Computer Science*, vol. 5824, pp. 174–191. Springer (2009)
430. Kim, S., Mambo, M., Okamoto, T., Shizuya, H., Tada, M., Won, D.: On the security of the Okamoto-Tanaka ID-based key exchange scheme against active attacks. IEICE Transactions Fundamentals **E84-A**(1), 231–238 (2001)
431. Kim, Y., Perrig, A., Tsudik, G.: Simple and fault-tolerant key agreement for dynamic collaborative groups. In: 7th ACM Conference on Computer and Communications Security, pp. 235–244. ACM Press (2000)
432. Kim, Y., Perrig, A., Tsudik, G.: Tree-based group key agreement. ACM Trans. Inf. Syst. Secur. **7**(1), 60–96 (2004)
433. Klein, B., Otten, M., Beth, T.: Conference key distribution protocols in distributed systems. In: P.G. Farrell (ed.) Codes and Cyphers – Cryptography and Coding IV, pp. 225–241. IMA (1995)
434. Klíma, V., Pokorný, O., Rosa, T.: Attacking RSA-based sessions in SSL/TLS. In: C.D. Walter, et al. (eds.) Cryptographic Hardware and Embedded Systems (CHES) 2003, *Lecture Notes in Computer Science*, vol. 2779, pp. 426–440. Springer (2003). URL https://eprint.iacr.org/2003/052
435. Kobara, K., Imai, H.: Pretty-simple password-authenticated key-exchange under standard assumptions. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E85-A**(10), 2229–2237 (2002). URL https://eprint.iacr.org/2003/038
436. Kobara, K., Imai, H.: Pretty-simple password-authenticated key-exchange under standard assumptions (2003). URL https://eprint.iacr.org/2003/038
437. Koblitz, N.: Algebraic Aspects of Cryptography. Springer (1998)
438. Koblitz, N., Menezes, A.: Another look at "provable security". Journal of Cryptology **20**(1), 3–37 (2007)
439. Kohl, J., Neuman, C.: The Kerberos Network Authentication Service (V5). RFC 1510 (Historic) (1993). DOI 10.17487/RFC1510. URL https://www.rfc-editor.org/rfc/rfc1510.txt
440. Kohl, J.T.: The use of encryption in Kerberos for network authentication. In: G. Brassard (ed.) Advances in Cryptology – CRYPTO '89, *Lecture Notes in Computer Science*, vol. 435, pp. 35–43. Springer (1989)

441. Kohl, J.T., Neuman, B.C., Ts'o, T.Y.: The evolution of the Kerberos authentication system. In: F. Brazier, et al. (eds.) Distributed Open Systems, pp. 78–94. IEEE Computer Society Press (1994)
442. Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive: Report 2013/367 (2013). URL https://eprint.iacr.org/2013/367
443. Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: (De-)Constructing TLS. Cryptology ePrint Archive, Report 2014/020 (2014). URL https://eprint.iacr.org/2014/020
444. Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: De-constructing TLS 1.3. In: 16th International Conference on Progress in Cryptology, INDOCRYPT 2015, *Lecture Notes in Computer Science*, vol. 9462, pp. 85–102. Springer (2015)
445. Kolesnikov, V., Rackoff, C.: Key exchange using passwords and long keys. In: S. Halevi, T. Rabin (eds.) Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, *Lecture Notes in Computer Science*, vol. 3876, pp. 100–119. Springer (2006). DOI 10.1007/11681878_6
446. Konstantinou, E.: An efficient constant round ID-based group key agreement protocol for ad hoc networks. In: J. Lopez, et al. (eds.) Network and System Security - 7th International Conference, NSS 2013, *Lecture Notes in Computer Science*, vol. 7873, pp. 563–574. Springer (2013)
447. Koyama, K.: Secure conference key distribution schemes for conspiracy attack. In: R.A. Rueppel (ed.) Advances in Cryptology – EUROCRYPT '92, *Lecture Notes in Computer Science*, vol. 658, pp. 449–453. Springer (1992)
448. Koyama, K., Ohta, K.: Identity-based conference key distribution systems. In: C. Pomerance (ed.) Advances in Cryptology – CRYPTO '87, *Lecture Notes in Computer Science*, vol. 293, pp. 175–184. Springer (1987)
449. Koyama, K., Ohta, K.: Security of improved identity-based conference key distribution systems. In: C.G. Günther (ed.) Advances in Cryptology – EUROCRYPT '88, *Lecture Notes in Computer Science*, vol. 330, pp. 11–19. Springer (1988)
450. Krawczyk, H.: SKEME: A versatile secure key exchange mechanism for Internet. In: Symposium on Network and Distributed System Security, pp. 114–127. IEEE Computer Society Press (1996)
451. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: J. Kilian (ed.) Advances in Cryptology – CRYPTO 2001, *Lecture Notes in Computer Science*, vol. 2139, pp. 310–331. Springer (2001). DOI 10.1007/3-540-44647-8_19
452. Krawczyk, H.: SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie–Hellman and its use in the IKE-protocols. In: D. Boneh (ed.) Advances in Cryptology - CRYPTO 2003, *Lecture Notes in Computer Science*, vol. 2729, pp. 400–425. Springer (2003). DOI 10.1007/978-3-540-45146-4_24
453. Krawczyk, H.: HMQV: A high-performance secure Diffie–Hellman protocol. In: V. Shoup (ed.) Advances in Cryptology – CRYPTO 2005, *Lecture Notes in Computer Science*, vol. 3621, pp. 546–566. Springer (2005). URL https://eprint.iacr.org/2005/176
454. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: T. Rabin (ed.) Advances in Cryptology - CRYPTO 2010, *Lecture Notes in Computer Science*, vol. 6223, pp. 631–648. Springer (2010)
455. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: T. Rabin (ed.) Advances in Cryptology - CRYPTO 2010, *Lecture Notes in Computer Science*, vol. 6223, pp. 631–648. Springer (2010)

456. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: R. Canetti, J. Garay (eds.) Advances in Cryptology – Proc. CRYPTO 2013, *Lecture Notes in Computer Science*, vol. 8042, pp. 429–448. Springer (2013). DOI 10.1007/978-3-642-40041-4_24. URL https://eprint.iacr.org/2013/339
457. Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. In: 1st IEEE European Symposium on Security and Privacy, EuroS&P 2016, pp. 81–96. IEEE (2016)
458. Ku, W.C., Wang, S.D.: Cryptanalysis of modified authenticated key agreement protocol. Electronics Letters **36**(21), 1770–1771 (2000)
459. Kudla, C., Paterson, K.G.: Modular security proofs for key agreement protocols. In: B.K. Roy (ed.) Advances in Cryptology - ASIACRYPT 2005, *Lecture Notes in Computer Science*, vol. 3788, pp. 549–565. Springer (2005)
460. Kühn, U., Pyshkin, A., Tews, E., Weinmann, R.P.: Variants of Bleichenbacher's low-exponent attack on PKCS#1 RSA signatures (2008). URL https://www-old.cdc.informatik.tu-darmstadt.de/reports/reports/sigflaw.pdf
461. Kumar, S., Paar, C., Pelzl, J., Pfeiffer, G., Schimmler, M.: Breaking ciphers with COPACOBANA – a cost-optimized parallel code breaker. In: L. Goubin, M. Matsui (eds.) Cryptographic Hardware and Embedded Systems (CHES) 2006, *Lecture Notes in Computer Science*, vol. 4249, pp. 101–118. Springer (2006)
462. Kunz-Jacques, S., Pointcheval, D.: About the security of MTI/C0 and MQV. In: R.D. Prisco, M. Yung (eds.) Security and Cryptography for Networks, 5th International Conference, SCN 2006, *Lecture Notes in Computer Science*, vol. 4116, pp. 156–172. Springer (2006)
463. Kunz-Jacques, S., Pointcheval, D.: A new key exchange protocol based on MQV assuming public computations. In: R.D. Prisco, M. Yung (eds.) Security and Cryptography for Networks, 5th International Conference, SCN 2006, *Lecture Notes in Computer Science*, vol. 4116, pp. 186–200. Springer (2006)
464. Küsters, R., Truderung, T.: Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In: Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, pp. 157–171. IEEE Computer Society (2009). DOI 10.1109/CSF.2009.17
465. Kwon, T.: Authentication and key agreement via memorable passwords. Cryptology ePrint Archive, Report 2000/026 (2000). URL https://eprint.iacr.org/2000/026
466. Kwon, T.: Authentication and key agreement via memorable passwords. In: Network and Distributed System Security Symposium – NDSS (2001). URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/Authentication-and-Key-Agreement-Via-Memorable-Passwords-Taekyoung-Kwon.pdf
467. Kwon, T., Song, J.: Efficient and secure password-based authentication protocols against guessing attacks. Computer Communications **21**, 853–861 (1998)
468. Kwon, T., Song, J.: Efficient key exchange and authentication protocols protecting weak secrets. IEICE Transactions Fundamentals **E81-A**(1), 156–163 (1998)
469. Laih, C.S., Lee, J.Y., Harn, L.: A new threshold scheme and its application in designing the conference key distribution cryptosystem. Information Processing Letters **32**, 95–99 (1989)
470. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: W. Susilo, et al. (eds.) Provable Security, First International Conference, ProvSec 2007, *Lecture Notes in Computer Science*, vol. 4784, pp. 1–16. Springer (2007)

