Ling Dong
Kefei Chen

Cryptographic Protocol

Security Analysis Based on Trusted Freshness

高等教育出版社
HIGHER EDUCATION PRESS

Springer

Ling Dong
Kefei Chen

Cryptographic Protocol

Security Analysis Based on Trusted Freshness

Ling Dong
Kefei Chen

Cryptographic Protocol

Security Analysis Based on Trusted Freshness

With 185 figures

高等教育出版社
HIGHER EDUCATION PRESS

Springer

Authors
Ling Dong
Dept. of Computer Science and Engineering
Shanghai Jiaotong University
Shanghai, 200240, P.R. China
E-mail: ldong@sh163.net

Kefei Chen
Dept. of Computer Science and Engineering
Shanghai Jiaotong University
Shanghai, 200240, P.R. China
E-mail: kfchen@sjtu.edu.cn

ISBN 978-7-04-031331-4
Higher Education Press, Beijing

ISBN 978-3-642-24072-0 ISBN 978-3-642-24073-7 (eBook)
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011937024

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

A network protocol is a formal description of digital message formats, and of the rules for exchanging those messages in or between computing systems. A cryptographic protocol (also called a security protocol) is a kind of special network protocol that performs a security-related function and applies cryptographic methods.

Network protocols are typically developed in layers, with each layer responsible for a different aspect of communication; the security mechanisms are therefore embedded in layers. For instance, Transport Layer Security (TLS) is a well-known cryptographic protocol proposed for adding security services to TCP. In fact, cryptographic protocols are widely used for key establishment, entity authentication, message authentication, and secure data transport, or non-repudiation methods. Due to the asynchronous nature of communication, although there are only a few messages in a cryptographic protocol, the protocol may not be as secure as intended. For example, the Needham-Schroeder public-key authentication protocol was found to be flawed in 1995, seventeen years after its publication.

This book focuses on the security analysis of cryptographic protocols, introducing a novel idea for security analysis based on trusted freshness, and a formalism based on this freshness principle and beliefs multisets about security properties. It tries to answer the following questions:

— What does protocol security (computational and practical) mean?
— Is it possible to verify the security via engineering methods?
— How can protocol security verification be made easy or done automatically?

The book is intended as a reference for researchers, engineers and graduate students in computer security or cryptography. A lot of examples of analyzing cryptographic protocols are illustrated in the book. It is especially useful for both researchers and practitioners engaged in the area of designing, security analyzing, and engineering practice of network security protocols.

This book would not have been written without the support and help of many people. It is impossible for us to list all the people who contributed all the way.

First of all, the authors would like to thank colleagues as well as current

and former students of the authors for various forms of support. We want to thank Prof. Jiren Cai and Prof. Dingyi Pei for their encouragement to the publishing project, special thanks to Prof. Xuejia Lai for his valuable comments on an earlier version of the book. We gratefully acknowledge the excellent cooperation with Higher Education Press and Springer, and especially Mrs. Hongying Chen.

This book is supported in part by the National High-Tech Program (863) of China, the National Basic Research Program (973) of China, and the National Natural Science Foundation of China.

Shanghai Jiaotong University Ling Dong
April 2011 Kefei Chen

Contents

1 Introduction of Cryptographic Protocols

Abstract Cryptographic protocols are communication protocols which are designed to provide security assurances of various kinds, using cryptographic mechanisms. This chapter gives a brief introduction of cryptographic protocols and the reason why we study these protocols.

A protocol consists of a set of rules (conventions) which determine the exchange of messages between two or more participants. Cryptographic protocols, also called security protocols, use cryptographic primitives in communication protocols to provide information security, such as confidentiality, authentication, integrity or nonrepudiation, in an insecure network. Encryption schemes, digital signatures, hash functions, and random number generations are among the cryptographic primitives which may be utilized to build cryptographic protocols.

Example 1.1 (A cryptographic protocol) Alice is an initiator who wants to establish a secure session between herself and the responder Bob with the aid of a trusted third party Trent, as shown in Fig. 1.1. Alice seeks to establish this connection with Bob by selecting a nonce N_A at random and sending it to Trent, and Trent returns the nonce N_A along with a selected new session key k_{AB} encrypted under the long-term key K_{AS} (shared between Alice and

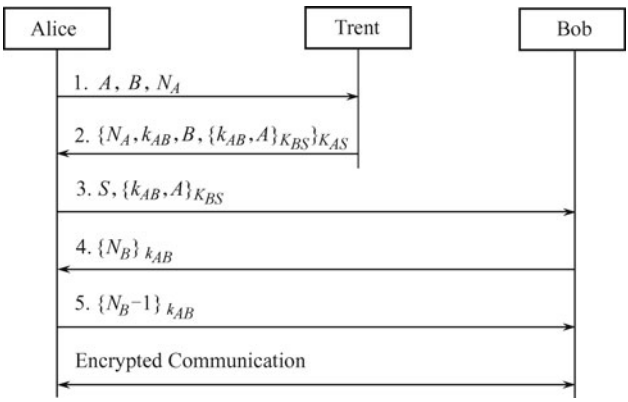


Fig. 1.1 Example of a cryptographic protocol.

Trent) and K_{BS} (shared between Bob and Trent) respectively. A successful run of this protocol does achieve the establishment of the shared key k_{AB} exclusively between Alice and Bob except Trent, then k_{AB} can be used for the subsequent communication between Alice and Bob.

1.1 Information security and cryptography

Over the ages, information was typically stored and transmitted on paper, whereas much of it now resides on magnetic media and is transmitted via computer networks. As we all know, it is much easier to copy and alter information stored and transmitted electronically than that on paper. Information security intends to provide security services for information in digital form. Information security objectives include confidentiality, data integrity, authentication, non-repudiation, access control, availability, fairness and so on. Computer and network security research and development focus on the first four general security services, from which other security services, such as access control, and fairness can be derived[1–5]. Many terms and concepts in this book are from Ref. [1] which is well addressed. For strict or inquisitive readers, please refer to book [1] for detailed information.

— Confidentiality is a service used to keep the content of information from all but those authorized to have it. That is, the information in a computer system or transmitted information cannot be comprehended by unauthorized parties. Secrecy is a term synonymous with confidentiality and privacy.
— Data integrity is a service which addresses the unauthorized modification of data. Modification includes creating, writing, deleting, changing, changing status, and delaying or replaying of transmitted messages.
— Authentication is a service related to identification, including entity authentication and data origin authentication. Entity authentication ensures that the identity of the party entering into a communication is not false. Data origin authentication ensures that the origin of information itself is not false. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed). In many applications, entity authentication is to allow resource usage to be tracked to identified entities.
— Non-repudiation is a service which prevents an entity from denying previous commitments or actions. A procedure involving a trusted third party is needed to resolve the dispute where an entity may deny that certain commitments were made or certain actions were taken. Commonly used fairness security in electronic commerce protocols can be derived from non-repudiation.
— Access control is a service which addresses the authorization of a party to access information resources. Only authorized parties may access the

information resources of the target system. To gain access to an information resource (e.g., computer account, printer, or software application), the user enters a (userid, password) pair, and explicitly or implicitly specifies a resource; here userid is a claim of identity, and password is the evidence supporting the claim. The system checks to see if the password matches corresponding data it holds for that userid, and if the stated identity is authorized to access the resource. Demonstration of knowledge of this secret (by revealing the password itself) is accepted by the system as corroboration of the entity's identity.

— Availability is a service which addresses the availability of the information resources, when they are needed, to authorized parties in a computer system.

— Fairness is a service to keep each honest protocol participant to have sufficient evidence (through acquisition of corroborative evidence) to solve the argumentation between or among parties, which may arise in or after a protocol run. It is the most important security service in a electronic commerce protocol.

Cryptography is to study mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. It is about the prevention and detection of cheating and other malicious activities. Cryptographic skills are the most common technical means of providing information security. Often the objectives of information security cannot solely be achieved through cryptographic primitives and protocols alone, but require procedural techniques and abidance of laws to achieve the desired security result[1].

1.2 Classes of cryptographic protocols

A cryptographic protocol is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective. Cryptographic protocols involving message exchanges require precise definition of both the messages to be exchanged and the actions to be taken by each party. Cryptographic protocols may be typically divided into four main categories, depending on the security objectives of the cryptographic protocol:

1.2.1 Authentication protocol

It is a protocol to provide one party some degree of assurance regarding the identity of another with whom it is purportedly communicating. An identification or an entity authentication technique assures one party (through

acquisition of corroborative evidence) of both: the identity of a second party involved, and that the second party was active at the time the evidence was created or acquired. Authentication protocol typically involves no meaningful messages other than the claim of being a particular entity. Authentication protocol could be broadly subdivided into unilateral entity authentication protocol, and mutual entity authentication protocol. Examples include Woo-Lam protocol[6], Zero knowledge proofs of identify[7] and Okamoto protocol[8].

— Unilateral entity authentication protocol, also called unilateral authentication protocol, is a protocol to assure a corroborated identity of a second party and that this party is active at the protocol run.
— Mutual entity authentication protocol, also called mutual authentication protocol, is a protocol to assure corroborated identities of both protocol parties and that they are active at the protocol run. Mutual authentication may be obtained by running any of the unilateral authentication mechanisms twice.

1.2.2 Key establishment protocol

It is a protocol to establish shared secrets, which are typically called or used to derive session keys. Key establishment is any process whereby a shared secret key becomes available to two or more parties for subsequent cryptographic use. Ideally, a session key is an ephemeral secret, i.e., the one whose use is restricted to a short time period such as a single telecommunications connection (or session), after which all trace of it is eliminated. While privacy of keying material is a requirement in key establishment protocols, source authentication is also typically needed. Encryption and signature primitives may respectively be used to provide these properties. Key establishment protocol can be broadly subdivided into key agreement protocol and key transport protocol.

— Key transport protocol is a key establishment technique with which one party creates or otherwise obtains a secret value, and securely transfers it to the other(s) as a session key.
— Key agreement protocol is a key establishment technique in which a session key is derived by two (or more) parties as a function of information contributed by or associated with each of these, (ideally) so that no party can predetermine the resulting value.
— Authenticated key establishment protocol is a protocol to establish a shared secret with a party whose identity has been (or can be) corroborated. Examples include Needham-Schroeder public-key protocol[9], IKE (Internet Key Exchange) protocol[10], Kerberos authentication protocol[11], X.509 protocol[12], DASS (Distributed Authentication Security Protocol)[13], etc.

1.2.3 Electronic commerce protocol

It is a protocol to provide secure electronic trades over network for two (or more) parties. The focuses of electronic commerce protocols are fairness and non-repudiation. Examples include SET (Secure Electronic Transaction)[14], IKP (Internet Keyed Payments)[15], etc.

1.2.4 Secure multi-party protocol

It is a protocol to assure secure collaborated run of computation for any parties of the protocol run in a distributed system. Examples include group key exchange protocols, multi-party authentication protocols, electronic vote protocols over net, electronic bid protocols, electronic cash protocols etc.

In the literature of cryptographic protocols, authentication protocols are commonly used to refer to both authentication protocols and key establishment protocols, and this is the case in this book.

1.3 Security of cryptographic protocols

An active adversary (perhaps by co-working with his friends distributed over an open communication network) is capable of intercepting, modifying, or injecting messages, and is good at doing so by impersonating other protocol principals. Even in the existence of active adversaries and communication errors, a secure cryptographic protocol should meet all claimed objectives. As for a key establishment protocol, this should include being operational, providing both secrecy and authenticity of the key, and being resilient. A key establishment protocol is operational (or compliant) if, in the absence of active adversaries and communications errors, honest participants who comply with its specification always complete the protocol having computed a common key and having knowledge of the identities of the parties with whom the key is shared. Key authenticity implies that the identities of the parties sharing the key are understood and corroborated, thus addressing impersonation and substitution. A key establishment protocol is resilient if it is impossible for an active adversary to mislead honest participants as to the final outcome[1].

Cryptographic protocols, such as authentication or authenticated key-establishment protocols, are difficult to design and debug. For example, IEEE 802.11 wired equivalent privacy (WEP) protocol[16], which is used to protect link-layer communications from eavesdropping and other attacks, has several serious security flaws. Anomalies and shortcomings have also been discovered in standards and proposed standards for Secure Sockets Layer[17], the later

IEEE 802.11i wireless authentication protocols[18], Kerberos[11], and others.

A successful attack on an authentication or authenticated key establishment protocol usually does not refer to breaking a cryptographic algorithm, e.g., via complexity theory-based cryptanalysis technique. Instead, it usually refers to adversary's unauthorized and undetected acquisition of cryptographic credential or nullification of cryptographic service without breaking a cryptographic algorithm. Of course, this is due to an error in protocol design, not the one in the cryptographic algorithm[3].

Here is a taxonomy of the possible attack types: message replay attack, man-in-the-middle attack, parallel session attack, reflection attack, interleaving attack, attack due to type flaw, attack due to name omission, attack due to misuse of cryptographic services, etc. We cannot exhaust all possible types of attacks even we could list all known attacks or attacks we can imagine, since the ability of an adversary is always developing. Furthermore, viewing from a lower-layer (the network layer) communication protocol, it actually does not require very sophisticated techniques for an adversary to mount various types of attacks. Hence, cryptographic protocols, especially authenticated protocols and key establishment protocols, are readily to contain security flaws, even under the great care of experts in the field.

The main objective of this section is to show the delicate nature of cryptographic protocols, especially authenticated protocols and key establishment protocols.

Example 1.2 (Needham-Schroeder public-key protocol) The Needham-Schroeder public-key protocol[9] provides mutual entity authentication and key transport for both parties, as shown in Fig. 1.2. The transported symmetric keying materials N_A and N_B may serve both as nonces for entity authentication and session key parts for further secure communication use. Combination of the resulting shared key parts allows computation of a joint key to which both parties contribute.

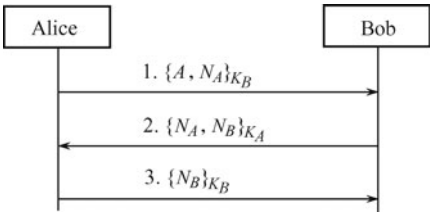


Fig. 1.2 Needham-Schroeder public-key protocol.

Notation

$\{Y\}_{K_X}$ denotes public-key encryption (e.g., RSA) of data Y using party X's public key K_X; $\{Y_1, Y_2\}_{K_X}$ denotes the encryption of the concatenation of Y_1 and Y_2. A stands for Alice, B for Bob, and I for the active adversary Malice. N_A and N_B are secret symmetric keying materials chosen by Alice and Bob, respectively.

Premise

K_A is Alice's public key, K_B is Bob's public key, and Alice and Bob possess each other's authentic public key. If this is not the case, while each party has an authentic certificate carrying its own public key, one additional message is required for certificate transport.

Actions

1) A randomly chooses a nonce N_A and sends B Message 1.

2) B recovers N_A upon receiving Message 1, and randomly chooses a nonce N_B and returns Message 2 to A.

3) Upon decrypting Message 2, A checks if the key material N_A recovered agrees with that sent in Message 1. (Provided N_A has never been previously used, this gives A both entity authentication of B and assurance that B knows this key). A sends B Message 3.

4) Upon decrypting Message 3, B checks if the key N_B recovered agrees with that sent in Message 2. A session key can be computed as $f(N_A, N_B)$ using an appropriate publicly known non-reversible function f.

Thus, a successful run of this protocol does achieve the establishment of the symmetric keying materials N_A and N_B, which are shared secrets exclusively between Alice and Bob. Further notice that since both parties contribute to these shared secrets recently, they are confident about the freshness of N_A and N_B. A and B trust the randomness of the secrets N_A and N_B since they are from a large space, which can be used to initialize a shared secret key $f(N_A, N_B)$ for subsequent secure communication between Alice and Bob.

Unfortunately, the Needham-Schroeder public-key protocol is vulnerable to an attack discovered by Lowe in 1995[19]. In the attack of Low, Malice intercepts the messages sent by (or to) Alice and Bob in Messages 1, 2, and 3, and replaces them with his own version. The following example is the attack.

Example 1.3 (Attack on Needham-Schroeder public-key protocol) The attack (see Fig. 1.3) involves two simultaneous runs of the protocol. In the first

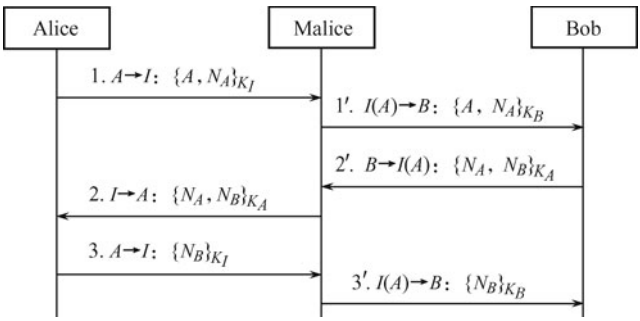


Fig. 1.3 Attack on Needham-Schroeder public-key protocol.

run (steps 1, 2, 3), Alice establishes a valid session with Malice; in the second run (steps $1'$, $2'$, $3'$), Malice establishes a bogus session with Bob by impersonating Alice. At last, Bob believes that Alice (in deed, it is Malice) has correctly established a session with him and they shared exclusively the secret nonces N_A and N_B.

Premise

Suppose that Malice has the public keys of all the protocol participants in his possession.

Actions

1) In step 1, Alice starts to establish a normal session with Malice, sending him a nonce N_A.

2) In step $1'$, via replaying the nonce N_A to Bob, Malice tries to establish a bogus session with Bob by impersonating Alice.

3) In step $2'$, Bob responds to Alice (Malice indeed) by selecting a new nonce N_B and returning it back along with N_A. Malice intercepts this message, but he cannot decrypt it because it is encrypted under Alice's public key.

4) In step 2, Malice therefore seeks to use Alice's private key to do the decryption for him, by forwarding the message to Alice; note that this message is of the form expected by Alice in the first run of the protocol.

5) In step 3, Alice decrypts the message to obtain N_B, and returns N_B to Malice (encryption under Malice's public key), thus completing the first run of the protocol.

6) In step $3'$, Malice decrypts Message 3 to obtain N_B using Malice's private key, and then constructs Message $3'$ and sends it to Bob by impersonating Alice, thus completing the second run of the protocol.

We can imagine the following consequences of this attack. Malice may include the shared nonces which suggest a session key within a subsequent message, and Bob will believe that the encrypted message using this session key originates from Alice.

Example 1.4 (Revised Needham-Schroeder public-key protocol) Figure 1.4 illustrates a revised Needham-Schroeder public-key protocol which is designed to enhance the security by indicating Alice's identity in Message 3. Alice assures Bob that N_A and N_B are exclusively symmetric keying ma-

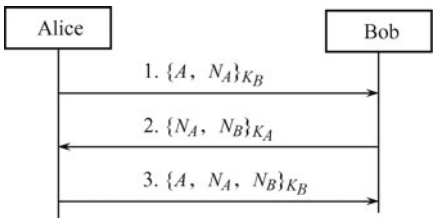


Fig. 1.4 A revise on Needham-Schroeder public-key protocol.

terials between Alice and Bob by explicitly indicating Alice's identity and encrypting Message 3 using Bob's public key.

Actions

1) Upon receiving Message 2, Alice should be assured that she is talking to Bob, since only Bob could be able to decrypt Message 1 to obtain N_A and this must have been done after her action of sending the nonce N_A out (a recent action).

2) Upon receiving Message 3, Bob should be assured that he is talking to Alice, since only Alice could be able to decrypt Message 2 to obtain N_B (a recent action). Bob should also be assured that N_A and N_B are exclusively symmetric keying materials between Alice and Bob since they are transmitted with the explicit identity of Alice, and only Bob could decrypt Message 3.

However, the revised protocol with security enhanced is not secure indeed. It could also be compromised by the attack discovered by Lowe[20].

Example 1.5 (Attack on the revised Needham-Schroeder public-key protocol) The new attack involves two simultaneous runs of the protocol, as shown in Fig. 1.5. In the first run (steps 1, 2, 3), Alice establishes a valid session with Malice; in the second run (steps $1'$, $2'$, $3'$) Malice tries to establish a bogus session with Bob by impersonating Alice. At last, Bob believes that Alice has correctly established a session with himself and they shared exclusively the nonces N_A and N_B. However, N_A and N_B are known by Malice.

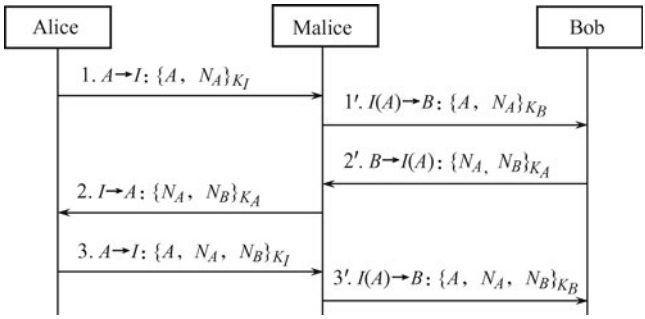


Fig. 1.5 Attack on the revised Needham-Schroeder public-key protocol.

Cryptographic protocols often comprise only a few messages, and protocol construction might seem a simple task. However this is clearly deceptive in practice, and the example we have shown above is an illustration.

1.4 Motivations of this book

Authentication protocols are the most commonly used cryptographic protocols and they are important in real world applications. Many and various

protocols have been proposed to provide authentication and key establishment security[1–18]. Although many of these protocols may seem relatively simple, in comparison with more complex distributed systems, they are subtle to design and very easily compromised, as we have witnessed in the above section.

Furthermore, authentication protocols are not only notoriously error-prone, and the flaws of these protocols are very difficult to detect. The current version of the Internet key exchange (IKE) protocol for Internet security is proved to be not secure as it promised, even after many years' protocol development by the committee of highly experienced computer security experts[21]. Many protocols have shown to be flawed even a long time after they were published. For example, the Needham-Schroeder public-key authentication protocol was found flawed by Lowe in 1995, seventeen years after its publication[9, 19].

The question of whether the security of an authentication protocol or an authenticated key establishment protocol is adequate has been extensively studied, with a large body of approaches proposed, including [20, 22 – 34] etc., and these approaches have played a very important role in protocol security analysis. While this book will introduce a new idea, which is more operational in practice, on how to uncover flaws in cryptographic protocols, the uncovering procedure can be done in a short time, by even a communication engineer without deep cryptographic knowledge background.

In this book, we will discuss the topic of the security of cryptographic protocols, especially that of authentication protocols. Our study of cryptographic protocol security in this book covers a wide range of topics in the subject with in depth discussions. Especially, we will put forward a novel idea, security analysis based on trusted freshness, which will indicate when a cryptographic protocol is secure, why a cryptographic protocol is flawed, and how to achieve the security of a cryptographic protocol. Security analysis based on trusted freshness is a new idea but not only a concrete means or formalism for analyzing the security of a cryptographic protocol. While it is an operational idea which can be easily employed by even a communication engineer without deep cryptographic background, and it can be utilized by information security researchers to invent systematic approaches (i.e., formal methods) for developing correct cryptographic protocols, or to invent formal approaches and automation tools for analyzing the security of existing cryptographic protocols.

This book includes 9 chapters. In Chapter 2, we will introduce some background knowledge of cryptography related to cryptographic protocols, and we will further study the principles to design cryptographic protocols in Chapter 3. Informal security analysis mechanisms, and reasons why taxonomy attacks on authentication protocol exist are in Chapter 4, and case studies of several protocols for real world applications are in Chapter 5. Formal definition of some security properties, formalism approaches of protocol security analysis, and design approaches to the development of correct authentication proto-

cols are in Chapters 6, 7, and 8. Chapter 9 introduces automated verification approaches to authentication protocols.

References

[1] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[2] Goldreich O (2003) Foundations of Cryptography. Cambridge University Press, New York

[3] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[4] Stallings W (2006) Cryptography and Network Security: Principles and Practice, 4th edn. Prentice Hall, New Jersey

[5] Stinson DR (2003) Cryptography: Theory and Practice, 2nd edn. CRC Press, New York

[6] Woo TYC, Lam SS (1992) Authentication for Distributed Systems. Computer 25(1): 39 – 52

[7] Feige U, Fiat A, Shamir A (1987) Zero Knowledge Proofs of Identify. In: STOC'87 Proceedings of the Nineteenth Annual ACM symposium on Theory of computing, New York, 25 – 27 May 1987

[8] Okamoto T (1993) Provably Secure and Practical Identification Schemes and Corresponding Signature Scheme. In: CRYPTO'92 Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara 16 – 20 Aug 1992. Lecture notes in computer science, vol 740, pp 31 – 53, Springer

[9] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[10] Harkins D, Carrel D (1998) The Internet Key Exchange Protocol (IKE), RFC 2409. http://www.ietf.org/rfc/rfc2409.txt. Accessed Dec 2005

[11] Miller SP, Neuman BC, Schiller JI, Saltzer JH (1987) Kerberos Authentication and Authorization System. Paper Presented at the Project Athena Technical Plan Section E.2.1. MIT, Boston

[12] CCITT (1987) CCITT Draft Recommendation X.509. The Directory-Authentication Framework (Version 7), New York

[13] Kaufman C (1993) Distributed Authentication Security Service, RFC 1507. http://www.ietf.org/rfc/rfc1507.txt. Accessed Sept 1993

[14] SET. Secure Electronic Transaction. The SET Standard Specification. http://www.setco.org/set-specifications. Accessed May 1997

[15] IBM Zurich Laboratory (1995) Internet Keyed Payments Protocol (IKP). http://www.zurich.ibm.com /Technology/Security/extern/ecommerce/spec. Accessed 30 June 1995

[16] ANSI/IEEE Std 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Sept 1999

[17] Freier AO, Karlton P, Kocher PC (1996) The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt. Accessed 18 Nov 1996

[18] IEEE Std 802.11i-2004. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Security Enhancements. July 2004

[19] Lowe G (1995) An Attack on the Needham-Schroeder Public Key Authentication Protocol. Information Processing Letters 56(3): 131 – 133

[20] Lowe G (1996) Breaking and Fixing the Needham-Schroeder Public-key Protocol Using FDR. In: TACAS'96 Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Passau, 27 – 29 Mar 1996. Lecture Notes in Computer Science (Lecture Notes in Software Configuration Management), vol 1055, pp 147 – 166, Springer

[21] Kaufman C (2005) Internet Key Exchange (IKEv2) Protocol, RFC 4306. http://tools.ietf.org/html/rfc4306. Accessed Dec 2005

[22] Canetti R, Krawczy H (2001) Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In: EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045, pp 453 – 474, Springer

[23] Burrows M, Abadi M, Needham R (1990) A Logic of Authentication. ACM Transactions on Computer Systems 8(1): 18 – 36

[24] Gong L, Needham R, Yahalom R (1990) Reasoning About Belief in Cryptographic Protocols. In: Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, Oakland, 7 – 9 May 1990

[25] Syverson PF, Oorschot PCV (1994) On Unifying Some Cryptographic Protocol Logics. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Oakland, 16 – 18 May 1994

[26] Lowe G (1999) Towards a Completeness Result for Model Checking of Security Protocols. Journal of Computer Security 7(2 – 3): 89 – 146

[27] Goldwasser S, Micali S (1984) Probabilistic Encryption. Journal of Computer and System Sciences 28(2): 270 – 299

[28] Bellare M, Rogaway P (1993) Entity Authentication and Key Distribution. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 773, pp 232 – 249, Springer

[29] Blanchet B (2006) A Computationally Sound Mechanized Prover for Security Protocols. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Berkeley/Oakland, 21 – 24 May 2006

[30] Datta A, Derek A, Mitchell JC, Roy A (2007) Protocol Composition Logic (PCL). Electronic Notes in Theoretical Computer Science 172: 311 – 358

[31] Canetti R, Herzog J (2006) Universally Composable Symbolic Analysis of Mutual Authentication and Key-exchange Protocols. In: Theory of Cryptography Conference Proceedings of TCC2006, New York, 4 – 7 Mar 2006

[32] Qing SH (2003) Design and Logical Analysis of Security Protocols. Journal of Software, 14(7): 1301 – 1309 (in Chinese)

[33] Qing SH (2003) Twenty Years Development of Security Protocols Research. Journal of Software 14(10): 1740 – 1752 (in Chinese)

[34] Feng DG, Fan H (2003) Security Protocol Theory and Method. Science Press, Beijing (in Chinese)

2 Background of Cryptographic Protocols

Abstract Some background knowledge including preliminary knowledge, cryptographic primitive knowledge, cryptographic protocol knowledge, cryptographic protocol security knowledge, and communication threat model knowledge are briefly introduced.

A brief introduction of background knowledge on cryptographic protocols is given in this chapter. Readers who are familiar with cryptography knowledge can skip this chapter without causing difficulty for reading the rest parts of this book, while inquisitive readers can refer to Refs. [1, 2] for sufficient reference materials.

2.1 Preliminaries

This section gives some preliminary knowledge of cryptographic protocols including functions, one way transformation, message space, adversary, etc.

2.1.1 Functions

Definition 2.1 A set consists of distinct objects which are called elements of the set. For example, a set X might consist of the elements a, b, c, and this is denoted by $X = \{a, b, c\}$.

Definition 2.2 A function (or transformation) is defined by two sets X and Y, and a rule f which assigns to each element in X precisely one element in Y. The set X is called the domain of the function and Y the codomain. If x is an element of X (usually written $x \in X$), the image of x is the element in Y for which the rule Y associates with x; the image y of x is denoted by $y = f(x)$. Standard notation for a function f from set X to set Y is $f : X \to Y$. If $y \in Y$, then a preimage of y is an element $x \in X$ for which $f(x) = y$. The set of all elements in Y which have at least one preimage is called the image of f, denoted by $\mathrm{Im}(f)$.

Definition 2.3 A function f from a set X to a set Y is called a one-way function if $f(x)$ is "easy" to compute for all $x \in X$ but for "essentially all" elements $y \in \mathrm{Im}(f)$ it is "computationally infeasible" to find any $x \in X$ so that $f(x) = y$.

Here and elsewhere, the terms easy and computationally infeasible (or hard) are intentionally left without formal definition; it is intended they are intended to be interpreted relative to an understood frame of reference. Easy might mean polynomial time and space, or more practically, within a certain number of machine operations or time units — perhaps seconds or milliseconds. A more specific definition of computationally infeasible might involve super-polynomial effort, require effort far exceeding understood resources, specify a lower bound on the number of operations or memory required in terms of a specified security parameter, or specify that the probability that a security property is violated is exponentially small.

Definition 2.4 A function (or transformation) is injective if each element in the codomain Y is the image of at most one element in the domain X.

Definition 2.5 A function (or transformation) is onto if each element in the codomain Y is the image of at least one element in the domain X. Equivalently, a function $f : X \to Y$ is onto if $\mathrm{Im}(f) = Y$.

Definition 2.6 If a function $f : X \to Y$ is injective and $\mathrm{Im}(f) = Y$, then f is called a bijection.

Definition 2.7 A trapdoor one-way function is a one-way function $f : X \to Y$ with the additional property that gives some extra information (called the trapdoor information) and it becomes feasible to find, for any given $y \in \mathrm{Im}(f)$, an $x \in X$ so that $f(x) = y$.

Definition 2.8 Let S be a finite set and let f be a bijection from S to S (i.e., $f : S \to S$). The function f is called an involution if $f = f^{-1}$. An equivalent way of stating this is $f(f(x)) = x$ for all $x \in S$.