471. Lancrenon, J., Skrobot, M., Tang, Q.: Two more efficient variants of the J-PAKE protocol. In: M. Manulis, et al. (eds.) Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, *Lecture Notes in Computer Science*, vol. 9696, pp. 58–76. Springer (2016). DOI 10.1007/978-3-319-39555-5_4
472. Lancrenon, J., Škrobot, M.: On the provable security of the Dragonfly protocol. In: J. Lopez, C.J. Mitchell (eds.) Information Security - 18th International Conference, ISC 2015, *Lecture Notes in Computer Science*, vol. 9290, pp. 244–261. Springer (2015). URL https://orbilu.uni.lu/bitstream/10993/24767/1/Dragonfly.pdf
473. Langley, A.: PKCS#1 signature validation (2014). URL https://www.imperialviolet.org/2014/09/26/pkcs1.html
474. Langley, A.: POODLE attacks on SSLv3 (2014). URL https://www.imperialviolet.org/2014/10/14/poodle.html
475. Langley, A.G.: BEAST followup (2012). URL https://www.imperialviolet.org/2012/01/15/beastfollowup.html
476. Langley, A.G.: Apple's SSL/TLS bug (2014). URL https://www.imperialviolet.org/2014/02/22/applebug.html
477. Lauter, K.E., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: M. Yung, et al. (eds.) Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, *Lecture Notes in Computer Science*, vol. 3958, pp. 378–394. Springer (2006)
478. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Designs, Codes and Cryptography **28**(2), 119–134 (2003)
479. Lee, J., Park, C.S.: An efficient authenticated key exchange protocol with a tight security reduction. Cryptology ePrint Archive, Report 2008/345 (2008). URL https://eprint.iacr.org/2008/345
480. Lee, J., Park, J.H.: Authenticated key exchange secure under the computational Diffie–Hellman assumption. Cryptology ePrint Archive, Report 2008/344 (2008). URL https://eprint.iacr.org/2008/344
481. Lenstra, A., Verheul, E.: The XTR public key system. In: M. Bellare (ed.) Advances in Cryptology – Crypto 2000, *Lecture Notes in Computer Science*, vol. 1880, pp. 1–19. Springer (2000)
482. Lenstra, A.K., de Weger, B.: On the possibility of constructing meaningful hash collisions for public keys. In: C. Boyd, J.M.G. Nieto (eds.) Information Security and Privacy, 10th Australasian Conference, *Lecture Notes in Computer Science*, vol. 3574, pp. 267–279. Springer (2005)
483. Lepidum: CCS injection vulnerability (2014). URL http://ccsinjection.lepidum.co.jp/
484. Leyden, J.: AVG on Heartbleed: It's dangerous to go alone. Take this (an AVG tool) (2014). URL http://www.theregister.co.uk/2014/05/20/heartbleed_still_prevalent/
485. Li, H., Wu, C., Sun, J.: A general compiler for password-authenticated group key exchange protocol. Inf. Process. Lett. **110**(4), 160–167 (2010). DOI 10.1016/j.ipl.2009.11.013
486. Li, S., Yuan, Q., Li, J.: Towards security two-part authenticated key agreement protocols. Cryptology ePrint Archive, Report 2005/300 (2005). URL https://eprint.iacr.org/2005/300
487. Li, X., Xu, J., Zhang, Z., Feng, D., Hu, H.: Multiple handshakes security of TLS 1.3 candidates. In: 37th IEEE Symposium on Security and Privacy, SP 2016 (2016)

488. Li, Y., Schäge, S., Yang, Z., Bader, C., Schwenk, J.: New modular compilers for authenticated key exchange. In: I. Boureanu, et al. (eds.) Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, *Lecture Notes in Computer Science*, vol. 8479, pp. 1–18. Springer (2014). DOI 10.1007/978-3-319-07536-5_1
489. Li, Y., Schäge, S., Yang, Z., Kohlar, F., Schwenk, J.: On the security of the pre-shared key ciphersuites of TLS. In: H. Krawczyk (ed.) Public Key Cryptography (PKC) 2014, *Lecture Notes in Computer Science*, vol. 8383, pp. 669–684. Springer (2014)
490. Li, Y., Yang, Z.: Strongly secure one-round group authenticated key exchange in the standard model. In: M. Abdalla, et al. (eds.) Cryptology and Network Security - 12th International Conference, CANS 2013, *Lecture Notes in Computer Science*, vol. 8257, pp. 122–138. Springer (2013)
491. Lim, C.H., Lee, P.J.: Several practical protocols for authentication and key exchange. Information Processing Letters **53**, 91–96 (1995)
492. Lim, C.H., Lee, P.J.: A key recovery attack on discrete log-based schemes using a prime order subgroup. In: B.S. Kaliski Jr. (ed.) Advances in Cryptology – CRYPTO '97, *Lecture Notes in Computer Science*, vol. 1294, pp. 249–263. Springer (1997)
493. Lin, C.L., Sun, H.M., Hwang, T.: Three-party encrypted key exchange: Attacks and a solution. ACM Operating Systems Review **34**(4), 12–20 (2000)
494. Lin, C.L., Sun, H.M., Steiner, M., Hwang, T.: Three-party encrypted key exchange without server public-keys. IEEE Communications Letters **5**(12), 497–499 (2001)
495. Lin, I.C., Chang, C.C., Hwang, M.S.: Security enhancement for the 'simple authentication key agreement algorithm'. In: 24th Computer Software and Applications Conference (COMPSAC 2000), pp. 113–115. IEEE Computer Society Press (2000)
496. Lippold, G., Boyd, C., Nieto, J.G.: Strongly secure certificateless key agreement. Cryptology ePrint Archive, Report 2009/219 (2009). URL https://eprint.iacr.org/2009/219
497. Lippold, G., Boyd, C., Nieto, J.M.G.: Strongly secure certificateless key agreement. In: H. Shacham, B. Waters (eds.) Pairing-Based Cryptography - Pairing 2009, *Lecture Notes in Computer Science*, vol. 5671, pp. 206–230. Springer (2009)
498. Liu, S., Sakurai, K., Weng, J., Zhang, F., Zhao, Y.: Security model and analysis of FHMQV, revisited. In: D. Lin, et al. (eds.) Information Security and Cryptology - Inscrypt 2013, *Lecture Notes in Computer Science*, vol. 8567, pp. 255–269. Springer (2014)
499. Liu, W., Ma, W., Yang, Y.: A new attack on the BAN modified Andrew secure RPC protocol. In: Proceedings of Second International Conference on Networks Security, Wireless Communications and Trusted Computing, pp. 219–222. IEEE (2010). DOI 10.1145/2898420.2898424
500. Lomas, T.M.A., Gong, L., Saltzer, J.H., Needham, R.M.: Reducing risks from poorly chosen keys. In: G.R. Andrews (ed.) Proceedings of the Twelfth ACM Symposium on Operating System Principles, SOSP 1989, pp. 14–18. ACM (1989). DOI 10.1145/74850.74853
501. Lowe, G.: Breaking and fixing the Needham-Schroeder public key protocol using FDR. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 147–166. Springer (1996)
502. Lowe, G.: Some new attacks upon security protocols. In: 9th IEEE Computer Security Foundations Workshop, pp. 162–169. IEEE Computer Society Press (1996)
503. Lowe, G.: Casper: A compiler for the analysis of security protocols. In: 10th IEEE Computer Security Foundations Workshop, pp. 18–30. IEEE Computer Society Press (1997)
504. Lowe, G.: A hierarchy of authentication specification. In: 10th IEEE Computer Security Foundations Workshop, pp. 31–43. IEEE Computer Society Press (1997)

505. Lu, S., Zhao, J., Cheng, Q.: Cryptanalysis and improvement of an efficient authenticated key exchange protocol with tight security reduction. Int. J. Communication Systems **29**(3), 567–578 (2016). DOI 10.1002/dac.2899
506. Lucks, S.: Open key exchange: How to defeat dictionary attacks without encrypting public keys. In: B. Christianson, et al. (eds.) Security Protocols – 5th International Workshop, *Lecture Notes in Computer Science*, vol. 1361, pp. 79–90. Springer (1998)
507. MacKenzie, P.: More efficient password-authenticated key exchange. In: D. Naccache (ed.) Topics in Cryptology – CT-RSA 2001, *Lecture Notes in Computer Science*, vol. 2020, pp. 361–377. Springer (2001)
508. MacKenzie, P.: On the security of the SPEKE password-authenticated key exchange protocol (2001). URL https://eprint.iacr.org/2001/057
509. MacKenzie, P.: The PAK suite: Protocols for password-authenticated key exchange. Tech. Rep. 2002-46, DIMACS (2002). URL http://dimacs.rutgers.edu/TechnicalReports/abstracts/2002/2002-46.html
510. MacKenzie, P., Patel, S., Swaminathan, R.: Password-authenticated key exchange based on RSA. In: T. Okamoto (ed.) Advances in Cryptology – ASIACRYPT 2000, *Lecture Notes in Computer Science*, vol. 1976, pp. 599–613. Springer (2000)
511. MacKenzie, P.D., Patel, S.: Hard bits of the discrete log with applications to password authentication. In: A. Menezes (ed.) Topics in Cryptology - CT-RSA 2005, *Lecture Notes in Computer Science*, vol. 3376, pp. 209–226. Springer (2005). DOI 10.1007/978-3-540-30574-3_15
512. MacKenzie, P.D., Shrimpton, T., Jakobsson, M.: Threshold password-authenticated key exchange. In: M. Yung (ed.) Advances in Cryptology - CRYPTO 2002, *Lecture Notes in Computer Science*, vol. 2442, pp. 385–400. Springer (2002). DOI 10.1007/3-540-45708-9_25
513. Mailloux, N., Miri, A., Nevins, M.: COMPASS: Authenticated group key agreement from signcryption. In: J. García-Alfaro, et al. (eds.) Foundations and Practice of Security - 5th International Symposium, FPS 2012, *Lecture Notes in Computer Science*, vol. 7743, pp. 95–114. Springer (2012)
514. Mambo, M., Shizuya, H.: A note on the complexity of breaking Okamoto-Tanaka ID-based key exchange scheme. IEICE Transactions Fundamentals **E82-A**(1), 77–80 (1999)
515. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: M. Matsui (ed.) Fast Software Encryption, 8th International Workshop, *Lecture Notes in Computer Science*, vol. 2355, pp. 152–164. Springer (2001)
516. Manulis, M.: Group key exchange enabling on-demand derivation of peer-to-peer keys. In: M. Abdalla, et al. (eds.) Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, *Lecture Notes in Computer Science*, vol. 5536, pp. 1–19 (2009)
517. Manulis, M., Stebila, D., Denham, N.: Secure modular password authentication for the web using channel bindings. In: L. Chen, C.J. Mitchell (eds.) Security Standardisation Research - First International Conference, SSR 2014, *Lecture Notes in Computer Science*, vol. 8893, pp. 167–189. Springer (2014)
518. Manulis, M., Suzuki, K., Ustaoglu, B.: Modeling leakage of ephemeral secrets in tripartite/group key exchange. In: D. Lee, S. Hong (eds.) Information, Security and Cryptology - ICISC 2009, *Lecture Notes in Computer Science*, vol. 5984, pp. 16–33. Springer (2009)
519. Manulis, M., Suzuki, K., Ustaoglu, B.: Modeling leakage of ephemeral secrets in tripartite/group key exchange. IEICE Transactions **96-A**(1), 101–110 (2013)
520. Mao, W.: Modern Cryptography: Theory and Practice. Prentice Hall (2003)