2.1.2 Terminology

Definition 2.9 A denotes a finite set called the alphabet of definition. For example, $A = \{0, 1\}$, the binary alphabet, is a frequently used alphabet of definition.

Definition 2.10 M denotes a set called the message space. M consists of strings of symbols from an alphabet of definition. An element of M is called a plaintext message or simply a plaintext. For example, M may consist of binary strings, English texts, computer codes, etc.

Definition 2.11 C denotes a set called the ciphertext space. C consists of strings of symbols from an alphabet of definition, which may differ from the alphabet of definition for M. An element of C is called a ciphertext.

Definition 2.12 K denotes a set called the key space. An element of K is called a key. In this book, we usually use upper case K as a long-term key, and lower case k as a temporal session key.

Definition 2.13 An encryption scheme consists of a set $E_e : e \in K$ of encryption transformations (encryption algorithm) and a corresponding set $D_d : d \in K$ of decryption transformations (decryption algorithm) with the property that for each $e \in K$ there is a unique key $d \in K$ so that $D_d = E_e^{-1}$, that is, $D_d(E_e(m)) = m$ for all $m \in M$.

Definition 2.14 The keys e and d in the preceding definition are referred to as a key pair and sometimes denoted by (e, d). Note that e and d could be the same.

Public key e, and private key d are paired keys in a public-key system; symmetric keys e and d ($e = d$) are equal keys in a symmetric-key (single-key) system.

A fundamental premise in cryptography is that the sets $M, C, K, \{E_e : e \in K\}, \{D_d : d \in K\}$ are public knowledge, and the only thing to keep secret is the particular key pair (e, d) that is being used.

The other premise in cryptography a priori is that for point-to-point mechanisms, parties A and B shared a secret key, or A and B each shares a secret key with a trusted party T in a symmetric-key mechanism; A and B know the opponent parties' public key, or A and B both have the knowledge of the public key of a trusted party T in a public-key mechanism.

These shared long-term keys are initially established by non-cryptographic, and out-of-band techniques providing confidentiality and authenticity (e.g., in person, or by trusted courier).

The objective of attacks is to confuse a run of a protocol, to systematically recover plaintext from ciphertext, or even more drastically, to deduce the decryption key.

Definition 2.15 An adversary is an unauthorized "third" party in a communication who tries to defeat the information security service provided between the sender and receiver. Various other names are synonymous with adversary such as enemy, attacker, opponent, tapper, eavesdropper, intruder, saboteur, and interloper. We usually call the adversary "Malice" in this book.

An adversary will often attempt to play the role of either the legitimate sender or the legitimate receiver.

Definition 2.16 A passive adversary is an adversary who is only capable of reading information from an unsecured channel.

Definition 2.17 An active adversary is an adversary who is capable of not only reading information, but also transmitting, altering, or deleting information on an unsecured channel.

In a practical cryptographic setting, it is prudent to make the assumption that the adversary is an active adversary, and it is very powerful.

Definition 2.18 A passive attack is the one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data.

Passive attacks are in the nature of eavesdropping on, or monitoring of transmitted messages. Two classical types of passive attacks are release of message contents and traffic analysis. Passive attacks are very difficult to detect because they do not involve any alternation of data. However, it is feasible to prevent passive attacks.

Definition 2.19 An active attack is one where the adversary attempts to delete, add, or in some other way alter the transmission on the channel besides the monitoring of transmitted messages. An active attacker threatens data integrity and authentication as well as confidentiality.

Active attacks involve some replay, creation, insertion, deletion or modification of transmissions, masquerade of entity, and denial of service.

2.2 Cryptographic primitives

This section gives a brief introduction to some cryptographic knowledge including primitive, encryption, signature, identification, symmetric-key system etc.

2.2.1 Cryptology

Definition 2.20 An algorithm or a primitive is a well-defined computational procedure that takes a variable input and halts with an output.

Examples of primitives include encryption schemes, hash functions, and digital signature schemes. Primitives, when applied in various ways and with various inputs, will typically exhibit different characteristics.

Definition 2.21 A cryptosystem is a general term referring to a set of cryptographic primitives used to provide information security services. Most often the term is used in conjunction with primitives providing confidentiality, i.e., encryption. An encryption algorithm and a decryption algorithm plus the description of the format of messages and keys form a cryptographic system

or a cryptosystem.

Definition 2.22 Cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques, and, more generally, information security services.

Computational complexity is one of the most important foundations for modern cryptography.

Definition 2.23 A polynomial-time algorithm, also called efficient algorithm or a good algorithm, is an algorithm whose worst-case running time function could be expressed by a polynomial in the size of the input. That is, the algorithm is deterministic or randomized with polynomial execution time. Thus, informally speaking, a computational problem is said to be easy or tractable if there exists a good or efficient algorithm to solve the problem. Any algorithm whose running time cannot be so bounded by polynomial execution time is called an exponential-time algorithm. Thus, informally speaking, a computational problem is said to be hard or intractable if there doesn't exist an efficient algorithm to solve the problem.

Note that there are, however, some practical situations when this distinction is not appropriate for average-case complexity is more important than worst-case complexity in cryptography – a necessary condition for an encryption scheme to be considered secure is that the corresponding cryptanalysis problem is difficult on average (or more precisely, almost always difficult), and not just for some isolated cases[1].

Definition 2.24 Non-deterministic polynomial-time (NP) problems are those in which we couldn't find an efficient algorithm to solve in polynomial time. Many cryptographic primitives are based on the difficulty of these intractable problems, also called hard problems, such as the integer factorization problem and the discrete logarithm problem. Some hard problems in computational complexity theory can provide a high confidence in the security of a cryptographic algorithm or protocol.

Clearly, a cryptographic algorithm must be designed so that it is tractable for a legitimate user, but is intractable for a non-user or an attacker and so constitutes a difficult problem to solve. Here, "intractable" means that the widely available computational methods cannot effectively handle these problems, that is, they cannot solve these problems in polynomial time.

Definition 2.25 An encryption scheme is said to be breakable if a third party, without prior knowledge of the key pair (e, d), can systematically recover plaintext from corresponding ciphertext within some appropriate time frame.

Encryption schemes are not inherent to provide data integrity, and in some cases the encryption scheme may not be compromised while the cryptographic protocol may fail to provide confidentiality adequately.

Cryptographic techniques are typically divided into two generic types: symmetric-key system and public-key system.

Definition 2.26 A symmetric-key system is a system involving two transformations – one for the originator and one for the recipient – both of which make use of either the same secret key or two keys easily computed from each other. Symmetric-key system is also referred to as conventional cryptosystem.

Definition 2.27 An public-key system is a system involving two related transformations – one defined by a public key (the public transformation), and the other defined by a private key — with the property that it is computationally infeasible to determine the private transformation from the public transformation.

Most well-known public-key cryptosystems are based on the difficulty of intractable problems, such as the integer factorization problem, the discrete logarithm problem, and elliptic-curve discrete logarithm problem (ECDLP).

2.2.2 Symmetric-key encryption

Definition 2.28 Symmetric-key encryption is a cryptographic primitive which uses the same encryption and decryption keys, $e = d$, to perform encrypting and decrypting. Other terms used in the literature are single-key encryption, one-key encryption, private key encryption, secret key encryption, and conventional encryption.

In a two-party communication, the key in symmetric-key encryption must remain secret at both ends, and sound cryptographic practice dictates that the key be changed frequently perhaps for each communication session. In symmetric-key case, the only system that has proven secure is the one-time pad.

The symmetric-key primitives can be designed to have much higher rates of data throughput in encrypting and decrypting than public-key encryption.

There are two classes of symmetric-key encryption schemes which are commonly distinguished: block ciphers and stream ciphers.

Definition 2.29 A block cipher is an encryption scheme which breaks up the plaintext messages to be transmitted into strings (called blocks) of a fixed length t over an alphabet A, and encrypts one block at a time.

Most well-known symmetric-key encryption techniques are block ciphers. For example, DES, 3DES, IDEA etc.

Definition 2.30 A stream cipher is, in one sense, a very simple block cipher having block length equal to one.

What makes stream ciphers useful is the fact that the encryption transformation can change for each symbol of plaintext encrypted. They are especially useful in some situations, such as highly probable error transmission channel, or buffering of data is limited.

2.2.3 Public-key encryption

Definition 2.31 A public-key encryption is a cryptographic primitive which uses different encryption and decryption keys, the public-key e and the private key d and the two keys match each other; the encryption key e needn't be kept secret, and only the party who is the owner of e can decrypt a ciphertext encrypted under e using the matching private key d. Other terms used in the literature are asymmetric cryptosystems.

Each entity in the network has a public/private encryption key pair (e, d). The public-key e along with the identity of the entity is usually stored in a central repository. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario. Per-communication interaction with the central repository can be eliminated if entities store certificates locally. Depending on the mode of usage, a private/public-key pair (e, d) may remain unchanged for considerable periods of time, e.g., many communication sessions (even several years).

Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes. That is, the computational performance of a public-key encryption is inferior to that of a symmetric-key encryption.

2.2.4 Digital signatures

Definition 2.32 A digital signature is a cryptographic primitive which entails transforming the message and some secret information held by the entity into a tag called a signature in digital form, contrary to handwritten signature in previous centuries. Digital signature is fundamental in authentication, authorization, and nonrepudiation.

A digital signature of a message is a number dependent on some secret known only to the signer, and additionally, the content of the message being signed. Signatures must be verifiable; if a dispute arises as to whether a party signed a document (caused by either a lying signer trying to repudiate a signature it did create, or a fraudulent claimant), an unbiased third party should be able to resolve the matter equitably, without requiring access to the signer's secret information (private key).

A digital signature scheme (or digital signature mechanism) consists of a signature generation algorithm and an associated verification algorithm.

Schemes for unforgeable digital signature can be constructed using the same computational assumptions as used in the constructing of private-key encryption schemes, where the encryption key d should be kept secret, and all users including the adversary can decrypt a ciphertext encrypted under d using the matching public-key e.

Digital signatures have many applications in information security, including authentication, data integrity, and non-repudiation. One of the most significant applications of digital signatures is the certification of public-keys in large networks. Certification is a means for a trusted third party to bind the identity of a user to a public-key, so that at some later time, other entities can authenticate a public-key without assistance from a trusted third party.

Public-key techniques may be used to establish a key for a symmetric-key system being used by communicating entities A and B. In this scenario A and B can take advantage of the long-term nature of the public/private keys of the public-key scheme and the performance efficiency of the symmetric-key scheme. Note that data encryption is frequently the most time consuming part of the encryption process, and the public-key scheme for key establishment is a small fraction of the total encryption process between A and B[1].

The size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

2.2.5 Hash Functions

Definition 2.33 A hash function is a cryptographic primitive which is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed n bits length, called hash values. For any given $x \in X$, the corresponding hash value y is a binary string of some fixed n bits length, $y = h(x)$. Hash value is also referred to as a hashcode, hash-result, or simply hash.

The cryptographic hash function is often informally called a one-way hash function. The basic idea of cryptographic hash functions is that a hash value serves as a compact representative image (sometimes called an imprint, digital fingerprint, or message digest) of an input string, and can be used as if it were uniquely identifiable with that string.

The main properties of cryptographic hash functions include:

1) Preimage resistance — one-way, for essentially all pre-specified outputs, is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' so that $h(x') = y$ when giving any y for which a corresponding input is not known. A hash function with this property is called a one-way hash function.

2) 2nd-preimage resistance — weak collision resistance is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ so that $h(x') = h(x)$.

3) Collision resistance — strong collision resistance is computationally infeasible to find any two distinct inputs x, x' which hash to the same output so that $h(x) = h(x')$ (Note that here there is free choice of both inputs contrary to 2nd-preimage resistance). A hash function with this property is called a collision resistance hash function.

At the highest level, hash functions may be split into two generic classes:

1) An unkeyed hash function is a specific hash function whose specification dictates a single input parameter, a message.

2) A keyed hash functions is a specific hash function whose specification dictates two distinct inputs, a message and a secret key.

The most common cryptographic uses of hash functions are with digital signatures and are for data integrity. With digital signatures, a long message is usually hashed (using a publicly available hash function) and only the hash value is signed in place of the original message. For data integrity, the hash value corresponding to a particular input is computed at some point in time by the sender, and later the hash value is recomputed by the receiver using the particular input at hand which is the same as the sender, and then the recomputed hash value is compared for equality with the original hash value. The problem of preserving the integrity of a potentially large message is thus reduced to that of a small fixed-size hash value.

Specific applications, which hash functions are used for data integrity in conjunction with digital signature schemes, include virus protection and software distribution.

Keyed hash functions can be used to provide data origin authentication as conventional encryption schemes do. Keyed hash functions could not directly recover the input binary strings from a computed hash value while conventional encryption schemes could recover the plaintext from a ciphertext with a common secret key at hand. When they are used to provide integrity and data origin authentication information security services, they behave almost the same. Hence, in this book, we do not distinguish keyed hash functions from conventional encryption schemes accurately when they are used to provide data origin authentication, and we take keyed hash functions as a sort of symmetric-key primitives in this case since they both require a common secret key shared by some specific identities to complete computing.

2.2.6 Message authentication

Definition 2.34 Manipulation detection codes (MDCs) are a subclass of unkeyed hash functions to provide a representative image or hash for message manipulation detection. MDCs is also called modification detection codes

(MDCs), and less commonly known as message integrity codes (MICs).

MDCs may be further classified, the specific two primary classes of MDCs are:

1) one-way hash functions (OWHFs): for these hash functions, finding an input which hashes to a pre-specified hash value is difficult;

2) collision resistant hash functions (CRHFs): for these hash functions, finding any two inputs having the same hash value is difficult.

Definition 2.35 Message authentication codes (MACs) are a distinct class of hash functions, parameterized by a secret key k, which provide the detection of message alternation on an insecure channel for encryption schemes.

MAC algorithms are also a subclass of keyed hash functions which take two functionally distinct inputs, a message and a secret key k, and produce a fixed-size (say n-bit) output or hash value, with the design intent that it is infeasible in practice to produce the same output without knowledge of the key k. That is, a hash value should be uniquely identifiable with a single input in practice, and collisions should be computationally difficult to find (essentially never occurring in practice).

MACs can be used, without the use of any additional mechanisms, to provide data integrity and symmetric data origin authentication (the source of a message), as well as identification in symmetric-key schemes. That is, MAC allows message authentication by symmetric techniques.

Distinction should be made between the use of a MAC algorithm and that of a MDC, where MAC is with a secret key included as part of its message input.

It is generally assumed that the algorithmic specification of a hash function is public knowledge. Thus in the case of MDCs, given a message as input, anyone may compute the hash value, and in the case of MACs, given a message as input, anyone with knowledge of the key may compute the hash value.

A message authentication scheme does not necessarily constitute a digital signature scheme. In some sense, message authentication is similar to a digital signature. The difference between these two is that in the setting of message authentication it is not required that a "third" party (who may be the dishonest adversaries) should be able to verify the validity of authentication tags produced by the designed users[2].

Table 2.1 indicates various types of algorithms commonly used to achieve the specified cryptographic objectives. The classification given requires specification of both the type of algorithm (e.g., encryption vs. signature) and the intended use (e.g., confidentiality vs. entity authentication).

Data origin authentication is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some (typically unspecified) time in the past.

Table 2.1 Types of algorithms commonly used to meet specified objectives

Cryptographic objective (usage)	symmetric-key	public-key
confidentiality	encryption	encryption
data integrity	MDC	signature
message authentication	MAC	signature
data origin authentication	MAC	signature
non-repudiation	/	signature
key transport	encryption	encryption
key agreement	various methods	Diffie-Hellman
entity authentication (by challenge-response protocols)	1. MAC 2. encryption	1. signature 2. decryption 3. customized

Message authentication is a term used analogously with data origin authentication. It provides data origin authentication with respect to the original message source (and data integrity, but no uniqueness and timeliness guarantees).