521. Mao, W., Boyd, C.: On the use of encryption in cryptographic protocols. In: P.G. Farrell (ed.) Codes and Cyphers – Cryptography and Coding IV, pp. 251–262 (1995)
522. Mao, W., Paterson, K.G.: On the plausible deniability feature of internet protocols (2002). URL http://www.isg.rhul.ac.uk/~kp/IKE.ps
523. Margaritelli, S.: SSL stripping and HSTS bypass with BetterCap (2016). URL https://www.bettercap.org/legacy/
524. Marlinspike, M.: New tricks for defeating SSL in practice. In: Black Hat DC (2009). https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf, http://www.thoughtcrime.org/software/sslstrip/
525. Mathuria, A., Sriram, G.: New attacks on ISO key establishment protocols. IACR Cryptology ePrint Archive (2008). URL https://eprint.iacr.org/2008/336
526. Matsumoto, T., Takashima, Y., Imai, H.: On seeking smart public-key-distribution systems. Transactions of the IECE of Japan **E69**(2), 99–106 (1986)
527. Maurer, U.: Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In: Y. Desmedt (ed.) Advances in Cryptology – CRYPTO '94, *Lecture Notes in Computer Science*, vol. 839, pp. 271–281. Springer (1994)
528. Maurer, U.: Constructive cryptography — a new paradigm for security definitions and proofs. In: S. Mödersheim, C. Palamidessi (eds.) Theory of Security and Applications, *Lecture Notes in Computer Science*, vol. 6993, pp. 33–56. Springer (2012). DOI 10.1007/978-3-642-27375-9_3
529. Mavrogiannopoulos, N., Vercauteren, F., Velichkov, V., Preneel, B.: A cross-protocol attack on the TLS protocol. In: Proc. 2012 ACM Conference on Computer and Communications Security (CCS), pp. 62–72. ACM (2012). DOI 10.1145/2382196.2382206
530. Mayer, A., Yung, M.: Secure protocol transformation via 'expansion': From two-party to groups. In: 6th ACM Conference on Computer and Communications Security, pp. 83–92. ACM Press (1999)
531. McCullagh, N., Barreto, P.S.L.M.: A new two-party identity-based authenticated key agreement. In: A. Menezes (ed.) Topics in Cryptology - CT-RSA 2005, *Lecture Notes in Computer Science*, vol. 3376, pp. 262–274. Springer (2005). URL https://eprint.iacr.org/2004/122
532. McCurley, K.S.: A key distribution system equivalent to factoring. Journal of Cryptology **1**(2), 95–105 (1988)
533. McGrew, D.A., Sherman, A.T.: Key establishment in large dynamic groups using one-way function trees (1998). URL http://www.cs.umbc.edu/~sherman/Papers/itse.ps
534. Meadows, C.: Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In: E. Bertino, et al. (eds.) 4th European Symposium on Research in Computer Security, ESORICS 1996, *Lecture Notes in Computer Science*, vol. 1146, pp. 351–364. Springer (1996)
535. Meadows, C.: The NRL Protocol Analyzer: An overview. The Journal of Logic Programming **26**(2), 113–131 (1996)
536. Meadows, C.: Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In: IEEE Symposium on Security and Privacy, pp. 216–231. IEEE Computer Society Press (1999)
537. Meadows, C.: A formal framework and evaluation method for network denial of service. In: 12th IEEE Computer Security Foundations Workshop, pp. 4–13. IEEE Computer Society Press (1999)
538. Meadows, C.: Formal methods for cryptographic protocol analysis: Emerging issues and trends. IEEE Journal on Selected Areas in Communications **21**(1), 44–54 (2003)

539. Meadows, C.A.: A cost-based framework for analysis of denial of service networks. Journal of Computer Security **9**(1/2), 143–164 (2001). URL https://www.nrl.navy.mil/itd/chacs/sites/www.nrl.navy.mil.itd.chacs/files/pdfs/Meadows2000.pdf
540. Meadows, C.A.: Emerging issues and trends in formal methods in cryptographic protocol analysis: Twelve years later. In: N. Martí-Oliet, et al. (eds.) Logic, Rewriting, and Concurrency, *Lecture Notes in Computer Science*, vol. 9200, pp. 475–492. Springer (2015). DOI 10.1007/978-3-319-23165-5_22
541. Meier, S., Cremers, C.J.F., Basin, D.A.: Strong invariants for the efficient construction of machine-checked protocol security proofs. In: Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, pp. 231–245. IEEE Computer Society (2010). DOI 10.1109/CSF.2010.23
542. Meier, S., Schmidt, B., Cremers, C., Basin, D.A.: The TAMARIN prover for the symbolic analysis of security protocols. In: N. Sharygina, H. Veith (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, *Lecture Notes in Computer Science*, vol. 8044, pp. 696–701. Springer (2013). DOI 10.1007/978-3-642-39799-8_48
543. Menezes, A.: Another look at HMQV. J. Mathematical Cryptology **1**(1), 47–64 (2007)
544. Menezes, A., Okamoto, T., Vanstone, S.A.: Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Transactions on Information Theory **39**(5), 1639–1646 (1993)
545. Menezes, A., Ustaoglu, B.: On the importance of public-key validation in the MQV and HMQV key agreement protocols. In: R. Barua, T. Lange (eds.) Progress in Cryptology - INDOCRYPT 2006, *Lecture Notes in Computer Science*, vol. 4329, pp. 133–147. Springer (2006)
546. Menezes, A., Ustaoglu, B.: Comparing the pre- and post-specified peer models for key agreement. In: Y. Mu, et al. (eds.) Information Security and Privacy, ACISP 2008, *Lecture Notes in Computer Science*, vol. 5107, pp. 53–68. Springer (2008)
547. Menezes, A., Ustaoglu, B.: Security arguments for the UM key agreement protocol in the NIST SP 800-56a standard. In: M. Abe, V.D. Gligor (eds.) Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, pp. 261–270. ACM (2008)
548. Menezes, A., Ustaoglu, B.: Comparing the pre- and post-specified peer models for key agreement. IJACT **1**(3), 236–250 (2009)
549. Menezes, A., Ustaoglu, B.: On reusing ephemeral keys in Diffie–Hellman key agreement protocols. IJACT **2**(2), 154–158 (2010). DOI 10.1504/IJACT.2010.038308
550. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
551. Menezes, A.J., Qu, M., Vanstone, S.A.: Some new key agreement protocols providing implicit authentication. In: Workshop on Selected Areas in Cryptography (SAC'95), pp. 22–32 (1995)
552. Meyer, C., Schwenk, J.: Lessons learned from previous SSL/TLS attacks - a brief chronology of attacks and weaknesses. Cryptology ePrint Archive, Report 2013/049 (2013). URL https://eprint.iacr.org/2013/049
553. Meyer, C., Somorovsky, J., Weiss, E., Schwenk, J., Schinzel, S., Tews, E.: Revisiting SSL/TLS implementations: New Bleichenbacher side channels and attacks. In: 23rd USENIX Security Symposium, pp. 733–748. USENIX Association (2014)
554. Millen, J.K.: A necessarily parallel attack. In: Workshop on Formal Methods and Security Protocols (1999). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.5086&rep=rep1&type=pdf