In deed, data origin authentication implicitly provides data integrity since, if the message was modified during transmission, the message sender would no longer be the originator. Message authentication itself provides no timeliness guarantees, while entity authentication involves corroboration of a claimant's identity at the current instant in time through actual communications with an associated verifier during execution of the protocol itself.

Encryption may provide privacy of keying material in key establishment protocols, while signature and MAC may provide data integrity, message authentication and data origin authentication.

We use the terms identification and entity authentication synonymously throughout this book. Elsewhere in the literature of cryptography, identification sometimes implies only a claimed or stated identity whereas entity authentication suggests a corroborated identity. In many applications, the motivation for identification is to provide access service allowing resource usage to be tracked to identified entities, and to facilitate appropriate billing. In general, an authentication is considered flawed if a principal concludes a normal run of the protocol with its intended communication partner while the intended partner would have a different conclusion.

Identification schemes are closely related to, but simpler than digital signature schemes, which involve a variable message and typically provide a non-repudiation feature allowing disputes to be resolved by judging after the fact. Identifications do not have "lifetimes" as signatures do and a claimed identity is either corroborated or rejected immediately, with associated privileges or access either granted or denied in real time.

One of the primary purposes of entity authentication is to facilitate access control to a resource, when an access privilege is linked to a particular identity.

Entity authentication techniques may be divided into three main categories, depending on which of the following securities is based on[1]:

1) Something known. Examples include standard passwords, Personal Identification Numbers (PINs), and the secret or private keys.

2) Something possessed. It is typically a physical accessory, resembling a passport in function. Examples include magnetic-striped cards, smart cards (also called chipcards or IC cards), and hand-held customized calculators (also called password generators, or Tokencard, such as RSA Secure ID) which provide time-variant passwords or passcodes. The generator usually contains a device-specific secret key.

3) Something inherent (to a human individual). This category includes methods which make use of human physical characteristics and involuntary actions (biometrics), such as handwritten signatures, and fingerprints.

The important points in practice about cryptographic primitives are:

1) Public-key cryptography facilitates efficient signatures (particularly non-repudiation) and key management.

2) Symmetric-key cryptography is efficient for encryption and some data integrity applications.

2.3 Cryptographic protocols

This section gives some background knowledge about the cryptographic protocols including Message-driven protocol, secured channel, etc.

A communication protocol or protocol is a collection of interactive communication procedures that specify a particular processing of incoming messages and the generation of outgoing messages. A communication procedure including a sequence of determined steps runs between or among co-operative parties.

A Message-driven protocol is initially triggered at a party by an external "call" and later by the arrival of messages. Upon each of these events, and according to the protocol specification, the protocol processes information and may generate and transmit a message, and/or wait for the next message to arrive.

Note that message-driven protocols are asynchronous in nature, and this reflects the prevalent form of communication in today's networks.

Recall that a cryptographic protocol is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective.

A mechanism is a more general term encompassing protocols, including algorithms (specifying the steps followed by a single entity), and non-cryptographic techniques (e.g., hardware protection and procedural controls) to achieve specific security objectives.

2.3.1 Secure channel

Definition 2.36 A channel is a means of conveying information from one entity to another.

1) A physically secure channel (or secure channel) is the one that is not physically accessible to the adversary. Some channels are assumed to be physically secure, and these include trusted couriers, personal contact between communicating parties, and a dedicated communication link, for example.

2) An unsecured channel (or open channel, or unprotected channel) is the one from which parties other than those for which the information is intended can reorder, delete, insert, or read.

3) A secured channel (or protected channel,or authenticated channel) is the one from which an adversary does not have the ability to reorder, delete, insert, or read. A secured channel may be secured by physical or cryptographic techniques. Secure channel is a type of secured channel.

Definition 2.37 A session is a copy of a protocol run or a protocol instance at a party. Several copies of protocol run (i.e., interactive subroutines or other protocols) may be simultaneously instantiated by one party.

Technically, a session is an interactive subroutine executed inside a party. Each session is identified by the party that runs it, the parties with whom the session communicates and a session-identifier. The session-identifiers are used in practice to bind transmitted messages to their corresponding sessions.

A session could be completed at one party but not necessarily at the other. Each run of a protocol at a given party creates a local state for that session during execution. Typically, the local state of a session is mostly independent of local states of other sessions. When a session ends its run and outputs a key in a key establishment protocol, we call this session a complete one and assume that its local state is erased.

A session key is an ephemeral secret whose use is restricted to a short time period such as a single telecommunications connection (or a session), after which all trace of it is eliminated.

Key establishment protocols result in a shared ephemeral session key or shared ephemeral session key parts for subsequent communications. Motivation for using ephemeral keys includes the following:

1) to limit available ciphertext (under a fixed key) for cryptanalytic attack;

2) to limit exposure, with respect to both time period and quantity of data, in the event of (session) key compromise;

3) to avoid long-term storage of a large number of distinct secret keys (in the case where one terminal communicates with a large number of others) by creating keys only when actually required;

4) to create independence across communication sessions or applications. It is also desirable in practice to avoid the requirement of maintaining state information across sessions.

2.3.2 Principals

An entity (or a party) is someone or something that sends, receives, or manipulates information over point-to-point channels. An entity may be a person, a computer terminal, etc., or even an adversary.

Definition 2.38 A communicating entity or party is formally called principal in key establishment protocol with a unique name. They are probabilistic polynomial-time users, processes, or machines in current cryptographic literature, and we usually denote principals by $P_1, P_2, ..., P_n$, etc.

Definition 2.39 A trusted third party (TTP) is an entity in the network who is trusted by all other entities. This party is referred to by a variety of names depending on the role it plays: trusted third party, trusted server, authentication server, key distribution center (KDC), key translation center (KTC), and certification authority (CA).

Definition 2.40 A key translation center (KTC) is a trusted server which allows two parties A and B, which do not directly shared keying material, to establish a secure communication through use of the long-term keys K_{AT} and K_{BT} they respectively shared with TTP T.

In symmetric-key case, each entity shares a distinct symmetric key with the TTP. These keys are assumed to have been distributed over a secured channel. In public-key case, each entity knows the public-key of the TTP. The TTP carefully verifies the identity of each entity, and signs a message consisting of an identifier and the entity's authentic public-key.

From a communication viewpoint, TTPs may be classified based on their real-time interactions with other communicating entities such as A and/or B.

1) In-line TTP: it is an intermediary TTP, serving as the real-time means of communication between A and B.

2) On-line TTP: it is a TTP involved in real-time communication during each protocol instance (communicating with A and/or B), but A and B communicate directly rather than through T.

3) Off-line TTP: it is a TTP not involved in real-time communication during a protocol instance, but T provides or prepares information a priori, which is available to A and/or B and the priori is used during protocol execution.

In-line third parties are of particular interest when A and B belong to different security domains or cannot otherwise interact directly due to non-interoperable security mechanisms. Examples of in-line third parties include KDCs or KTCs which provide the communication paths between A and B, as in IEEE 802.11i[3].

On-line third parties are widely used in key establishment protocols. Examples include Needham-Schroeder shared key protocol[4], Woo-Lam

protocol[5] etc.

A typical example of an off-line third party is a certification authority (CA) producing public-key certificates and placing them in a public directory; here, the directory owner may be an on-line third party, but the certification authority is not. Protocols with off-line third parties usually involve fewer real-time message exchanges, and do not require real-time availability of the third parties.

Definition 2.41 A certificate is a TTP singed message, which binds the identity of an entity to its public-key. If the signing key (private key) of the TTP is compromised, all communications become insecure.

In this book, we suppose the honest entity TTP itself is unconditionally secure, while the TTP could be impersonated by an adversary with the requirement that the TTP machine itself has not been crashed.

In order for an entity B to verify the authenticity of the public-key of an entity A, B must have an authentic copy of the public signature verification function of the TTP. For simplicity, in this book, we assume that the authenticity of this verification function is provided to B by noncryptographic means, for example by B obtaining it from the TTP in person.

Definition 2.42 Certificate Authority (CA) is a trusted third party who is responsible for establishing and vouching for the authenticity of the public-key bound to the subject entity. CA's responsibility includes binding public-keys to distinguished names through signed certificates, managing certificate serial numbers, and certificate revocation.

The authenticity of the CA's public-key may be originally provided by non-cryptographic means including personal acquisition, or through trusted couriers; authenticity of the CA's public-key is required, but not secrecy. This CA public-key allows any system user, through certificate acquisition and verification, to transitively acquire trust in the authenticity of the public-key in any certificate signed by that CA.

Definition 2.43 A public-key certificate is a data structure consisting of a data part and a signature part. The data part contains cleartext data including, as a minimum, a public-key and a string identifying the subject party to be associated therewith. The signature part consists of the digital signature of a certification authority over the data part, thereby binding the subject entity's identity to the specified public-key.

2.3.3 Time-variant parameters

Definition 2.44 A time-variant parameter (TVP), also called freshness component, is a value used no more than once for the same purpose. It typ-

ically serves to prevent (undetectable) replay. It is sometimes called nonce, unique number, or non-repeating value.

Often, to ensure protocol security, the integrity of such TVPs must be guaranteed (e.g., by cryptographically binding them with other data via a MAC or digital signature algorithm).

Uniqueness means to be unique, and it is often required only within a given key lifetime or time window.

A uniqueness guarantee may be provided by a TVP such as a timestamp or a never-repeated sequential counter. The never-repeated sequential counter may not provide (real-time) timeliness, and thus are not appropriate to entity authentication.

Definition 2.45 Timeliness (or freshness) typically means to be recent, it is in the sense of having originated subsequent to the beginning of the current protocol instance.

A timeliness guarantee may also be provided using TVPs such as timestamps. Note that timeliness alone does not rule out interleaving attacks using parallel sessions.

Three main classes of time-variant parameters are below:

1) Random number. Random numbers may be used to provide uniqueness and timeliness assurances, and to preclude certain replay and interleaving attacks on protocols. In a challenge-response mechanism, the challenger must temporarily maintain per-connection short-term state information, but only until the response is verified. The term nonce is most often used to refer to a "random" number in a challenge-response mechanism, but the required randomness properties vary.

In protocol descriptions, "choose a random number" is intended to express "pick a number with uniform distribution from a specified sample space" or "select from a uniform distribution".

2) Sequence number (or serial number, or counter value). A sequence number serves as a unique number identifying a message, and is typically used to detect message replays. Forced delays of messages with sequence number are not detectable in general. It may also be used to provide timeliness in conjunction with the maintenance of pairwise (sender, receiver) state information. As a consequence of the overhead and synchronization maintenance necessary, sequence numbers are most appropriate for smaller, and closed groups.

3) Timestamp. Timestamps may be used to provide timeliness and uniqueness guarantees, to detect message replays and forced delays. It provides timeliness in conjunction with distributed timeclocks. Timestamp-based protocols require that timeclocks be both loosely synchronized (fixing clock drift) and secured from modification.

2.3.4 Challenge and response

A challenge is typically a nonce chosen by one entity at the outset of the protocol and subsequent challenges will differ from each other.

Definition 2.46 A challenge-response protocol (or Challenge-response mechanism) is that one entity (the claimant, or the sender) "proves" its identity to another entity (the verifier, the receiver) by demonstrating one's possession of a secret to be associated with the claimant, without revealing the secret itself to the verifier during the protocol.

This is done by providing a response to a time-variant challenge, where the response depends on both the claimant's secret and the time-variant challenge. Answering a challenge in challenge-response protocols requires some type of computing device and secure storage for long-term keying material.

Challenge-response mechanisms may be implemented via symmetric-key techniques, public-key techniques, and zero-knowledge techniques. Examples of challenge-response protocols based on symmetric-key encryption are the Kerberos protocol[6] and the Needham-Schroeder shared-key protocol[4]. Challenge-response mechanisms may also be implemented by keyed hash functions which could provide data origin authentication security service similar to a symmetric-key block cipher MAC, e.g., IEEE 802.11i[3].

2.3.5 Other classes of cryptographic protocols

Besides the classification of cryptographic protocols in Chapter 1, there exist other classifications. For example, cryptographic protocols are classified by time-variant parameters (TVPs) used:

1) Challenge-response protocols: One entity includes a (new) time-variant challenge, mostly a random number, in an outgoing message, then other entity provides a response to this challenge in the next protocol message, where the response depends on both the entity's secret and the challenge. This protocol instance is then deemed to be fresh based on the reasoning that the random number links the two messages. Typically the protocol involves one additional message compared to timestamp-based protocols, and the challenger must temporarily maintain state information, but only until the response is verified.

2) Timestamp-based protocols: timestamps are in conjunction with distributed timeclocks in a timestamp-based cryptographic protocol. Timestamp-based protocols require that timeclocks be both synchronized and secured. Timestamps in protocols may typically be replaced by a random number challenge plus a return message.

3) Sequence number-based protocols: sequence numbers are in conjunction with the maintenance of pairwise (claimant, verifier) state information in

a sequence number-based protocol. Sequence number should be guaranteed to be increasing and unique.

In addition, classification of cryptographic protocols can be also based on the use of trusted third party (cryptographic protocols with trusted third party; cryptographic protocols without trusted third party), or based on the cryptosystems used (symmetric-key based protocols and public-key based protocols).

2.4 Security of cryptographic protocols

This section gives some background knowledge about the security of cryptographic protocols including attack models, security models, analysis methods for protocol security, etc.

2.4.1 Attacks on primitives

Some generic types of attacks on encryption schemes, signature schemes, message authentication code schemes are given below.

1. Classification of attacks on encryption schemes

The following are some generic types of attacks on encryption schemes:

1) Ciphertext-only attack. The adversary (or cryptanalyst) tries to deduce the decryption key or plaintext by only observing ciphertext. Any encryption scheme vulnerable to this type of attack is considered to be completely insecure.

2) Known-plaintext attack. The adversary tries to deduce the decryption key or plaintext in possession of a quantity of plaintext and corresponding ciphertext.

3) Chosen-plaintext attack. The adversary chooses plaintext and is then given corresponding ciphertext. Subsequently, the adversary tries to recover plaintext corresponding to previously unseen ciphertext by any information deduced.

4) Adaptive chosen-plaintext attack. It is a chosen-plaintext attack wherein the choice of plaintext may depend on the ciphertext received from previous requests.

5) Chosen-ciphertext attack. The adversary chooses ciphertext and is then given corresponding plaintext. Subsequently, the adversary tries to recover plaintexts corresponding to different ciphertext by any information deduced.

6) Adaptive chosen-ciphertext attack. It is a chosen-ciphertext attack where the choice of ciphertext may depend on the plaintext received from previous requests.

2. Classification of attacks on signature schemes

The objective of the attacker on signature schemes is to compute the private key information of the signer, or to forge signatures, that is, to produce signatures which will be accepted as those of some other entity. The following are some generic types of attacks on signature schemes:

1) Key-only attack. The adversary tries to produce signatures which will be accepted as those of some other entity while the adversary knows only the signer's public-key.

2) Known-message attack. The adversary tries to forge signatures in possession of a quantity of signatures for a set of messages which are known to the adversary but not chosen by him.

3) Chosen-message attack. The adversary obtains valid signatures from a chosen list of messages before attempting to break the signature scheme. This attack is non-adaptive in the sense that messages are chosen before any signatures are seen.

4) Adaptive chosen-message attack. The adversary uses the signer as an oracle, and he may even request signatures of messages which depend on previously obtained signatures or messages.

3. Classification of attacks on message authentication codes

$h_k(x)$ is the message authentication code given the input value key k and some text x. To attack a MAC means: given one or more pairs $(x_i, h_k(x_i))$, without prior knowledge of a key k, the adversary can compute a new text-MAC pair $(x, h_k(x))$ for some text $x \neq x_i$. The following are some generic types of attacks on message authentication codes:

1) Known-text attack. The adversary obtains one or more text-MAC pairs $(x_i, h_k(x_i))$.