555. Mironov, I.: (Not so) Random shuffles of RC4. In: M. Yung (ed.) Advances in Cryptology – CRYPTO 2002, *Lecture Notes in Computer Science*, vol. 2442, pp. 304–319. Springer (2002)
556. Mitchell, C.J.: Making serial number based authentication robust against loss of state. ACM Operating Systems Review **34**(3), 56–59 (2000). DOI 10.1145/506117.506124
557. Mitchell, C.J.: Breaking the simple authenticated key agreement (SAKA) protocol. Tech. Rep. RHUL-MA-2001-2, Royal Holloway, University of London, Department of Mathematics (2001). URL http://www.ma.rhul.ac.uk/static/techrep/2001/RHUL-MA-2001-2.pdf
558. Mitchell, C.J., Thomas, A.: Standardising authentication protocols based on public key techniques. Journal of Computer Security **2**, 23–36 (1993)
559. Mitchell, C.J., Ward, M., Wilson, P.: On key control in key agreement protocols. Electronics Letters **34**, 980–981 (1998)
560. Mitchell, C.J., Yeun, C.Y.: Fixing a problem in the Helsinki protocol. ACM Operating Systems Review **32**(4), 21–24 (1998)
561. Mitchell, J.C., Mitchell, M., Stern, U.: Automated analysis of cryptographic protocols using Murϕ. In: IEEE Symposium on Security and Privacy, pp. 141–151. IEEE Computer Society Press (1997)
562. Mitchell, J.C., Shmatikov, V., Stern, U.: Finite-state analysis of SSL 3.0. In: 7th USENIX Security Symposium (1998). URL https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/mitchell/mitchell.pdf
563. Moeller, B., Langley, A.: TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks. RFC 7507 (Proposed Standard) (2015). DOI 10.17487/RFC7507. URL https://www.rfc-editor.org/rfc/rfc7507.txt
564. Möller, B.: Security of CBC ciphersuites in SSL/TLS: Problems and countermeasures (2002). URL https://www.openssl.org/~bodo/tls-cbc.txt
565. Möller, B., Duong, T., Kotowicz, K.: This POODLE bites: Exploiting the SSL 3.0 fallback (2014). URL https://www.openssl.org/~bodo/ssl-poodle.pdf
566. Moriyama, D., Okamoto, T.: An eCK-secure authenticated key exchange protocol without random oracles. In: J. Pieprzyk, F. Zhang (eds.) Provable Security, Third International Conference, ProvSec 2009, *Lecture Notes in Computer Science*, vol. 5848, pp. 154–167. Springer (2009)
567. Moriyama, D., Okamoto, T.: Leakage resilient eCK-secure key exchange protocol without random oracles. In: B.S.N. Cheung, et al. (eds.) Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, pp. 441–447. ACM (2011). DOI 10.1145/1966913.1966976
568. Morrissey, P., Smart, N.P., Warinschi, B.: A modular security analysis of the TLS handshake protocol. In: J. Pieprzyk (ed.) Advances in Cryptology - ASIACRYPT 2008, *Lecture Notes in Computer Science*, vol. 5350, pp. 55–73. Springer (2008)
569. Mu, Y., Varadharajan, V.: On the design of security protocols for mobile communications. In: J. Pieprzyk, et al. (eds.) Information Security and Privacy, First Australasian Conference, ACISP'96, *Lecture Notes in Computer Science*, vol. 1172, pp. 134–145. Springer (1996)
570. Nam, J., Choo, K.K.R., Kim, J., Kang, H.K., Kim, J., Paik, J., Won, D.: Password-only authenticated three-party key exchange with provable security in the standard model. The Scientific World Journal (2014). DOI 10.1155/2014/825072
571. Nam, J., Choo, K.K.R., Paik, J., Won, D.: Password-only authenticated three-party key exchange proven secure against insider dictionary attacks. The Scientific World Journal (2014). DOI 10.1155/2014/802359

572. National Institute of Standards and Technology: Escrowed Encryption Standard (EES) (1994). URL https://csrc.nist.gov/CSRC/media/Publications/fips/185/archive/1994-02-09/documents/fips185.pdf
573. National Institute of Standards and Technology: Entity Authentication Using Public Key Cryptography (1997). URL https://csrc.nist.gov/publications/detail/fips/196/archive/1997-02-18
574. National Institute of Standards and Technology: Advanced Encryption Standard (AES) (2001). DOI 10.6028/NIST.FIPS.197
575. National Institute of Standards and Technology: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC (2007). DOI 10.6028/NIST.SP.800-38D
576. National Institute of Standards and Technology: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography (2011). URL https://csrc.nist.gov/Projects/Key-Management/Key-Establishment
577. National Institute of Standards and Technology: Digital Signature Standard (DSS) (2013). DOI 10.6028/NIST.FIPS.186-4
578. National Institute of Standards and Technology: Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography (2014). DOI 10.6028/NIST.SP.800-56Br1
579. National Institute of Standards and Technology: Secure Hash Standard (2015). DOI 10.6028/NIST.FIPS.180-4
580. National Security Agency: SKIPJACK and KEA Algorithm Specification (1998). URL http://csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf
581. Needham, R., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Communications of the ACM **21**(12), 993–999 (1978)
582. Neuman, B.C., Ts'o, T.: Kerberos: An authentication service for computer networks. IEEE Communications Magazine **32**(9), 33–38 (1994)
583. Neupane, K., Steinwandt, R., Corona, A.S.: Scalable deniable group key establishment. In: J. García-Alfaro, et al. (eds.) Foundations and Practice of Security - 5th International Symposium, FPS 2012, *Lecture Notes in Computer Science*, vol. 7743, pp. 365–373. Springer (2012). DOI 10.1007/978-3-642-37119-6_24
584. Nyberg, K.: On one-pass authenticated key establshment schemes. In: Workshop on Selected Areas in Cryptography (SAC'95), pp. 2–8 (1995)
585. Nyberg, K., Rueppel, R.A.: A new signature scheme based on the DSA giving message recovery. In: 1st Conference on Computer and Communications Security, pp. 58–61. ACM Press (1993)
586. Nyberg, K., Rueppel, R.A.: Weaknesses in some recent key agreement protocols. Electronics Letters **30**(1), 26–27 (1994)
587. Nyberg, K., Rueppel, R.A.: Message recovery for signature schemes based on the discrete logarithm problem. In: A.D. Santis (ed.) Advances in Cryptology – EUROCRYPT '94, *Lecture Notes in Computer Science*, vol. 950, pp. 182–193. Springer (1995)
588. Ogata, K., Futatsugi, K.: Equational approach to formal analysis of TLS. In: 25th International Conference on Distributed Computing Systems (ICDCS), pp. 795–804. IEEE Computer Society (2005)
589. Okamoto, E.: Key distribution systems based on identification information. In: C. Pomerance (ed.) Advances in Cryptology – CRYPTO '87, *Lecture Notes in Computer Science*, vol. 293, pp. 194–202. Springer (1987)
590. Okamoto, E., Tanaka, K.: Key distribution system based on identification information. IEEE Journal on Selected Areas in Communications **7**(4), 481–485 (1989)

591. Okamoto, T.: Authenticated key exchange and key encapsulation in the standard model. In: K. Kurosawa (ed.) Advances in Cryptology - ASIACRYPT 2007, *Lecture Notes in Computer Science*, vol. 4833, pp. 474–484. Springer (2007). DOI 10.1007/978-3-540-76900-2_29
592. Okamoto, T., Tso, R., Okamoto, E.: One-way and two-party ID-based key agreement protocols using pairing. In: V. Torra, Y. Narukawa, S. Miyamoto (eds.) Modeling Decisions for Artificial Intelligence, Second International Conference, MDAI 2005, *Lecture Notes in Computer Science*, vol. 3558, pp. 122–133. Springer (2005)
593. van Oorschot, P.C.: An alternate explanation of two BAN-logic 'failures'. In: T. Helleseth (ed.) Advances in Cryptology – EUROCRYPT '93, *Lecture Notes in Computer Science*, vol. 765, pp. 443–447. Springer (1994)
594. van Oorschot, P.C., Wiener, M.J.: On Diffie-Hellman key agreement with short exponents. In: U. Maurer (ed.) Advances in Cryptology – EUROCRYPT '96, *Lecture Notes in Computer Science*, vol. 1070, pp. 332–343. Springer (1996)
595. OpenSSL: OpenSSL security advisory [07-Jan-2009] (2009). URL http://www.openssl.org/news/secadv_20090107.txt
596. Orman, H.: The OAKLEY Key Determination Protocol. RFC 2412 (Informational) (1998). DOI 10.17487/RFC2412. URL https://www.rfc-editor.org/rfc/rfc2412.txt
597. Otway, D., Rees, O.: Efficient and timely mutual authentication. ACM Operating Systems Review **21**(1), 8–10 (1987)
598. Pan, J., Wang, L.: TMQV: A strongly eCK-secure Diffie-Hellman protocol without gap assumption. In: X. Boyen, X. Chen (eds.) Provable Security - 5th International Conference, ProvSec 2011, *Lecture Notes in Computer Science*, vol. 6980, pp. 380–388. Springer (2011)
599. Pankova, A., Laud, P.: Symbolic analysis of cryptographic protocols containing bilinear pairings. In: S. Chong (ed.) 25th IEEE Computer Security Foundations Symposium, CSF 2012, pp. 63–77. IEEE Computer Society (2012). DOI 10.1109/CSF.2012.10
600. Park, C., Kurosawa, K., Okamoto, T., Tsujii, S.: On key distribution and authentication in mobile radio networks. In: T. Helleseth (ed.) Advances in Cryptology – EUROCRYPT '93, *Lecture Notes in Computer Science*, vol. 765, pp. 461–465. Springer (1994)
601. Park, S., Nam, J., Kim, S., Won, D.: Efficient password-authenticated key exchange based on RSA. In: M. Abe (ed.) Topics in Cryptology - CT-RSA 2007, *Lecture Notes in Computer Science*, vol. 4377, pp. 309–323. Springer (2007). DOI 10.1007/11967668_20
602. Pass, R.: On deniability in the common reference string and random oracle model. In: D. Boneh (ed.) Advances in Cryptology – CRYPTO 2003, *Lecture Notes in Computer Science*, vol. 2729, pp. 316–337. Springer (2003)
603. Patel, S.: Number theoretic attacks on secure password schemes. In: IEEE Symposium on Security and Privacy, pp. 236–247. IEEE Computer Society Press (1997)
604. Paterson, K.G., van der Merwe, T.: Reactive and proactive standardisation of TLS. In: L. Chen, et al. (eds.) Security Standardisation Research, Third International Conference, SSR 2016, *Lecture Notes in Computer Science*, vol. 10074, pp. 160–186. Springer (2016)
605. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the TLS record protocol. In: D.H. Lee, X. Wang (eds.) Advances in Cryptology - ASIACRYPT 2011, *Lecture Notes in Computer Science*, vol. 7073, pp. 372–389. Springer (2011)
606. Paulson, L.C.: Inductive analysis of the Internet protocol TLS. ACM Transactions on Information and System Security **2**(3), 332–351 (1999)

607. Paulson, L.C.: Relation between secrets: Two formal analyses of the Yahalom protocol. Journal of Computer Security **9**, 197–216 (2001)
608. Percival, C.: Cache missing for fun and profit (2005). URL http://www.daemonology.net/papers/htt.pdf
609. Pereira, O., Quisquater, J.J.: A security analysis of the Cliques protocols suites. In: 14th IEEE Computer Security Foundations Workshop, pp. 73–81. IEEE Computer Society Press (2001)
610. Perlman, R., Kaufman, C.: Key exchange in IPSec: Analysis of IKE. IEEE Internet Computing **4**(6), 50–56 (2000)
611. Perrig, A.: Efficient collaborative key management protocols for secure autonomous group communication. In: 1999 International Workshop on Cryptographic Techniques and Electronic Commerce, pp. 192–202. City University of Hong Kong Press (1999)
612. Phan, R.C.: Fixing the integrated Diffie-Hellman-DSA key exchange protocol. IEEE Communications Letters **9**(6), 570–572 (2005). DOI 10.1109/LCOMM.2005.1437374
613. Phan, R.C., Goi, B.: Cryptanalysis of the N-party encrypted Diffie–Hellman key exchange using different passwords. In: J. Zhou, et al. (eds.) Applied Cryptography and Network Security, 4th International Conference, ACNS 2006, *Lecture Notes in Computer Science*, vol. 3989, pp. 226–238 (2006). DOI 10.1007/11767480_15
614. Pieprzyk, J., Li, C.H.: Multiparty key agreement protocols. IEE Proceedings - Computers and Digital Techniques **147**(4), 229–236 (2000)
615. Pieprzyk, J., Wang, H.: Malleability attacks on multi-party key agreement protocols. Progress in Computer Science and Applied Logic **23**, 229–236 (2004)
616. Pointcheval, D.: Password-based authenticated key exchange. In: M. Fischlin, et al. (eds.) Public Key Cryptography - PKC 2012, *Lecture Notes in Computer Science*, vol. 7293, pp. 390–397. Springer (2012)
617. Pointcheval, D., Wang, G.: VTBPEKE: Verifier-based two-basis password exponential key exchange. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security, ASIACCS 2017. ACM (2017). DOI 10.1145/3052973.3053026
618. Pointcheval, D., Zimmer, S.: Multi-factor authenticated key exchange. In: S.M. Bellovin, et al. (eds.) Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, *Lecture Notes in Computer Science*, vol. 5037, pp. 277–295 (2008)
619. Popov, A.: Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard) (2015). DOI 10.17487/RFC7465. URL https://www.rfc-editor.org/rfc/rfc7465.txt
620. Prado, A., Harris, N., Gluck, Y.: SSL, gone in 30 seconds: A BREACH beyond CRIME. In: Black Hat USA 2013 (2013). URL https://www.blackhat.com/us-13/archives.html#Prado
621. Qualys SSL Labs: SSL Pulse (2018). URL https://www.ssllabs.com/ssl-pulse/
622. Rabin, M.O.: Digitalized signatures and public key functions as intractable as factorization. Tech. Rep. MIT-LCS-TR-212, MIT Laboratory for Computer Science (1979). URL https://apps.dtic.mil/dtic/tr/fulltext/u2/a078415.pdf
623. Ray, M., Dispensa, S.: Renegotiating TLS (2009). URL https://svn.dd-wrt.com/export/21663/src/router/matrixssl/doc/Renegotiating_TLS.pdf
624. Rescorla, E.: Keying Material Exporters for Transport Layer Security (TLS). RFC 5705 (Proposed Standard) (2010). DOI 10.17487/RFC5705. URL https://www.rfc-editor.org/rfc/rfc5705.txt

625. Rescorla, E.: Security impact of the Rizzo/Duong CBC "BEAST" attack (2011). URL http://www.educatedguesswork.org/2011/09/security_impact_of_the_rizzodu.html
626. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard) (2018). DOI 10.17487/RFC8446. URL https://www.rfc-editor.org/rfc/rfc8446.txt
627. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security. RFC 4347 (Proposed Standard) (2006). DOI 10.17487/RFC4347. URL https://www.rfc-editor.org/rfc/rfc4347.txt
628. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard) (2012). DOI 10.17487/RFC6347. URL https://www.rfc-editor.org/rfc/rfc6347.txt
629. Rescorla, E., Ray, M., Dispensa, S., Oskov, N.: Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard) (2010). DOI 10.17487/RFC5746. URL https://www.rfc-editor.org/rfc/rfc5746.txt
630. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
631. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: C. Boyd (ed.) Advances in Cryptology - ASIACRYPT 2001, *Lecture Notes in Computer Science*, vol. 2248, pp. 552–565. Springer (2001). DOI 10.1007/3-540-45682-1_32
632. Rizzo, J., Duong, T.: The CRIME attack (2012). URL http://goo.gl/mlw1X1. Presented at ekoparty '12
633. Roe, M., Christianson, B., Wheeler, D.: Secure sessions from weak secrets. Tech. rep., Computer Laboratory, University of Cambridge (1998). URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-445.html
634. Rogaway, P.: On the role definitions in and beyond cryptography. In: M.J. Maher (ed.) ASIAN, *Lecture Notes in Computer Science*, vol. 3321, pp. 13–32. Springer (2004). URL http://www.cs.ucdavis.edu/~rogaway/papers/def.html
635. Rogaway, P.: Evaluation of some blockcipher modes of operation. Tech. rep., Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan (2011). URL http://web.cs.ucdavis.edu/~rogaway/papers/modes.pdf
636. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall (1998)
637. Rueppel, R.A., van Oorschot, P.C.: Modern key agreement techniques. Computer Communications **17**(7), 458–465 (1994)
638. Ryan, P., Schneider, S.: Modelling and Analysis of Security Protocols. Addison-Wesley (2001)
639. Ryu, E.K., Yoon, E.J., Yoo, K.Y.: An efficient ID-based authenticated key agreement protocol from pairings. In: 3rd International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols - NETWORKING 2004, *Lecture Notes in Computer Science*, vol. 3042, pp. 1464–1469. Springer (2004)
640. Saeednia, S.: Improvement of Günther's identity-based key exchange protocol. Electronics Letters **36**(18), 1535–1536 (2000)
641. Saeednia, S.: A note on Girault's self-certified model. Tech. Rep. 2001/100, Cryptology ePrint Archive (2001). URL https://eprint.iacr.org/2001/100
642. Saeednia, S., Safavi-Naini, R.: Efficient identity-based conference key distribution protocols. In: C. Boyd, et al. (eds.) Information Security and Privacy – Third Australasian Conference, *Lecture Notes in Computer Science*, vol. 1438, pp. 320–331. Springer (1998)

643. Safford, D., Hess, D.K., Schales, D.L.: Texas A&M University anarchistic key authorization (AKA). In: Sixth USENIX Security Symposium (1996). URL https://www.usenix.org/legacy/publications/library/proceedings/sec96/safford.html
644. Saha, M., RoyChowdhury, D.: Provably secure key establishment protocol using one-way functions. Journal of Discrete Mathematical Sciences and Cryptography pp. 139–158 (2013)
645. Saint, E.L., Fedronic, D., Liu, S.: Open protocol for access control identification and ticketing with privacy specifications (2011). URL http://www.smartcardalliance.org/resources/pdf/OPACITY_Protocol-3-5-3.pdf
646. Sakai, R., Kasahara, M.: ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054 (2003). URL https://eprint.iacr.org/2003/054
647. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: Symposium on Cryptography and Information Security. Okinawa (2000)
648. Sakazaki, H., Okamoto, E., Mambo, M.: Constructing identity-based key distribution systems over elliptic curves. IEICE Transactions Fundamentals **E81-A**(10), 2138–2143 (1998)
649. Salowey, J., Zhou, H., Eronen, P., Tschofenig, H.: Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077 (Proposed Standard) (2008). DOI 10.17487/RFC5077. URL https://www.rfc-editor.org/rfc/rfc5077.txt
650. Sarkar, P.G., Fitzgerald, S.: Attacks on SSL: A comprehensive study of BEAST, CRIME, TIME, BREACH, Lucky 13, and RC4 biases (2013). URL https://www.isecpartners.com/media/106031/ssl_attacks_survey.pdf
651. Sarr, A.P., Elbaz-Vincent, P.: On the security of the (F)HMQV protocol. In: Progress in Cryptology – AFRICACRYPT 2016, *Lecture Notes in Computer Science*, vol. 9646, pp. 207–224. Springer (2016)
652. Sarr, A.P., Elbaz-Vincent, P., Bajard, J.: A secure and efficient authenticated Diffie-Hellman protocol. In: F. Martinelli, B. Preneel (eds.) Public Key Infrastructures, Services and Applications - EuroPKI 2009, *Lecture Notes in Computer Science*, vol. 6391, pp. 83–98. Springer (2009)
653. Sarr, A.P., Elbaz-Vincent, P., Bajard, J.C.: A new security model for authenticated key agreement. In: J.A. Garay, R.D. Prisco (eds.) Security and Cryptography for Networks, SCN 2010, *Lecture Notes in Computer Science*, vol. 6280, pp. 219–234. Springer (2010)
654. Satyanarayanan, M.: Integrating security in a large distributed system. ACM Transactions on Computer Systems **7**(3), 247–280 (1989)
655. Scheidler, R., Buchmann, J.A., Williams, H.C.: A key-exchange protocol using real quadratic fields. Journal of Cryptology **7**(3), 171–199 (1994)
656. Schmidt, B., Meier, S., Cremers, C.J.F., Basin, D.A.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: S. Chong (ed.) 25th IEEE Computer Security Foundations Symposium, CSF 2012, pp. 78–94. IEEE Computer Society (2012). DOI 10.1109/CSF.2012.25
657. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: G. Brassard (ed.) Advances in Cryptology – Crypto '89, *Lecture Notes in Computer Science*, vol. 435, pp. 239–252. Springer (1990)
658. Schridde, C., Smith, M., Freisleben, B.: An identity-based key agreement protocol for the network layer. In: R. Ostrovsky, et al. (eds.) Sixth Conference on Security and