2) Chosen-text attack. The adversary chooses x_i and obtains more text-MAC pairs $(x_i, h_k(x_i))$.

3) Adaptive chosen-text attack. The adversary may choose x_i to obtain new text-MAC pair $(x_i, h_k(x_i))$ based on the results of prior queries.

Some practical applications may limit the number of interactions allowed for text-MAC pair queries over a fixed period of time, or may be designed so as to compute MACs only for inputs created within the application itself; but it is practical to allow access to an unlimited number of text-MAC pairs, or to allow MAC verification of an unlimited number of messages and to accept any with a correct MAC for further processing.

2.4.2 Attacks on protocols

Definition 2.47 A protocol failure (or mechanism failure) occurs when a mechanism fails to meet the goals for which it is intended, in a manner

whereby an adversary gains advantage not by breaking an underlying primitive such as an encryption algorithm directly, but by manipulating the protocol or mechanism itself.

That is, the underlying primitive has not been compromised but the protocol has failed to provide the intended security service adequately.

An adversary in a key establishment protocol may pursue many strategies, including attempting to:

1) Deduce a session key using information gained by eavesdropping.

2) Participate covertly in a protocol initiated by one party with another, and influence it, e.g., by altering messages so as to be able to deduce the key.

3) Initiate one or more protocol executions (possibly simultaneously), and combine (interleave) messages from one with another, so as to masquerade as some party or carry out one of the above attacks.

4) Without being able to deduce the session key itself, deceive a legitimate party regarding the adversary as the identity of intended party with which it shares a key.

In a unauthenticated key establishment protocol, impersonation is usually possible. In an entity authentication protocol, where there is no session key to attack, an adversary's objective is to deceive a legitimate party to believe that the protocol has been run successfully with a party other than the adversary.

The following are some generic types of attacks on cryptographic protocols:

1) Known-key attack. The adversary obtains some keys used previously and then uses this information to determine new keys.

2) Replay. The adversary records a communication session and replays the entire session, or a portion thereof, at some later point in time.

3) Impersonation. The adversary personates the identity of one of the legitimate parties in a network.

4) Dictionary. This is usually an attack against passwords. Typically, a password is stored in a computer file as the image of an unkeyed hash function. When a user logs on and enters a password, it is hashed and the image is compared with the stored value. An adversary can take a list of probable passwords, hash all entries in this list, and then compare this with the list of true encrypted passwords in file with the hope of finding matches.

5) Forward search. This attack is similar in spirit to the dictionary attack and is used to decrypt messages.

6) Interleaving attack. This type of attack usually involves parallel sessions and impersonation in an authentication protocol.

2.4.3 Security of protocols

It is typically assumed that protocol messages are transmitted over unsecured

channels and the adversary has complete control of the data therein, with the ability to record, alter, delete, insert, redirect, and reuse the past or current messages, and inject new messages. In general a protocol has the following properties:

1) Operational property. In the absence of active adversaries and communication errors, honest participants who comply with its specification always complete the protocol. For example, for a key establishment protocol, the honest participants always compute a common key and corroborate the identities of the parties with whom the key is shared.

2) Completeness property. In the absence of active adversaries and communication errors, a protocol is complete if, given an honest claimant and an honest verifier, the protocol succeeds with overwhelming probability (i.e., the verifier accepts the claimant's claim or they both have the knowledge of the new session key). The definition of overwhelming generally implies that the probability of failure is not of practical significance.

3) Soundness property. An interactive proof of a protocol is sound if there exists an expected polynomial-time algorithm M with the following property: if a dishonest prover (impersonating A) can, with non-negligible probability, successfully execute the protocol with B, then M can be used to extract A's secret from this prover's knowledge with overwhelming probability.

Suppose all protocols are operational and complete in this book, which is the basic correctness requirement of protocols.

1. Attack models

The following are some types of attack models:

1) Passive attack. The adversary is passive when facing a ciphertext, i.e., all that the adversary could do about a ciphertext is eavesdropping.

2) Indistinguishable chosen-plaintext attack (IND-CPA). In this attack model, the adversary is allowed to obtain an assistance in the encryption mode to break the target cryptosystems.

3) Indistinguishable chosen-ciphertext attack (IND-CCA). In this attack model, the adversary is allowed to obtain a conditional assistance in the decryption mode to break the target cryptosystems. Other synonymous names are lunchtime attack, midnight attack or indifferent chosen-ciphertext attack.

4) Indistinguishable adaptive chosen-ciphertext attack (IND-CCA2). In this attack model, the adversary is allowed to obtain an assistance in the decryption mode to break the target cryptosystems. Other synonymous names are small-hours attack.

2. Security models

Definition 2.48 (All-or-nothing security) For a given ciphertext output from a given encryption algorithm, the adversary either succeeds with obtaining the whole block of the targeted secret, or fails with nothing. Here "all" means to find the whole plaintext block which in general has a size stipulated by a security parameter of the cryptosystem; "nothing" means that

the adversary does not have any knowledge about the targeted secret before or after its attacking attempt.

All-or-nothing secrecy is unfit for the real world since the guarantee of the secrecy is valid only if the attacker is passive, i.e., all that the attacker could do about a ciphertext is eavesdropping. Hence, if a cryptosystem is all-or-nothing secure, then it is a "textbook crypto". Numerous attacks have been discovered in practice on the textbook cryptos.

Definition 2.49 (Semantic security, IND-CPA security) A cryptosystem with a security parameter k is said to be semantically secure: after the IND-CPA attack game being played with any polynomially bounded adversary, the advantage Adv for the adversary Malice to distinguish the two plaintexts chosen by the adversary is negligible. The semantic security is also called the security for indistinguishable chosen-plaintext attack, for short IND-CPA security.

Informally speaking, semantic security means that whatever is efficiently computable about the plaintext given the ciphertext, is also efficiently computable without the ciphertext. The notion of the IND-CPA security captures the intuition that any polynomially bounded adversary should not be able to obtain any apriori information about a plaintext.

Definition 2.50 (IND-CCA security) A cryptosystem with a security parameter k is said to be secure against an indistinguishable chosen-ciphertext attack (IND-CCA security): after the IND-CCA attack game being played with any polynomially bounded adversary, the advantage Adv for the adversary Malice to distinguish the two plaintexts chosen by the adversary is negligible.

Lunchtime attack is a quite restrictive attack model in that the decryption assistance provided to Malice is only available in a short period of time, which is not a reasonable or realistic scenario. In reality, naive users will remain permanently naive, and Malice will definitely strike back, probably even in the afternoon tea-break time[2]!

Definition 2.51 (IND-CCA2 security) A cryptosystem with a security parameter k is said to be secure against an indistinguishable adaptive chosen-ciphertext attack (IND-CCA2 security): after the CCA2 attack game being played with any polynomially bounded adversary, the advantage Adv for the adversary Malice to distinguish the two plaintexts chosen by the adversary is negligible.

Most cryptosystems which are IND-CPA secure may be particularly vulnerable in IND-CCA (or IND-CCA2) model. In CCA and CCA2 models, an adversary (now he is Malice) may get decryption assistance, that is, he may be in a certain level of control of a "decryption box" and so may have some ciphertext of his choice to be decrypted for him even though he does

not have possession of the decryption key. Such an assistance is treated as a "cryptanalysis training course" provided to Malice in order to ease his attack job. These modes of attacks, particularly CCA2 model, are realistic in many applications of public-key cryptography[7]. Nowadays, IND-CCA2 is becoming the standard and fit-for-application security notion for public-key cryptosystems[2]. New public-key encryption schemes need to have this security quality for general purpose applications in real world setting.

2.4.4 Analysis methods for protocol security

We identify two distinct approaches for analyzing cryptographic protocols: Informal approaches and formal approaches (or formalisms). Formal approaches are a natural extension to informal ones and they are more important in protocol security analysis field. The following are some types of analysis methods for protocol security[1, 2, 7, 8].

1. Ad hoc and practical analysis

It is also called heuristic security. Protocols are typically designed to counter standard attacks, and shown to follow accepted principles. This approach, perhaps, is the most commonly used and practical one, but it may provide least satisfying of protocol security. Claims of security in this class generally remain questionable, and unforeseen attacks remain a threat.

2. Complexity-theoretic analysis

It is also called computational security or computationally secure. An appropriate model of computation is defined, and adversaries are modeled as having polynomial computational power (they may mount attacks involving time and space polynomial in the size of appropriate security parameters). Security analysis of this mathematical type helps a protocol designer or analyzer to consider using correct or more precise cryptographic services, and so protocol flaws can be avoided.

Provable security, also called provably secure, may be considered as part of a special sub-class of the computational security. Provable Security is a formal method for proving the security of cryptographic schemes, in which the difficulty of breaking a particular scheme is formally related to that of solving a widely believed computational hard problem, such as integer factorization or the computation of discrete logarithms. Random oracle is a very powerful and imaginary hash function with the "mixing-transformation" property: for any input, the distribution of the output hash values is uniform in the function's output space. In provable security, random oracle is used to construct public-key encryption schemes out of using the basic and popular public-key cryptographic primitives.

Provable security is the most commonly used analysis method in com-

putational security. Hence, in this book, we typically use provable security to refer to computational security. It is often required for a scheme to be secure in this class, and most of the best known public-key and symmetric key schemes in current use are in it.

3. Information-theoretic analysis

This approach uses mathematical proofs involving entropy relationships to prove that protocols are unconditionally secure. An adversary is assumed to have unlimited computational resources, and the question is whether or not there is enough information available to defeat the system. Unconditional security for encryption systems is also called perfect secrecy. While unconditional security is ultimately desirable, this approach is not applicable to most practical schemes. This approach cannot be combined with computational complexity arguments because it allows unlimited computation, while computational complexity requires that the adversaries should only have polynomial computational power.

4. Symbolic manipulation analysis

It is also called formal methods, verification methods or formalisms. This approach uses a set of abstract symbols to express security properties and these abstract symbols can be manipulated. The so-called approaches include formal logic systems, term re-writing systems, expert systems, and various other methods which combine algebraic and state-transition techniques.

On one hand, symbolic formal analysis methods are simple but have proven to be of utility in finding flaws and redundancies in protocols, and some are automatable to varying degrees. On the other hand, the "proofs" provided are proofs within the specified formal system, and cannot be interpreted as absolute proofs of security. For example, the security of symbolic view regards an encryption as a deterministic function. The foundations for formal cryptology need to be strengthened.

2.5 Communication threat model

This section gives a brief introduction to the communication threat model.

2.5.1 Dolev-Yao threat model

Dolev and Yao propose a communication threat model[9], which has been widely accepted as the standard threat model for cryptographic protocols.

The Dolev-Yao threat model supposes that Malice, the attacker, controls the entire communication network, so Malice is able to observe all message traffic over the network, to intercept, read, modify or destroy messages. Fur-

ther more, Malice may perform transformation operations on the intercepted messages (such as encryption or decryption as long as he has in his possession of the correct keys), and send his messages to other principals by masquerading as some principal.

The Dolev-Yao threat model requires very few quantity assumptions on the behavior of the adversary. Here are the basic assumptions for the protocol environment in Dolev-Yao threat model.

1) In a perfect public-key system, as long as
— the one-way functions used are unbreakable;
— the public directory, including all the text $m \in M$ and its corresponding ciphertext $E_e(m)$ pairs, is secure and cannot be tampered with;
— everyone has access to the encryptions under public-key e;
— only the public-key owner has the corresponding private key d.

2) In a two-party protocol, only the two users who wish to communicate are involved in the transmission process; the assistance of a third party in decryption or encryption is not needed.

3) In a uniform protocol, the same format is used by every pair of users who wish to communicate.

4) The adversaries are "active" eavesdroppers: someone who first taps the communication line to obtain messages will try everything he can to discover the plaintext. More precisely, the followings are assumed:
— He can obtain any message passing through the network.
— He is a legitimate user of the network, and thus particularly can initiate a conversation with any other user.
— He will have the opportunity to become a receiver to any user.
— He can send messages to any principal by impersonating any other principal.

Recall that Malice only has polynomial computational power, so he has the following characteristics:
— Malice cannot guess a random number which is chosen from a sufficiently large space.
— Without the correct secret (or private) key, Malice cannot retrieve plaintext from given ciphertext, and cannot create valid ciphertext from given plaintext.
— Malice is not in control of many private areas of the computing environment, such as accessing the memory of a entity's offline computing device.

2.5.2 Assumptions of protocol environment

The Dolev-Yao-like threat model will be applied to the protocol analysis in this book, where Malice is assumed to have the entire control of the vulnerable network and his computational capability is polynomially bounded.

To clarify the threats that cryptographic protocols may be subject to, and to motivate the need for specific protocol characteristics, precise assumptions of protocol environment in our protocol analysis are given.

1. Assumptions of cryptosystems

1) Suppose cryptographic primitives are perfect. That is, when examining the security of protocols, it is assumed that the underlying cryptographic mechanisms used, such as encryption algorithms and digital signatures schemes, are secure in the protocol run. An adversary is hypothesized to be not a cryptanalyst attacking the underlying primitives such as encryption algorithms directly, or rather the one attempting to subvert the protocol objectives by manipulating the protocol or mechanism itself.

Suppose cryptographic primitives are unbreakable in protocol analysis, so an adversary gains advantages not by breaking an underlying primitive such as an encryption algorithm directly, but by manipulating the protocol or mechanism itself.

Recall that security for indistinguishable adaptive chosen-ciphertext attack (IND-CCA2) is fit-for-application in a practical cryptographic setting, and it can be mathematically specified and proved independent of qualifying assumptions. Hence, we suppose our "perfect" cryptographic primitives are practical primitives which are IND-CCA2 secure. Namely, under IND-CCA2, the failures in a cryptographic protocol are not in any way related to the strength or weakness of the primitive used, but related to the protocol logic flaws.

2) The secret (or private) key d of the particular key pair (e, d), in which the communication parties are being used, should be kept secret while communicating securely. Without the correct private key, Malice cannot be able to retrieve plaintext from given ciphertext, and cannot create valid ciphertext from given plaintext.

History has shown that maintaining the secrecy, the secret (or private) key, of the encryption and decryption transformations is very difficult indeed.

3) Suppose that a legitimate party is either totally corrupted or totally secure. That is, the secrecy or the local state for a session during execution is either totally known by the adversary or totally not.

In practice, it is a common case that the adversary finds part of the local state or other information, and then he can deduce the decryption key or recover the desired plaintext. However, we omit this information exposure case for it is out of the scope of this book: analyzing the security of the protocol itself.

4) Suppose that each party has his own private key and the public-keys of other parties (including the adversary) in public-key case; Suppose that each party shares long-term keys with co-operative principals or the trusted third party in symmetric-key case. Furthermore, private keys and shared keys are commonly assumed to be too long to guess in a computationally feasible way.

In common, public-keys or long-term keys are deployed safely before the

protocol run via authenticated channel or even traditional communication means. However, in some special cases, public-keys or long-term keys may also be transmitted in a cryptographic protocol.

5) Malice cannot guess a random number which is chosen from a sufficiently large space.

2. Assumptions of receiving messages

The receiving information of a participant is referred to as bit string. Further decision should be made to determine whether this bit string is a message or not. Suppose each entity can distinguish the sentence structure of a message from random bit strings and has the ability to recognize a basic message.

3. Assumptions of participants

Suppose each participant, given an honest claimant and an honest verifier, will comply with the determined protocol steps, and the protocol will succeed with overwhelming probability (i.e., for key establishment protocol, the verifier accepts the claimant's claim or they both have the knowledge of the new session key).

4. Assumptions of encryption and decryption

Suppose the encryption and decryption manipulation on a message are inverse functions, that is, they obey a set of term-rewriting rules. For example, $D_d(E_e(m)) = m$ for all $m \in M$, where public-key e and private key d are paired keys in a public-key system, and symmetric keys e and d are equal keys ($e = d$) in a symmetric-key system.