Cryptography for Networks, *Lecture Notes in Computer Science*, vol. 5229, pp. 409–422. Springer (2008)
659. Scott, M.: Security of ID-based key exchange scheme. Electronics Letters **34**(7), 653–654 (1998)
660. Scott, M.: Authenticated ID-based key exchange and remote log-in with simple token and PIN number. Cryptology ePrint Archive, Report 2002/164 (2002). URL https://eprint.iacr.org/2002/164. Updated version December 2004
661. Seggelmann, R., Tuexen, M., Williams, M.: Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. RFC 6520 (Proposed Standard) (2012). DOI 10.17487/RFC6520. URL https://www.rfc-editor.org/rfc/rfc6520.txt
662. Sen Gupta, S., Maitra, S., Paul, G., Sarkar, S.: (Non-)Random sequences from (non-) random permutations – analysis of RC4 stream cipher. Journal of Cryptology **27**(1), 67–108 (2014). DOI 10.1007/s00145-012-9138-1
663. Seo, D.H., Sweeney, P.: Simple authenticated key agreement algorithm. Electronics Letters **35**(13), 1073–1074 (1999)
664. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
665. Shamir, A.: Identity-based cryptosystems and signature schemes. In: G.R. Blakley, D. Chaum (eds.) Advances in Cryptology, Proceedings of CRYPTO '84, *Lecture Notes in Computer Science*, vol. 196, pp. 47–53. Springer (1985)
666. Sheffer, Y., Holz, R., Saint-Andre, P.: Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). RFC 7457 (Informational) (2015). DOI 10.17487/RFC7457. URL https://www.rfc-editor.org/rfc/rfc7457.txt
667. Shieh, S.P., Yang, W.H., Sun, H.M.: An authentication protocol without trusted third party. IEEE Communications Letters **1**(3), 87–89 (1997)
668. Shim, K.: Efficient ID-based authenticated key agreement protocol based on Weil pairing. IEE Electronics Letters **39**, 653–654 (2003)
669. Shim, K.: Some attacks on Chikazawa-Yamagishi ID-based key sharing scheme. IEEE Communications Letters **7**(3), 145–147 (2003)
670. Shim, K.A.: Cryptanalysis of two ID-based authenticated key agreement protocols from pairings. Cryptology ePrint Archive, Report 2005/357 (2005). URL https://eprint.iacr.org/2005/357
671. Shimbo, A., Kawamura, S.: Cryptanalysis of several conference key distribution schemes. In: H. Imai, et al. (eds.) Advances in Cryptology – ASIACRYPT '91, *Lecture Notes in Computer Science*, vol. 739, pp. 265–276. Springer (1993)
672. Shin, S., Kobara, K.: Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2. RFC 6628 (Experimental) (2012). DOI 10.17487/RFC6628. URL https://www.rfc-editor.org/rfc/rfc6628.txt
673. Shin, S., Kobara, K., Imai, H.: Security proof of AugPAKE (2010). URL https://eprint.iacr.org/2010/334
674. Shoup, V.: On formal models for secure key exchange (1999). URL https://www.shoup.net/papers/skey.pdf
675. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004). URL https://eprint.iacr.org/2004/332
676. Shoup, V., Rubin, A.D.: Session key distribution using smart cards. In: U.M. Maurer (ed.) Advances in Cryptology - EUROCRYPT '96, *Lecture Notes in Computer Science*, vol. 1070, pp. 321–331. Springer (1996)

677. Simpson, W.A.: Photuris: Design criteria. In: H. Heys, et al. (eds.) Selected Areas in Cryptography, 6th International Workshop, *Lecture Notes in Computer Science*, vol. 1758, pp. 226–241. Springer (2000)
678. Smart, N.: Cryptography Made Simple. Springer (2015). DOI 10.1007/978-3-319-21936-3
679. Smart, N.P.: Identity-based authenticated key agreement protocol based on Weil pairing. Electronics Letters **38**(13), 630–632 (2002)
680. Smart, N.P.: Efficient key encapsulation to multiple parties. In: C. Blundo, S. Cimato (eds.) Security in Communication Networks, 4th International Conference, SCN 2004, *Lecture Notes in Computer Science*, vol. 3352, pp. 208–219. Springer (2005)
681. Smart, N.P., Siksek, S.: A fast Diffie-Hellman protocol in genus 2. Journal of Cryptology **12**(1), 67–73 (1999)
682. Smith, J., Nieto, J.M.G., Boyd, C.: Modelling denial of service attacks on JFK with Meadows's cost-based framework. In: R. Buyya, et al. (eds.) The proceedings of the Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006) and the Fourth Australasian Information Security Workshop (Network Security) (AISW 2006), *CRPIT*, vol. 54, pp. 125–134. Australian Computer Society (2006). DOI 10.1145/1151828.1151844
683. Smyshlyaev, S., Alekseev, E., Oshkin, I., Popov, V.: The Security Evaluated Standardized Password-Authenticated Key Exchange (SESPAKE) Protocol. RFC 8133 (Informational) (2017). DOI 10.17487/RFC8133. URL https://www.rfc-editor.org/rfc/rfc8133.txt
684. Smyth, B., Pironti, A.: Truncating TLS connections to violate beliefs in web applications. In: Black Hat USA (2013). URL http://www.bensmyth.com/files/Smyth13-truncation-attacks-to-violate-beliefs.pdf
685. Song, B., Kim, K.: Two-pass authenticated key agreement protocol with key confirmation. In: B. Roy, et al. (eds.) Progress in Cryptology – INDOCRYPT 2000, *Lecture Notes in Computer Science*, vol. 1977, pp. 237–249. Springer (2000)
686. Stebila, D., Kuppusamy, L., Rangasamy, J., Boyd, C., Nieto, J.M.G.: Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In: A. Kiayias (ed.) Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, *Lecture Notes in Computer Science*, vol. 6558, pp. 284–301. Springer (2011). DOI 10.1007/978-3-642-19074-2_19
687. Stebila, D., Udupi, P., Chang, S.: Multi-factor password-authenticated key exchange. In: Proceedings of the Eighth Australasian Conference on Information Security-Volume 105, pp. 56–66. Australian Computer Society (2010)
688. Steer, D.G., Strawczynski, L., Diffie, W., Wiener, M.: A secure audio teleconference system. In: S. Goldwasser (ed.) Advances in Cryptology – CRYPTO '88, *Lecture Notes in Computer Science*, vol. 403, pp. 520–528. Springer (1989)
689. Steiner, M.: Secure group key agreement. Ph.D. thesis, Universität des Saarlandes (2002). URL http://www.semper.org/sirene/publ/Stei_02.thesis-final.pdf
690. Steiner, M., Buhler, P., Eirich, T., Waidner, M.: Secure password-based cipher suite for TLS. ACM Transactions on Information and System Security **4**(2), 134–157 (2001)
691. Steiner, M., Tsudik, G., Waidner, M.: Refinement and extension of encrypted key exchange. ACM Operating Systems Review **29**(3), 22–30 (1995)
692. Steiner, M., Tsudik, G., Waidner, M.: Diffie-Hellman key distribution extended to group communication. In: 3rd ACM Conference on Computer and Communications Security, pp. 31–37. ACM Press (1996)

693. Steiner, M., Tsudik, G., Waidner, M.: Key agreement in dynamic peer groups. IEEE Transactions on Parallel and Distributed Systems **11**(8), 769–780 (2000)
694. Steinwandt, R., Corona, A.S.: Attribute-based group key establishment. IACR Cryptology ePrint Archive, Report 2010/235 (2010). URL https://eprint.iacr.org/2010/235
695. Stevens, M.: Attacks on hash functions and applications. Ph.D. thesis, Centrum Wiskunde & Informatica (CWI), Universiteit Leiden (2012). URL https://marc-stevens.nl/research/papers/PhD%20Thesis%20Marc%20Stevens%20-%20Attacks%20on%20Hash%20Functions%20and%20Applications.pdf
696. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-prefix collisions for MD5 and applications. International Journal of Applied Cryptography **2**(4), 322–359 (2012)
697. Stinson, D.R.: On some methods for unconditionally secure key distribution and broadcast encryption. Designs, Codes and Cryptography **12**, 215–243 (1997)
698. Stolbunov, A.: Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. Adv. in Math. of Comm. **4**(2), 215–235 (2010). DOI 10.3934/amc.2010.4.215
699. Stubblebine, S.G., Gligor, V.D.: On message integrity in cryptographic protocols. In: IEEE Symposium on Research in Security and Privacy, pp. 85–104. IEEE Computer Society Press (1992)
700. Stubblebine, S.G., Meadows, C.A.: Formal characterization and automated analysis of known-pair and chosen-text attacks. IEEE Journal on Selected Areas in Communications **18**(4), 571–581 (2000)
701. Sun, H.M., Hsieh, B.T.: Security analysis of Shim's authenticated key agreement protocols from pairings. Cryptology ePrint Archive, Report 2003/113 (2003). URL https://eprint.iacr.org/2003/113
702. Swanson, C., Jao, D.: A study of two-party certificateless authenticated key-agreement protocols. In: Progress in Cryptology - INDOCRYPT 2009, *Lecture Notes in Computer Science*, vol. 5922, pp. 57–71. Springer (2009)
703. Syverson, P.: A taxonomy of replay attacks. In: 7th IEEE Computer Security Foundations Workshop, pp. 187–191. IEEE Computer Society Press (1994)
704. Syverson, P.: Limitations on design principles for public key protocols. In: IEEE Symposium on Security and Privacy, pp. 62–72. IEEE Computer Society Press (1996)
705. Syverson, P., van Oorschot, P.C.: On unifying some cryptographic protocol logics. In: IEEE Symposium on Research in Security and Privacy, pp. 14–28. IEEE Computer Society Press (1994)
706. Szydlo, M., Kaliski Jr., B.S.: Proofs for two-server password authentication. In: A. Menezes (ed.) Topics in Cryptology - CT-RSA 2005, *Lecture Notes in Computer Science*, vol. 3376, pp. 227–244. Springer (2005). DOI 10.1007/978-3-540-30574-3_16
707. Tanaka, K., Okamoto, E.: Key distribution system for mail systems using ID-related information directory. Computers and Security **10**, 25–33 (1991)
708. Tatebayashi, M., Matsuzaki, N., Newman Jr., D.B.: Key distribution protocol for digital mobile communication systems. In: G. Brassard (ed.) Advances in Cryptology – Crypto '89, *Lecture Notes in Computer Science*, vol. 435, pp. 324–334. Springer (1989)
709. Teo, J.C.M., Choo, K.R.: Security improvements to anonymous ID-based group key agreement for wireless networks. In: S. Latifi (ed.) Seventh International Conference on Information Technology: New Generations, ITNG 2010, pp. 732–737. IEEE Computer Society (2010)

710. The Debian Project: Debian Security Advisory DSA-1571-1 openssl – predictable random number generator (2008). URL http://www.debian.org/security/2008/dsa-1571
711. Tian, H., Susilo, W., Ming, Y., Wang, Y.: A provable secure ID-based explicit authenticated key agreement protocol without random oracles. Journal of Computer Science and Technology **23**(5), 832–842 (2008)
712. Toorani, M.: Security analysis of J-PAKE. In: IEEE Symposium on Computers and Communications, ISCC 2014, pp. 1–6. IEEE Computer Society (2014). DOI 10.1109/ISCC.2014.6912576
713. Toorani, M.: Cryptanalysis of a robust key agreement based on public key authentication. Security and Communication Networks **9**(1), 19–26 (2016)
714. Tsai, Y.W., Hwang, T.: ID based public key cryptosystems based on Okamoto and Tanaka's ID based one way communication scheme. Electronics Letters **26**(10), 666–668 (1990)
715. Tseng, Y.M.: Weakness in simple authenticated key agreement protocol. Electronics Letters **36**(1), 48–49 (2000)
716. Tsudik, G., Herreweghen, E.V.: Some remarks on protecting weak keys and poorly-chosen secrets from guessing attacks. In: 12th Symposium on Reliable Distributed Sytems, pp. 136–142 (1993)
717. Tzeng, W.G., Tzeng, Z.J.: Round-efficient conference key agreement protocols with provable security. In: T. Okamoto (ed.) Advances in Cryptology – ASIACRYPT 2000, *Lecture Notes in Computer Science*, vol. 1976, pp. 614–627. Springer (2000)
718. Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Des. Codes Cryptography **46**(3), 329–342 (2008)
719. Ustaoglu, B.: Comparing *SessionStateReveal* and *EphemeralKeyReveal* for Diffie–Hellman protocols. In: J. Pieprzyk, F. Zhang (eds.) Provable Security, ProvSec 2009, *Lecture Notes in Computer Science*, vol. 5848, pp. 183–197. Springer (2009)
720. Vanhoef, M., Goethem, T.V.: HEIST: HTTP encrypted information can be stolen through TCP-windows. In: Black Hat USA (2016). URL https://tom.vg/papers/heist_blackhat2016.pdf
721. Vanhoef, M., Piessens, F.: All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In: 24th USENIX Security Symposium, pp. 97–112. USENIX Association (2015)
722. Venema, W., Orlando, M.: Vulnerability note VU#555316: STARTTLS plaintext command injection vulnerability (2011). URL http://www.kb.cert.org/vuls/id/555316
723. Viet, D., Yamamura, A., Tanaka, H.: Anonymous password-based authenticated key exchange. In: S. Maitra, et al. (eds.) Progress in Cryptology - INDOCRYPT 2005, *Lecture Notes in Computer Science*, vol. 3797, pp. 244–257. Springer (2005). DOI 10.1007/11596219_20
724. Wagner, D., Schneier, B.: Analysis of the SSL 3.0 protocol. In: Second USENIX Workshop on Electronic Commerce (1996). URL https://www.usenix.org/legacy/publications/library/proceedings/ec96/wagner.html
725. Wallner, D., Harder, E., Agee, R.: Key Management for Multicast: Issues and Architectures. RFC 2627 (Informational) (1999). DOI 10.17487/RFC2627. URL https://www.rfc-editor.org/rfc/rfc2627.txt
726. Wan, Z., Ren, K., Lou, W., Preneel, B.: Anonymous ID-based group key agreement for wireless networks. In: WCNC 2008, IEEE Wireless Communications & Networking Conference, March 31 2008 - April 3 2008, Las Vegas, Nevada, USA, Conference Proceedings, pp. 2615–2620. IEEE (2008)

727. Wang, S., Cao, Z., Bao, H.: Security of an efficient ID-based authenticated key agreement protocol from pairings. In: G. Chen, et al. (eds.) ISPA Workshops, *Lecture Notes in Computer Science*, vol. 3759, pp. 342–349. Springer (2005)
728. Wang, S., Cao, Z., Choo, K.K.R., Wang, L.: An improved identity-based key agreement protocol and its security proof. Inf. Sci. **179**(3), 307–318 (2009)
729. Wang, W., Hu, L.: Efficient and provably secure generic construction of three-party password-based authenticated key exchange protocols. In: R. Barua, T. Lange (eds.) Progress in Cryptology - INDOCRYPT 2006, *Lecture Notes in Computer Science*, vol. 4329, pp. 118–132. Springer (2006). DOI 10.1007/11941378_10
730. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199 (2004). URL https://eprint.iacr.org/2004/199
731. Wang, Y.: Efficient identity-based and authenticated key agreement protocol. Cryptology ePrint Archive, Report 2005/108 (2005). URL https://eprint.iacr.org/2005/108
732. Wang, Y.: Efficient identity-based and authenticated key agreement protocol. In: M.L. Gavrilova, C.K. Tan (eds.) Transactions on Computational Science XVII, *Lecture Notes in Computer Science*, vol. 7420, pp. 172–197. Springer (2013). DOI 10.1007/978-3-642-35840-1_9
733. Wei, F., Kumar, N., He, D., Yeo, S.S.: A general compiler for password-authenticated group key exchange protocol in the standard model. Discrete Applied Mathematics **241**, 78–86 (2018). DOI 10.1016/j.dam.2016.04.007
734. Williamson, M.J.: Non-secret encryption using a finite field. Tech. rep., GCHQ, UK (1974). URL https://www.gchq.gov.uk/non-secret-encryption-using-finite-field
735. Wong, C.K., Gouda, M., Lam, S.S.: Secure group communications using key graphs. ACM Computer Communication Review **28**(4), 68–79 (1998)
736. Woo, T.Y.C., Lam, S.S.: Authentication for distributed systems. IEEE Computer **25**(1), 39–52 (1992)
737. Woo, T.Y.C., Lam, S.S.: A lesson on authentication protocol design. ACM Operating Systems Review **28**(3), 24–37 (1994)
738. Wu, J., Ustaoglu, B.: Efficient key exchange with tight security reduction. Cryptology ePrint Archive, Report 2009/288 (2009). URL https://eprint.iacr.org/2009/288
739. Wu, Q., Mu, Y., Susilo, W., Qin, B., Domingo-Ferrer, J.: Asymmetric group key agreement. In: Advances in Cryptology - EUROCRYPT 2009, *Lecture Notes in Computer Science*, vol. 5479, pp. 153–170. Springer (2009)
740. Wu, T.: The secure remote password protocol. In: Network and Distributed System Security Symposium. Internet Society (1998). URL https://www.ndss-symposium.org/ndss1998/secure-remote-password-protocol/
741. Wu, T.: A real-world analysis of Kerberos password security. In: Network and Distributed System Security Symposium. Internet Society (1999). URL https://www.ndss-symposium.org/ndss1999/real-world-abstract-analysis-kerberos-password-security/
742. Wu, T.: SRP-6: Improvements and refinements to the secure remote password protocol. Tech. rep., Submissions to IEEE P1363.2 (2002). URL http://srp.stanford.edu/srp6.ps
743. Wu, T.Y., Tseng, Y.M.: An ID-based mutual authentication and key exchange protocol for low-power mobile devices. The Computer Journal **53**(7), 1062–1070 (2010)