5. Assumptions of adversary

In general, it is assumed that Malice is very clever in manipulating communications over the open network. His manipulation techniques are unpredictable because they are unspecified.

Malice is able to observe all message traffic over the network, to intercept, read, modify or destroy messages at will, to perform transformation operations on the intercepted messages, and to send his messages to other principals by masquerading as some principal. However, Malice's computational capability is polynomially bounded, therefore there is a set of "words" that Malice does not know naturally at the beginning of a protocol run, this set of words can be secret messages or cryptographic keys for which a protocol is meant to protect.

Malice represents a coalition of bad guys and thereby may use a large number of computers across the open network in parallel. Hence, an adversary usually means a group of attackers in this literature.

Furthermore, suppose the attacker can perform a kind of cryptanalysis training course that helps him to obtain a conditional assistance, in the decryption mode or encryption mode, and makes him more experienced in the

future.

2.5.3 Expressions of cryptographic protocols

We adopt the formal notation expressions in the Dolev-Yao-like threat model, which distinguish the underlying primitives from the cryptographic protocols explicitly and the security of a cryptographic protocol is discussed under a "perfect" primitive.

One message procedure step usually is

$$\text{Message 2} \quad A \rightarrow B \quad \{m\}_{K_{AB}}.$$

"Message 2" means that this is the second message exchanged in a protocol. "$A \rightarrow B : \{m\}_{K_{AB}}$" indicates that the protocol participant A has sent the message "$\{m\}_{K_{AB}}$" to the opponent protocol participant B. "$\rightarrow$" shows that the message is sent from the arrow end terminal A to the arrow top terminal B. m is a text where $m \in M$, K_{AB} is a long-term key between the principals A and B. $\{m\}_{K_{AB}}$ is an encryption of m under key K_{AB}.

$I(A)$ (or $I(B)$) is the adversary Malice I impersonating A (or B respectively).

References

[1] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[2] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[3] IEEE Std 802.11i-2004. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Security Enhancements. July 2004

[4] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[5] Woo TYC, Lam SS (1992) Authentication for Distributed Systems. Computer 25(1): 39 – 52

[6] Miller SP, Neuman BC, Schiller JI, Saltzer JH (1987) Kerberos Authentication and Authorization System. Paper Presented at the Project Athena Technical Plan Section E.2.1. MIT, Boston

[7] Stallings W (2006) Cryptography and Network Security: Principles and Practice, 4th edn. Prentice Hall, New Jersey

[8] Goldreich O (2003) Foundations of Cryptography. Cambridge University Press, New York

[9] Dolev D, Yao AC (1983) On the Security of Public-key Protocols. IEEE Transactions on Information Theory 29(2): 198 – 208

3 Engineering Principles for Security Design of Protocols

Abstract Informal methods are useful in helping designers to find implicit assumptions and to avoid them. Prudent engineering principles presented by Abadi and Needham in the excellent paper have shown considerable use for the protocol designers. Cryptographic protocol engineering is a new notion introduced in this book to give a set of principles for cryptographic protocol design, which is derived from software engineering method. Cryptographic protocol engineering principles are composed of protocol engineering requirement analysis principles, detailed protocol design principles and protocol provable security principles. Furthermore, the protocol engineering principles are demonstrated with some well-known published protocols.

An enormous increase has been seen in the development and use of cryptographic protocols in distributed system in the past two decades. These cryptographic protocols intend to provide confidentiality, authenticity, integrity or nonrepudiation for applications. But, unfortunately, design of a cryptographic protocol, especially an authentication protocol, remains extremely error-prone, even for experts in this area. Some protocols have shown to be flawed even a long time after they were published. Needham-Schroeder public-key authentication protocol was found flawed by Lowe in 1995, seventeen years after its publication[1, 2].

The study of successful attacks on cryptographic protocols helps designers to learn from previous design errors, to understand general attack methods. However, claims of protocol security may remain questionable, although protocols are typically designed to counter standard attacks, and are shown to follow accepted principles, but they may be subject to unforeseen attacks. The reasons of protocol failure are implicit assumptions, such as implicit security strength, implicit attack models, implicit assumptions of features of a cryptographic algorithm, implicit principal identities, implicit freshness information in some messages, etc. These implicit assumptions may lead flaws to the protocols[2–6].

3.1 Introduction of engineering principles

Over time, various formalisms have been used to analyze cryptographic protocols, such as BAN logic, Random Oracle Model, Model Checking etc.[7–9]. These formalisms are useful to find some published or even some previously unrecognized flaws and redundancies. However, they are not beneficial directly to protocol designers in preventing flaws, and it is relatively complicated and difficult to prove security via formalisms for some cryptographic protocols in modern communication.

Informal methods are useful in helping designers to find implicit assumptions and to avoid them[3, 5, 6, 10, 11]. An excellent paper [3] by Abadi and Needham presents some informal principles for the design of cryptographic protocols and these principles have shown considerable use for the protocol designers. To make it more systematic and operational, in this book we put forward the notion of cryptographic protocol engineering where we regard the design of cryptographic protocol as an engineering and point out some important possible implicit assumptions, e.g., what the security requirements of a cryptographic protocol are, which attack model this protocol is acted upon, where this protocol is applied and how to avoid some particular features of a cryptographic algorithm, according to various design phases.

When designing a cryptographic protocol, it is important to identify assumptions in the protocol design and to determine the effect on the security objectives if that assumption is violated. Study of uncovered flaws in protocols motivates and allows an understanding of various design features of protocols, and that of successful attacks helps designers to avoid standard attacks. This explicitly or implicitly defines the threats a protocol is intended to address, and formulate design principles.

3.1.1 Prudent engineering principles

Abadi and Needham propose eleven heuristic prudent engineering principles for cryptographic protocols design, including two general principles and the famous and important principle about entity's identity[3].

The first basic principle of the two general principles is concerned with the content of a message:

> **P1**: Every message should say what it means: The interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content — though if there is a suitable formalism available, that is good too.

That is, all elements of the sentence meaning should be explicitly represented in the message, so that a recipient can recover the meaning without any context.

Another basic principle is concerned with the circumstances:

P2: 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.

The above two principles are basic general principles for the design of secure cryptographic protocols. Other Abadi and Needham's principles are:

P3: 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.

Principle P3 has proved to be a correct and well recognized principle for protocol design. Numerous known attacks could be avoided if this principle is followed[1, 6].

P4: Be clear about why encryption is being done. Encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security, and its improper use can lead to errors.

P5: When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to infer that the principal that signs a message and then encrypts it for privacy knows the content of the message.

P6: Be clear what properties you are assuming about nonces. What may do for ensuring temporal succession may not do for ensuring association and perhaps association is best established by other means.

P7: The use of a predictable quantity (such as the value of a counter) can serve in guaranteeing newness, through a challenge-response exchange. But 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.

P8: If timestamps are used as freshness guarantees by reference to absolute time, then the difference between local clocks at various machines must be much less than the allowable age of a message deemed to be valid. Furthermore, the time maintenance mechanism everywhere becomes part of the trusted computing base.

P9: A key may have been used recently, for example, to encrypt a nonce, yet be quite old, and possibly compromised. Recent use does not make the key look any better than it would otherwise.

P10: If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce that the message belongs to this protocol, and in fact to a particular run of the protocol, and to know its number in the protocol.

P11: The protocol designer should know which trust relations his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit though they will be founded on judgment and policy rather than on logic.

3.1.2 Cryptographic protocol engineering principles

The Abadi-Needham prudent engineering principles help the researchers a lot in designing a secure protocol. However, the design of cryptographic protocols is still an error-prone job for its subtlety. Some fixed versions of cryptographic

protocols illustrated by Abadi and Needham in [3] are found flawed, even if they have followed the design principles. We introduce the cryptographic protocol engineering notion into the analyzing of protocol security, set out some new principles for protocol design, and try to improve this situation.

Cryptographic protocol engineering is a new notion introduced in this book to give a set of principles for cryptographic protocol design, which is derived from software engineering method. Similar to software engineering, cryptographic protocol engineering is the application of engineering to protocol design, that is, the application of a systematic, disciplined, quantifiable approach to the development of cryptographic protocol design, and the study of these approaches.

Cryptographic protocol engineering means at least two different things: the first is that protocol design isn't merely the act of designing messages but a sequence of applications to protocol design. Secondly, the term "cryptographic protocol engineering" has been used to describe "building of cryptographic protocols which are so complex in practice although there may be only several messages in a protocol"[6].

Cryptographic protocol engineering principles are presented based on the software engineering idea, which help the designers to avoid implicit assumptions in protocol design.

Cryptographic protocol engineering principles are divided into three parts:

1) protocol engineering requirements analysis phase,
2) detailed protocol design phase, and
3) protocol provable security phase

corresponding to

1) software requirements analysis phase,
2) detailed software design phase, and
3) software test phase

in a software design. Note that the presented principles may be affinal with some previous protocol design principles and not completely original, but they are more operational[3, 12, 13].

First, we give principles to help protocol designers to answer what the security requirements of a cryptographic protocol are, which attack model this protocol actes upon, where this protocol is applied, when and which cryptographic service should be used. Then, we present principles on avoiding subtleties and errors in detailed protocol design, which is also the focus of [3]. At last, we address principles about provable security for protocol design. Furthermore, the protocol engineering principles are demonstrated with some well-known published protocols.

It is worth noting that we present these protocol engineering principles at an abstract level, and do not intend to discuss the concrete method on how to choose a cryptographic algorithm or an attack model for a particular protocol, on the correct implementation or appropriate application of a particular protocol.

Although these principles themselves are informal, they are useful directly

for protocol designers along the whole cryptographic protocol design. If designers adhere to these principles, a lot of confusion and a number of mistakes can be avoided and the risk of designing flawed protocols will be reduced.

3.2 Protocol engineering requirement analysis

Protocol engineering requirement analysis phase in the protocol design corresponds to software requirement analysis phase in the software development, and the protocol requirement analysis is about the security requirement analysis, plaintext analysis, application environment analysis, etc.

3.2.1 Security requirement analysis

Principle 3.1 Protocol designers should be clear about the security requirements of a cryptographic protocol, especially the security strength requirements of the protocol.

The security requirements of a cryptographic protocol include not only the security goals of the protocol but also the security strength to be reached.

The security goals of a cryptographic protocol are usually indicated explicitly, such as providing confidentiality, authenticity, integrity or nonrepudiation for applications.

1) For a public-key certificate, the objective may be to make one entity's public-key available to others over unsecured media without danger of undetectable manipulation.

2) For a key establishment protocol, the objective may be to provide both secrecy and authenticity of the key for protocol participants, which implies the identities of the parties sharing the key, and the temporality and the uniqueness of the key between these parties.

The security strength requirements also should be clearly stated for a cryptographic protocol. It is important to consider what assurances and properties an intended application requires while designing or selecting a cryptographic technique for use. The security strength requirements in various environments such as the mainframe computer systems, VPNs, LANs, and Internet are different.

The strength of a particular cryptographic algorithm is completely different. Some achieves all-or-nothing security, some semantic security, some chosen-ciphertext attack security, while other non-malleable security.

All-or-nothing security means that an adversary either succeeds in obtaining the whole block of the targeted secret, or fails with nothing. The all-or-nothing security for a cryptosystem is not sufficient. It is based on passive attack and could not defense indistinguishable chosen-plaintext attack

(IND-CPA). Most of the primitives in the textbook are all-or-nothing security ones.

Semantic security[13, 14], also called security for indistinguishable chosen-plaintext attack, means that a ciphertext doesn't leak any useful information about the plaintext to any adversary whose computational power is polynomially bounded. A protocol may defense chosen-plaintext attack, but it may not withstand the chosen-ciphertext attack (IND-CCA) or adaptive chosen-ciphertext attack (IND-CCA2).

Protocol designers often copy features from existing protocols to achieve confidentiality, authenticity or integrity, but they fail to take into account the particular features of a cryptographic algorithm used. Although protocol designers may have in their minds the cryptographic algorithm they expect to use, the strength of a cryptographic algorithm can be completely undermined by using it in an unsafe way. As a result of this, misuse of textbook cryptosystems frequently appears in the literature of cryptographic protocols even for serious real-world applications. Direct encryption of a password under a basic public-key encryption algorithm is a typical example of textbook crypto[13].

Example 3.1 With the apriori information in plaintext, a textbook encryption algorithm such as RSA does not hide partial information about a plaintext very well, hence it is less harder to recover some information in a cryptosystem than to recover the whole plaintext block. That is, if the plaintext message input to a basic or textbook public-key cryptographic primitives has a random distribution, then extracting a single bit of the plaintext message from a ciphertext is as hard as extracting the whole block of the plaintext.

Example 3.2 The RSA least significant bit can be as strong as the whole block of the plaintext. But if the owner of an RSA public-key acts as a decryption oracle to return a whole data block to a decryption request, he may leak the least significant bit.

Example 3.3 Optimal Asymmetric Encryption Padding (OAEP) and Cramer-Shoup public-key cryptosystem are formally provably secure under IND-CCA2, and they are practical public-key cryptosystems[15–18].

The protocol designers should be clear to the particular features including the security strength property of a cryptographic algorithm they intend to use before the algorithm is practically used.

3.2.2 Plaintext analysis

Principle 3.2 If a plaintext is characteristic, some measures should be taken to hide the apriori information of the plaintext.

Plaintexts are usually implicitly assumed to be random in the plaintext space. However, in many applications, plaintexts may contain apriori information that can be guessed easily.

Example 3.4 Plaintexts may be a password in a password dictionary, the name list of the candidates to be voted, or a value from a known range of salaries etc. To guess such information about a plaintext encrypted under an all-or-nothing encryption algorithm, an adversary can simply reencrypt the plaintext and see if the result is the same as the targeted ciphertext.

Example 3.5 ElGamal cryptosystem achieves the distribution of the encrypted plaintext message uniformly over the entire message space. But if a plaintext message is not in the subgroup generated by g (g is a random multiplication generator element of the group $\mathbb{Z}_p^*$ where p is a randomly chosen prime number), ElGamal cryptosystem will become a deterministic scheme that may leak partial information and an adversary can launch the meet-in-the-middle attack on it[13, 19].

Besides this, partial apriori information about the plaintext may provide the adversary an unfair advantage in some applications.

Example 3.6 Shamir, Rivest and Adleman propose a fair deal protocol for RSA mental poker game, which directly applies one-way trapdoor functions in the protocol. The variation of the RSA cryptosystem in this approach could not hide the quadratic residuosity (QR) information in plaintexts. By selecting a plaintext card in QR_N (quadratic residuosity modulo a composite number N) and the other not, the adversary can guess which card the opponent has got. That is to say, it could not defense indistinguishable chosen-plaintext attack (IND-CPA). This approach can be fixed by forcing all the cards chosen from QR_N.

Shannon's work establishes a principle of cryptographic practice that the plaintext should be as random as possible for most cryptographic applications. Apriori information of the plaintext can be hidden by data compression, by homophonic substitution, or by hash function. With the application of hash function, etc., the plaintext message could become randomized, then the problem of finding any information about the plaintext can be as hard as solving the difficulty hard problem.

3.2.3 Application environment analysis

Principle 3.3 Protocol designers should explicitly indicate the application environment where the protocol is fit for use.

The stringency of cryptographic requirements is different, depending on the susceptibility of the environment in question to various types of attacks.

Certain cryptographic protocol can be applied in a passive attack environment or in an adaptive attack environment, in local area networks or in Internet, in wired networks or in wireless networks.

For an environment with active adversaries, who are capable of intercepting, modifying, or injecting messages, neither party has assurances of the source identity of the incoming message or the identity of the party which may know the resulting key in a key establishment protocol.

If a protocol designer fails to indicate the environment where this protocol should be applied clearly, and caters for wide applications of their cryptographic protocol, then, as a result, the protocol may not offer the required security as it has guaranteed. Application environment analysis is tightly related to the security strength to be reached. The bandwidth and computation abilities should also be considered.

To avoid the determination of signature and encryption, probabilistic security is introduced, and many applicable cryptographic protocols also use randomized algorithm. However, the randomness property may be lost in some scenarios. In the case of the ElGamal encryption and signature, the randomization is due to the randomness of the ephemeral key in randomized algorithm.

Example 3.7 In ElGamal Signature scheme, if an ephemeral key l is reused to issue two signatures $(r_1, s_1), (r_2, s_2)$ for two messages $m_1 \neq m_2 \pmod{p-1}$, the private key k^{-1} may be computed from it: Since $r = g^l \pmod{p}$, and $s = l^{-1}(m - k^{-1}r) \pmod{p-1}$, we have $l(s_1 - s_2) \equiv (m_1 - m_2) \pmod{p-1}$, then we have $l^{-1} \equiv (s_1 - s_2)/(m_1 - m_2) \pmod{p-1}$, at last we can compute private key $k^{-1} \equiv (m_1 - ls_1)/r \pmod{p-1}$. Hence, this ephemeral key l should be chosen randomly from $\mathbb{Z}^*_{p-1}$, and shouldn't be reused, otherwise the private key k^{-1} could be deduced[13, 19].

In some applications, e.g., in wireless communication environment, especially when smart cards are used, random values are often far from being perfect and may be monitored by using probing or electromagnetic analysis[13, 20]. Then the assumption of perfect random generator is not available to devices equipped with limited randomness source. Hence, one must be prudent while using the probabilistic security cryptosystem, especially in a device equipped with limited randomness source.

3.2.4 Attack model and adversary abilities analysis

Principle 3.4 The attack model and the adversary abilities should be explicitly specified in security definition of a cryptographic protocol.

Different answers to assumptions of attack models give protocol designers completely different security[21–23]. For example, all-or-nothing security

is based on passive attack model; semantic security is based on chosen-plaintext attack model (IND-CPA); CCA2 security is based on adaptive chosen-ciphertext attack model. A protocol designed to be secure against one type of attack models may not be secure against another. What are the implicit assumptions of attack models behind the designer's protocol? and are they clear or obscure in designer's protocol?

Example 3.8 Rabin developed a public-key cryptosystem based on the difficulty of computing a square root modulo a composite integer. In all-or-nothing security, if there exists an algorithm for forging a Rabin signature, then the composite modulus can be factored via using this forging algorithm. While in an IND-CPA model, the adaptive adversary can ask the signer to issue the signature of a message $m = s^2 \pmod N$, the adversary has chosen an arbitrary $s \in \mathbb{Z}_N^*$. N is a composite integer, and $N = p \times q$ with p, q (being distinct odd primes). If the reply s' of the signer is any one of four square roots of m and $s' \neq \pm s \pmod N$, then N can be factored by the adaptive attacker. This basic Rabin scheme is absolutely unusable in any real world application, since adaptive attack is unavoidable.

Moreover, even if the attack model is specified, the abilities of an adversary are usually dark and designers do not really know what they can do to defense the attacks.

There are many assumptions of the abilities of the adversary. Particularly, an adversary may ask for concurrent arbitrarily-interleaved executions of the protocol, may modify messages or even prevent their delivery, may impersonate participants in the protocol and act as a "man-in-the-middle", may corrupt all protocol participants and may take other vicious actions discovered in the future. As a matter of fact, most attacks on published protocols are found as a result of changing the assumptions of the abilities of the adversary[4, 10, 13, 24].

Example 3.9 In Needham Schroeder shared key protocol[1], an implicit assumption is made that the old session key would not be vulnerable to compromise, but in deed, the old session key may compromise, hence there exists an attack on Needham Schroeder shared key protocol as shown in Example 3.39.

In real world communication settings, it is better to consider an adversary as a probabilistic polynomial-time attacker that has full control of the communication links. The adversary can do everything but decrypt a confidential content without the corresponding key to the message. It is better to avoid concrete description of an adversary's ability, since we cannot exhaust all possible types of attacks even we list all known attacks or attacks we can imagine.

3.2.5 Cryptographic service requirement analysis

Principle 3.5 Protocol designers should use correct or more precise cryptographic services in appropriate occasion to avoid protocol flaws due to misuse of them.

Cryptographic algorithms can provide different and determined cryptographic services, and they need to be combined to meet various information security objectives. A given security objective, whose algorithms are most effective is determined by the basic properties of the algorithms. Misuse of cryptographic services may lead to protocol flaws.

Example 3.10 Cipher Block Chaining (CBC) is a common block-cryptographic algorithm for encryption of general data. It chains all the cipher blocks together and makes each cipher block depend on all previous data blocks. It seems that CBC can provide data integrity protection, but it could not provide data integrity protection in any sense. The only security service that the CBC mode offers is to randomize output ciphertext. CBC padding is applied to SSL, IPSEC, WTLS, etc. to provide data integrity protection, which induces security flaws[25].

Example 3.11 RSA encryption is a widely used cryptographic algorithm. Since partially known plaintext is not uncommon in application, it is now widely agreed that RSA encryption should avoid using small encryption exponent. Otherwise, if there exists $m < N^{1/e}$, then message m can be found efficiently by extracting the e-th root in integers from ciphertext $c = m^e \pmod N$.

Example 3.12 This example is related to ElGamal signature in GPG (GNU privacy guard)[26]. Both a short private exponent and a short nonce are used for the generation of ElGamal signatures in GPG version 1.0.2 (January 2000) and 1.2.3 (August 2003). The signer's private key can be recovered in less than a second on a PC by using lattice-based attack[27].

Protocol designers often misunderstand the available service a cryptographic algorithm can provide. As a result, the cryptographic protocol will be insecure if the algorithm is not chosen correctly. It is prudent to analyze the cryptographic service requirement and security strength requirements of the protocol following Principle 3.1 and Principle 3.5, and then to select an appropriate cryptographic algorithm correspondingly.

3.3 Detailed protocol design

Detailed protocol design is the most important part of the whole protocol design work. Due to the asynchronous and complicated nature of contemporary networks, it is not easier to achieve cryptographic protocol security

than to achieve that of a cryptographic algorithm. In this section, we take some typical cryptographic protocols as representative examples to explain the principles about detailed protocol design.

3.3.1 Liveness of the principal's identity

Principle 3.6 If the identity of a principal is essential to the meaning of a message, protocol designers should not only mention the principal's name explicitly but also authenticate the principal's liveness.

An important goal of authentication is to establish a lively communication between or among the initiate identity and the identities sought by the first.

The liveness of a principal means that from a legitimate participant A's point of view, the B believes that the intended communication partner A is alive in this protocol run. That is to say, whenever a principal A is to be authenticated with the principal B, there should be some evidence that A has truly taken part in the run of this protocol.

At first, a principal only trusts a time-variant parameter (say TVP, including nonce, timestamps, new session keys, shared parts of new session keys etc.) unambiguously generated by the principal himself. The liveness of A, from B's point of view, can only be deduced by a signed message that contains the TVP trusted by B. The signed message can be a signature of A, a message encrypted under the long-term shared key between A and B with data integrity protection, etc.

Often, the claimed source identity or source network address of a message is not explicitly included as a message field, since in some cases the identity of the principal can be deduced from other information like keys applied or context. This may cause failure in protocols. See more attack examples in [28] about principal's identity on protocol A, protocol B, protocol C and protocol D in [29].

The principle P3 in paper [3] indicates that if the identity of a principal is essential to the meaning of a message, the principal's name should be mentioned explicitly in the message. This is a useful principle, but it may not guarantee the liveness of a principal via mention of a principal's name explicitly. The absence of principal's liveness can still cause failure in cryptographic protocol. Here are some examples.

Example 3.13 The Woo-Lam protocol is an authentication protocol based on symmetric-key cryptography[30]. In the Woo-Lam protocol as shown in Fig. 3.1, the principal A intends to authenticate B with the aid of a trusted third party S.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & A \\ \text{Message 2} & B \rightarrow A: & N_B \end{array}$$

Message 3 $A \rightarrow B: \quad \{N_B\}_{K_{AS}}$
Message 4 $B \rightarrow S: \quad \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$
Message 5 $S \rightarrow B: \quad \{N_B\}_{K_{BS}}$

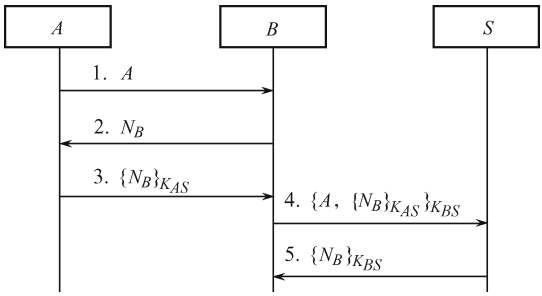


Fig. 3.1 The Woo-Lam authentication protocol.

Notation

A and B are two protocol principals, and S is trusted third party, a server; N_B is a nonce; K_{AS} and K_{BS} are keys that A and B shared with S respectively.

Premise

K_{AS} (K_{BS}) is the shared long-term key between A and S (B and S), which is initially established by non-cryptographic, and out-of-band techniques. N_B is randomly chosen by B.

Protocol actions

1) In Message 1, A claims his identity.
2) In Message 2, B randomly chooses a nonce N_B and sends it to A.
3) In Message 3, A returns this challenge encrypted under the shared long-term key K_{AS}.
4) In Message 4, B passes this encryption on to S for verification, bound with A's name encrypted under K_{BS}.
5) Upon receiving Message 4, S recovers N_B using K_{BS} and K_{AS}, and re-encrypts N_B using B's key K_{BS} and sends it to B in Message 5.
6) Upon receiving Message 5, B checks whether A has responded to the challenge N_B. If S replies $\{N_B\}_{K_{BS}}$, then B should be convinced that A is active in this protocol run.

1. Attack on the original Woo-Lam protocol

Abadi and Needham found that the original Woo-Lam protocol was flawed[3], as shown in Fig. 3.2, since nothing connects B's query to S with S's reply. Then the adversary I can impersonate A.

Message 1	$I(A) \rightarrow B:$	A
Message $1'$	$I \rightarrow B:$	I
Message 2	$B \rightarrow I(A):$	N_B
Message $2'$	$B \rightarrow I:$	N'_B
Message 3	$I(A) \rightarrow B:$	$\{N_B\}_{K_{IS}}$
Message $3'$	$I \rightarrow B:$	$\{N_B\}_{K_{IS}}$
Message 4	$B \rightarrow S:$	$\{A, \{N_B\}_{K_{IS}}\}_{K_{BS}}$
Message $4'$	$B \rightarrow S:$	$\{I, \{N_B\}_{K_{IS}}\}_{K_{BS}}$
Message 5	$S \rightarrow B:$	$\{N''_B\}_{K_{BS}}$
Message $5'$	$S \rightarrow B:$	$\{N_B\}_{K_{BS}}$

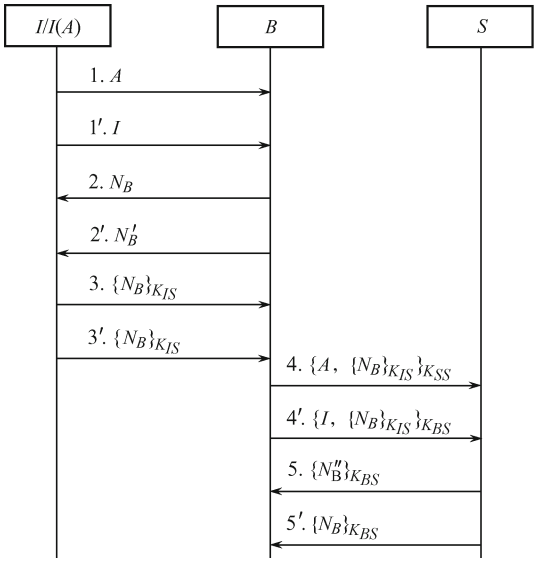


Fig. 3.2 An attack on the original Woo-Lam authentication protocol.

Notation

A and B are two legitimate protocol principals, S is a server, and I is an attacker with legitimate identity; N_B and N'_B are nonce; K_{AS}, K_{BS} and K_{IS} are long-term keys that A, B and I shared with S respectively; N''_B is the result of decrypting $\{N_B\}_{K_{IS}}$ using K_{AS}.

Premise

K_{AS} (K_{IS}, K_{BS}) is the shared long-term key between A and S (I and S, B and S), N_B is randomly chosen by B for the session between A and B, while N'_B is randomly chosen by B for the session between I and B.

Protocol actions

1) In Messages 1 and 1′, I tells B that both A (by impersonating A) and I want to establish a connection.

2) In Messages 2 and 2′, B provides challenges N_B and N'_B for sessions between A and B, and I and B respectively.

3) In Messages 3 and 3′, I replies to both challenges with the same $\{N_B\}_{K_{IS}}$.

4) Upon receiving Message 3, B could not find any abnormality since the received messages are encryption under K_{IS} which is not in B's possession. Hence B simply passes this encryption on to S for verification in Messages 4 and 4′.

5) Messages 5 and 5′ are the replies from S. One of these replies matches nothing, while the other one contains the challenge N_B intended for A. On the basis of the reply containing N_B, B must believe that A is active in this protocol run. On the basis of the reply containing N''_B, B believes that he has completed an unsuccessful protocol run with I.

2. Clark-Jacob attack on Woo-Lam protocol

Abadi and Needham gave a fixed version of the Woo-Lam Protocol in [3], by changing the last message of the protocol:

$$\text{Message 5} \quad S \rightarrow B: \quad \{A, N_B\}_{K_{BS}} \tag{3.1}$$

The fixed version indeed removes the attack in Fig. 3.2 for the attack will appear to be:

$$\text{Message 5} \quad S \rightarrow B: \quad \{I, N_B\}_{K_{BS}}$$

Since B is expecting (3.1), hence the attack could be detected. But this fixed version is still insecure. Upon receiving Message 3, the liveness of the principal A cannot be authenticated to B since B couldn't read the message. Upon receiving Message 5, B still couldn't authenticate the liveness of the principal A since there doesn't exist any evidence that A is alive. Messages 4 and 5 are similar since B has signed an encrypted message $\{N_B\}_{K_{AS}}$ that B could not read, which results in an attack discovered by Clark and Jacob in [6], as illustrated in Fig. 3.3.

$$\begin{aligned}
&\text{Message 1} \quad I(A) \rightarrow B: \quad A\\
&\text{Message 2} \quad B \rightarrow I(A): \quad N_B\\
&\text{Message 3} \quad I(A) \rightarrow B: \quad N_B\\
&\text{Message 4} \quad B \rightarrow I(S): \quad \{A, N_B\}_{K_{BS}}\\
&\text{Message 5} \quad I(S) \rightarrow B: \quad \{A, N_B\}_{K_{BS}}
\end{aligned}$$

3. Improved attack on the new fixed version against Clark-Jacob attack

To defense the Clark-Jacob attack, the last message of the protocol could be changed to:

$$\text{Message 5} \quad S \rightarrow B: \quad \{A, B, N_B\}_{K_{BS}} \tag{3.2}$$

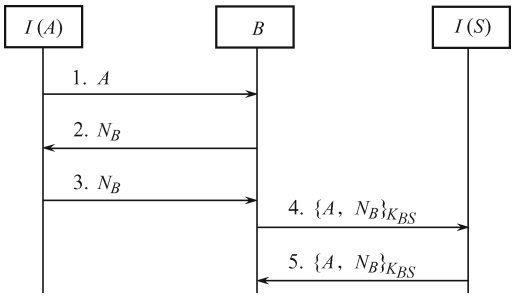


Fig. 3.3 The Clark-Jacob attack on the fixed Woo-Lam protocol.