744. Xie, G.: Cryptanalysis of Noel McCullagh and Paulo S. L. M. Barreto's two-party identity-based key agreement. Cryptology ePrint Archive, Report 2004/308 (2004). URL https://eprint.iacr.org/2004/308
745. Xie, G.: An ID-based key agreement scheme from pairing. Cryptology ePrint Archive, Report 2005/093 (2005). URL https://eprint.iacr.org/2005/093
746. Xu, J., Hu, X., Zhang, Z.: Round-optimal password-based group key exchange protocols in the standard model. In: T. Malkin, et al. (eds.) Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, *Lecture Notes in Computer Science*, vol. 9092, pp. 42–61. Springer (2015). DOI 10.1007/978-3-319-28166-7_3
747. Yacobi, Y.: Attack on the Koyama-Ohta identity based key distribution scheme. In: C. Pomerance (ed.) Advances in Cryptology – CRYPTO '87, *Lecture Notes in Computer Science*, vol. 293, pp. 429–433. Springer (1987)
748. Yacobi, Y.: A key distribution 'paradox'. In: A.J. Menezes, et al. (eds.) Advances in Cryptology – CRYPTO '90, *Lecture Notes in Computer Science*, vol. 537, pp. 268–273. Springer (1991)
749. Yamaguchi, S., Okayama, K., Miyahara, H.: Design and implementation of an authentication system in WIDE internet environment. In: IEEE Region 10 Conference on Computer and Communications Systems, pp. 653–657. Hong Kong (1990)
750. Yang, G., Tan, C.H.: Strongly secure certificateless key exchange without pairing. In: B.S.N. Cheung, et al. (eds.) ACM Symposium on Information, Computer and Communications Security – ASIACCS 2011, pp. 71–79. ACM (2011)
751. Yang, Z.: Efficient eCK-secure authenticated key exchange protocols in the standard model. In: S. Qing, et al. (eds.) Information and Communications Security - 15th International Conference, ICICS 2013, *Lecture Notes in Computer Science*, vol. 8233, pp. 185–193. Springer (2013). URL https://eprint.iacr.org/2013/365
752. Yang, Z.: Modelling simultaneous mutual authentication for authenticated key exchange. In: J.L. Danger, et al. (eds.) Foundations and Practice of Security - 6th International Symposium, FPS 2013, *Lecture Notes in Computer Science*, vol. 8352, pp. 46–62. Springer (2013). DOI 10.1007/978-3-319-05302-8_4
753. Yao, A., Zhao, Y.: Deniable internet key exchange. In: J. Zhou, M. Yung (eds.) Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, *Lecture Notes in Computer Science*, vol. 6123, pp. 329–348. Springer (2010). DOI 10.1007/978-3-642-13708-2_20
754. Yao, A.C., Zhao, Y.: OAKE: a new family of implicitly authenticated Diffie–Hellman protocols. In: ACM Conference on Computer and Communications Security, pp. 1113–1128. ACM (2013)
755. Yao, A.C., Zhao, Y.: Privacy-preserving authenticated key-exchange over internet. IEEE Trans. Information Forensics and Security **9**(1), 125–140 (2014). DOI 10.1109/TIFS.2013.2293457
756. Yarom, Y., Benger, N.: Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack (2014). URL https://eprint.iacr.org/2014/140. Cryptology ePrint Archive, Report 2014/140
757. Yen, S.M.: Cryptanalysis of an authentication and key distribution protocol. IEEE Communications Letters **3**(1), 7–8 (1999)
758. Yen, S.M., Liu, M.T.: High performance nonce-based authentication and key distribution protocols against password guessing attacks. IEICE Transactions Fundamentals **E80-A**(11), 2209–2217 (1997)
759. Yi, X., Hao, F., Bertino, E.: ID-based two-server password-authenticated key exchange. In: M. Kutylowski, J. Vaidya (eds.) 19th European Symposium on Research in Computer

Security, ESORICS 2014, *Lecture Notes in Computer Science*, vol. 8713, pp. 257–276. Springer (2014). DOI 10.1007/978-3-319-11212-1_15

760. Yi, X., Rao, F., Tari, Z., Hao, F., Bertino, E., Khalil, I., Zomaya, A.Y.: ID2S password-authenticated key exchange protocols. IEEE Trans. Computers **65**(12), 3687–3701 (2016). DOI 10.1109/TC.2016.2553031
761. Yilek, S., Rescorla, E., Shacham, H., Enright, B., Savage, S.: When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In: Proc. 9th ACM SIGCOMM Conference on Internet Measurement (IMC) 2009, pp. 15–27. ACM (2009). DOI 10.1145/1644893.1644896
762. Yoneyama, K.: Efficient and strongly secure password-based server aided key exchange (extended abstract). In: D.R. Chowdhury, et al. (eds.) Progress in Cryptology - INDOCRYPT 2008, *Lecture Notes in Computer Science*, vol. 5365, pp. 172–184. Springer (2008). DOI 10.1007/978-3-540-89754-5_14
763. Yoneyama, K.: Efficient and strongly secure password-based server aided key exchange. Journal of Information Processing **17**, 202–215 (2009)
764. Yoneyama, K.: Cross-realm password-based server aided key exchange. In: Y. Chung, M. Yung (eds.) Information Security Applications - 11th International Workshop, WISA 2010, *Lecture Notes in Computer Science*, vol. 6513, pp. 322–336. Springer (2010). DOI 10.1007/978-3-642-17955-6_24
765. Yoneyama, K.: Strongly secure two-pass attribute-based authenticated key exchange. In: M. Joye, et al. (eds.) Pairing-Based Cryptography - Pairing 2010, *Lecture Notes in Computer Science*, vol. 6487, pp. 147–166. Springer (2010)
766. Yoneyama, K.: One-round authenticated key exchange without implementation tricks. JIP **24**(1), 9–19 (2016). URL https://www.jstage.jst.go.jp/article/ipsjjip/24/1/24_9/_article
767. Yoneyama, K., Zhao, Y.: Taxonomical security consideration of authenticated key exchange resilient to intermediate computation leakage. In: X. Boyen, X. Chen (eds.) Provable Security - 5th International Conference, ProvSec 2011, *Lecture Notes in Computer Science*, vol. 6980, pp. 348–365. Springer (2011)
768. Youn, T., Park, Y., Kim, C.H., Lim, J.: Weakness in a RSA-based password authenticated key exchange protocol. Inf. Process. Lett. **108**(6), 339–342 (2008). DOI 10.1016/j.ipl.2008.06.002
769. Yuan, Q., Li, S.: A new efficient ID-based authenticated key agreement protocol. Cryptology ePrint Archive, Report 2005/309 (2005). URL https://eprint.iacr.org/2005/309
770. Zhang, F., Chen, X.: Attack on an ID-based authenticated group key agreement scheme from PKC 2004. Inf. Process. Lett. **91**(4), 191–193 (2004)
771. Zhang, L., Wu, Q., Qin, B., Domingo-Ferrer, J.: Identity-based authenticated asymmetric group key agreement protocol. In: M.T. Thai, S. Sahni (eds.) Computing and Combinatorics, 16th Annual International Conference, COCOON 2010, *Lecture Notes in Computer Science*, vol. 6196, pp. 510–519. Springer (2010)
772. Zhang, M.: Analysis of the SPEKE password-authenticated key exchange protocol. IEEE Communications Letters **8**(1), 63–65 (2004). DOI 10.1109/LCOMM.2003.822506
773. Zhang, Y., Wang, K., Li, B.: A deniable group key establishment protocol in the standard model. In: J. Kwak, et al. (eds.) Information Security, Practice and Experience, 6th International Conference, ISPEC 2010, *Lecture Notes in Computer Science*, vol. 6047, pp. 308–323. Springer (2010). DOI 10.1007/978-3-642-12827-1_23

774. Zhao, J., Gu, D., Gorantla, M.C.: Stronger security model of group key agreement. In: B.S.N. Cheung, et al. (eds.) Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, pp. 435–440. ACM (2011)
775. Zhao, S., Zhang, Q.: sHMQV: An efficient key exchange protocol for power-limited devices. In: J. Lopez, Y. Wu (eds.) Information Security Practice and Experience - 11th International Conference, ISPEC 2015, *Lecture Notes in Computer Science*, vol. 9065, pp. 154–167. Springer (2015). URL https://eprint.iacr.org/2015/110
776. Zhao, X., Zhang, F., Tian, H.: Dynamic asymmetric group key agreement for ad hoc networks. Ad Hoc Networks **9**(5), 928–939 (2011)
777. Zhao, Z., Dong, Z., Wang, Y.: Security analysis of a password-based authentication protocol proposed to IEEE 1363. Theor. Comput. Sci. **352**(1-3), 280–287 (2006). DOI 10.1016/j.tcs.2005.11.038
778. Zheng, M.H., Zhou, H., Li, J., Cui, G.: Efficient and provably secure password-based group key agreement protocol. Computer Standards & Interfaces **31**(5), 948–953 (2009). DOI 10.1016/j.csi.2008.09.021
779. Zheng, Y.: Digital signcryption or how to achieve cost (signature & encryption) << cost (signature) + cost (encryption). In: B.S. Kaliski Jr. (ed.) Advances in Cryptology – CRYPTO '97, *Lecture Notes in Computer Science*, vol. 1294, pp. 165–179. Springer (1997)
780. Zhou, J.: Fixing of security flaw in IKE protocols. Electronics Letters **35**(13), 1072–1073 (1999)
781. Zhou, J.: Further analysis of the Internet key exchange protocol. Computer Communications **23**, 1606–1612 (2000)
782. Zhou, L., Susilo, W., Mu, Y.: Efficient ID-based authenticated group key agreement from bilinear pairings. In: J. Cao, et al. (eds.) Mobile Ad-hoc and Sensor Networks, Second International Conference, MSN 2006, *Lecture Notes in Computer Science*, vol. 4325, pp. 521–532. Springer (2006)
783. Zimmermann, P.R.: The Official PGP User's Guide. MIT Press (1995)
784. Zuccherato, R.: Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME. RFC 2785 (Informational) (2000). DOI 10.17487/RFC2785. URL https://www.rfc-editor.org/rfc/rfc2785.txt

General Index

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9

Protocol Index

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9