The new fixed version removes the attack in Fig. 3.3 for the attack will appear to be:

$$\text{Message 5} \quad S \rightarrow B : \quad \{A, N_B\}_{K_{BS}}$$

Since B is expecting (3.2), the attack could be detected. Unfortunately, this new fixed version is also flawed, Fig. 3.4 indicates an improved attack.

Message 1	$A \rightarrow I :$	A
Message 1′	$I(A) \rightarrow B :$	A
Message 2′	$B \rightarrow I(A) :$	N_B
Message 2	$I \rightarrow A :$	N_B
Message 3	$A \rightarrow I :$	$\{N_B\}_{K_{AS}}$
Message 3′	$I(A) \rightarrow B :$	$\{N_B\}_{K_{AS}}$
Message 4′	$B \rightarrow S :$	$\left\{A, \{N_B\}_{K_{AS}}\right\}_{K_{BS}}$
Message 5′	$S \rightarrow B :$	$\{A, B, N_B\}_{K_{BS}}$
Message 4	$I \rightarrow S :$	$\left\{A, \{N_B\}_{K_{AS}}\right\}_{K_{IS}}$
Message 5	$S \rightarrow I :$	$\{A, I, N_B\}_{K_{IS}}$

Notation

A and B are two legitimate protocol principals, S is a server, and I is an attacker with legitimate identity; N_B is a nonce; K_{AS}, K_{BS} and K_{IS} are keys that A, B and I shared with S respectively.

Premise

K_{AS} (K_{BS}, K_{IS}) is the shared long-term key between A and S (B and S, I and S), N_B is randomly chosen by B for the session between A and B.

Protocol actions

1) In Message 1, A tells I that A wants to establish a connection with I; upon receiving Message 1, I establishes a connection with B instantly by

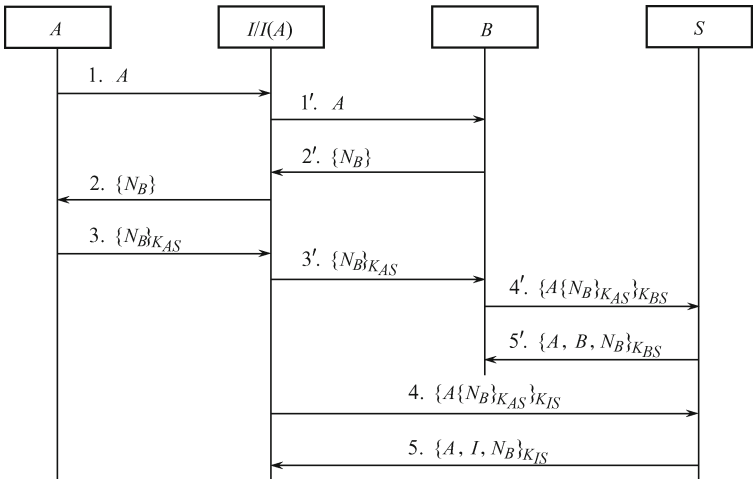


Fig. 3.4 An improved attack on the new fixed Woo-Lam protocol.

impersonating A.

2) In Message $2'$, B provides challenge N_B for the session between A (indeed I) and B; upon receiving Message $2'$, I provides the same challenge N_B to A for the session between A and I.

3) In Message 3, A returns this challenge N_B encrypted under K_{AS} to I; upon receiving Message 3, I passes $\{N_B\}_{K_{AS}}$ to B as A's response to the challenge N_B by impersonating A. As we have seen, in Message 3, A serves as an encryption oracle in the session between $I(A)$ and B.

4) Upon receiving Message $3'$, since the received message is an encryption under K_{AS}, B simply passes this encryption $\{N_B\}_{K_{AS}}$ on to S for verification in Message $4'$.

5) Message $5'$ is the reply from S which contains the challenge N_B intended for A and B. On the basis of the reply containing $\{A, B, N_B\}$, B believes that A is active in this protocol run.

6) For another protocol run between A and I, upon receiving Message 3, I continues the session with A by sending and receiving Message 4 and Message 5 normally, and at last, successfully complete the protocol run.

This is a perfect attack, all principals including A, B and S could not find any abnormality. Upon termination of the run of this fixed Woo-Lam protocol, B accepts "the run with A", but in fact A has not launched the run with B at all, and A thinks that he has completed a normal protocol run with I. As we have seen in this example, even if the principal names A and B are explicitly mentioned in this protocol revision, the absence of A's liveness still causes this flaw.

As we have seen, it is definitely an important and prudent principle to have the identities of the protocol participants explicitly specified in a protocol for developing secure authentication protocols. However, it is more important to authenticate the principal's liveness.

From the absence of security properties, it is convenient to give the revision of the above protocol. For example, in order to fix the absence of A's liveness in this protocol, there should be a message unambiguously generated by A, and generated not only for this protocol but for this run, to provide some evidence for B to authenticate A. We can achieve this by changing Message 3 to $\{B, N_B\}_{K_{AS}}$. In this fixed version, upon receiving Message 5, B can believe that it must be S who has generated the message $\{A, B, N_B\}_{K_{BS}}$. Since N_B is fresh for B, B believes that S is alive. From Message 4, the genuine S must have decrypted a message $\{B, N_B\}_{K_{AS}}$ to get N_B, so it must be A who has generated the message $\{B, N_B\}_{K_{AS}}$, and A knows that N_B is associated with A and B in this run, hence B believes the liveness of A.

4. Abadi-Needham simplified version of Woo-Lam Protocol

In [3], Abadi and Needham give a simplified version of Woo-Lam Protocol as shown in Fig. 3.5.

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

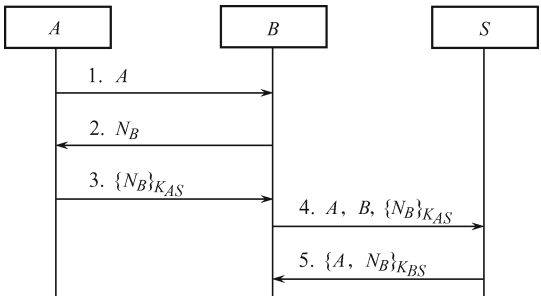


Fig. 3.5 The Abadi-Needham simplified version of Woo-Lam protocol.

Although Message 4 and Message 5 are dissimilar, the absence of A's liveness still exists in this version, hence there exists an attack on this protocol simplification, similar to that in Fig. 3.4.

Example 3.14 Fig. 3.6 illustrates a pair-key management approach which is a key establishment protocol in sensor networks based on symmetric-key cryptography [31]. It allows two sensor nodes A and B to establish a shared session key k_{AB} with the help of the trusted third party KDC.

1) $A \rightarrow KDC_j: \quad KDC_j, ID_A, ID_B, N_A$

2) $KDC_j \rightarrow A: \quad ID_A, KDC_j, ticket_B, N_A, \{k_{AB}, ID_B, N_A, L\}_{K_{A,KDC_j}}$

3) $A \rightarrow B: \quad ID_A, ID_B, ticket_B, \{ID_B, T_A\}_{k_{AB}}$

4) $B \rightarrow A: \quad ID_A, ID_B, \{T_A\}_{k_{AB}}$

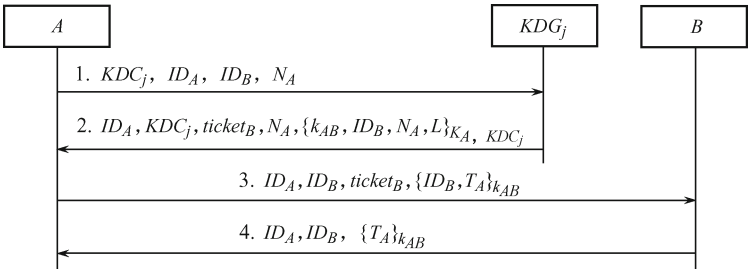


Fig. 3.6 A pair-key management approach in sensor networks.

Notation

A and B are sensor nodes who intend to run the pair-key management approach. KDC_j is the jth key distribution center (KDC). ID_A, ID_B are the certificates of nodes A and B respectively; N_A is a nonce, T_A is a timestamp, and k_{AB} is a new session key established between A and B with the help of KDC; $ticket_B$ is a ticket generated by KDC_j for B's session, and equals $\{k_{AB}, N_A, L\}_{K_{B,KDC_j}}$, where L is the life of the key k_{AB}; K_{A,KDC_j} and K_{B,KDC_j} are keys that sensor nodes A and B shared with KDC (KDC_j) respectively.

Premise

ID_A, ID_B are public information. K_{A,KDC_j} (K_{B,KDC_j}) is the shared long-term key between A and KDC_j (B and KDC_j), which is initially established by non-cryptographic, and out-of-band techniques; N_A is randomly chosen by A; T_A is a timestamp generated by A. k_{AB} is a new session key chosen by KDC_j.

Protocol actions

1) In Message 1, A claims the identity of himself, the identity of opponent sensor node B to the trusted third party KDC_j, and provides a nonce N_A.

2) In Message 2, KDC_j randomly chooses a new session key k_{AB} for A and B, encrypts k_{AB}, with the certificate of node B, the TVP N_A (to avoid replay attack), and the life of the key k_{AB} using K_{A,KDC_j}. Meanwhile, KDC_j generates $ticket_B$ which is an encryption of k_{AB} together with N_A and the life of the key k_{AB}.

3) Upon receiving Message 2, A recovers k_{AB} from $\{k_{AB}, ID_B, N_A, L\}_{K_{A,KDC_j}}$ using K_{A,KDC_j}.

4) In Message 3, A claims the identity of himself to the opponent sensor node B, forwards $ticket_B$, and shows his possession of k_{AB} via the encryption of plaintext $\{ID_B, T_A\}$ using k_{AB}.

5) Upon receiving Message 3, B recovers k_{AB} from $ticket_B$, and then checks whether A has possession of k_{AB} via decryption of $\{ID_B, T_A\}_{k_{AB}}$.

6) In Message 4, B encrypts T_A using k_{AB} to show his possession of k_{AB} to A.

7) Upon receiving Message 4, A checks whether B has possession of k_{AB} via decryption of $\{T_A\}_{k_{AB}}$.

Upon termination of this protocol run, it appears that both A and B have confirmed possession of k_{AB} by itself and the opponent participant.

Attack on the the pair-key management approach in sensor networks

This approach is also insecure. At the second step of the protocol, since N_A is fresh for A, A believes that KDC_j is alive and k_{AB} is associated with B and A. At the third step of the protocol, since there is no TVP trusted by B, and $ticket_B$ may be a replay message including a compromised key k_{AB}, B cannot believe that KDC_j or A is alive. At the fourth step of the protocol, A believes that only B can read the ticket $ticket_B$ to obtain k_{AB}, so it must be B who has encrypted the fresh timestamp T_A, hence A believes that B is alive. Upon termination of this run, the liveness of A has not been authenticated. From the absence of the liveness of A, we can construct an attack as shown in Fig. 3.7.

1) $I \rightarrow KDC_j: \quad KDC_j, ID_I, ID_B, N_I$

2) $KDC_j \rightarrow I: \quad ID_I, KDC_j, ticket_B, N_I, \{k_{AB}, ID_B, N_I, L\}_{K_{I,KDC_j}}$

3) $I(A) \rightarrow B: \quad ID_A, ID_B, ticket_B, \{ID_B, T_I\}_{k_{AB}}$

4) $B \rightarrow I(A): \quad ID_A, ID_B, \{T_I\}_{k_{AB}}$

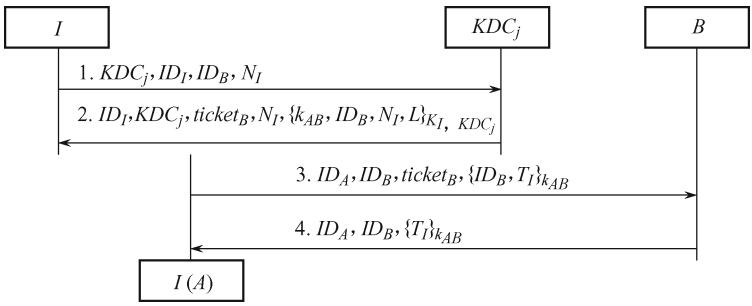


Fig. 3.7 An attack on the pair-key management approach in sensor networks.

Notation

T_I is a timestamp, ID_I is the certificate of the adversary I; K_{I,KDC_j} is the key that the adversary I shares with KDC (KDC_j). Other notations are the same as in Fig. 3.6.

premise

Besides the premises of the protocol in Fig. 3.6, the adversary I is also a legitimate sensor node, and ID_I is public information; K_{I,KDC_j} is the shared

long-term key between I and KDC_j.

Protocol actions

1) In Message 1, I claims the identity of himself, the identity of opponent sensor node B to the trusted third party KDC_j, and provides a nonce N_I.

2) In Message 2, KDC_j randomly chooses a new session key k_{AB} for I and B. Then KDC_j encrypts k_{AB} with the certificate of node B, the TVP N_A (to avoid replay attack), and the life of the key k_{AB} using K_{A,KDC_j}. Meanwhile, KDC_j generates $ticket_B$ which is an encryption of k_{AB} together with N_I and the life of the key k_{AB}.

3) Upon receiving Message 2, I recovers k_{AB} by using K_{I,KDC_j}, and records the $ticket_B$ $\{k_{AB}, N_I, L\}_{K_{B,KDC_j}}$.

4) In Message 3, the legitimate and malicious I impersonates sensor node A, and sends a message $ID_A, ID_B, ticket_B, \{ID_B, T_I\}_{k_{AB}}$ to the opponent sensor node B. Note that ID_A, ID_B are public information, I has recorded $ticket_B$, and I can encrypt $\{ID_B, T_I\}$ with possession of k_{AB}.

5) Upon receiving Message 3, B recovers k_{AB} from $ticket_B$, and then checks whether A (indeed, it is the adversary I) has possession of k_{AB} via decryption of $\{ID_B, T_I\}_{k_{AB}}$.

6) In Message 4, B encrypts T_I using k_{AB} to show his possession of k_{AB} to A (indeed, the adversary I).

7) Upon receiving Message 4, A (indeed, the adversary I) checks whether B has possession of k_{AB} via decryption of $\{T_I\}_{k_{AB}}$.

This is a perfect attack. Upon termination of the attack, B accepts the run with A even if A has not participated in the run with B at all. And B believes that B shares a new session key k_{AB} with A, but in deed B shares the key k_{AB} with the adversary I.

Example 3.15 In [32], Otway and Rees describe a key establishment protocol as shown in Fig. 3.8. It allows two parties A and B to establish a shared key k_{AB}, with the help of a server S.

Message 1 $A \rightarrow B: \ M, A, B, \{N_A, M, A, B\}_{K_{AS}}$

Message 2 $B \rightarrow S: \ M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$

Message 3 $S \rightarrow B: \ M, \{N_A, k_{AB}\}_{K_{AS}}, \{N_B, k_{AB}\}_{K_{BS}}$

Message 4 $B \rightarrow A: \ M, \{N_A, k_{AB}\}_{K_{AS}}$

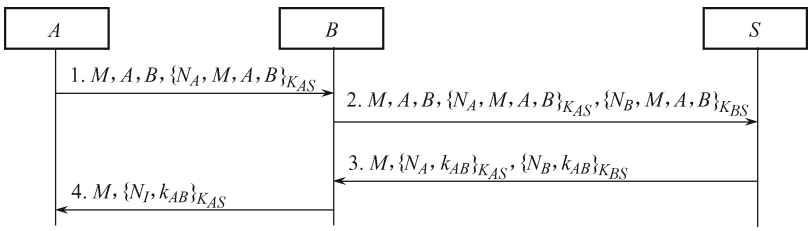


Fig. 3.8 The Otway-Rees key establishment protocol.

Notation

A and B are principals, S is an online trusted third party; M, N_A, N_B are nonces, and M is unique for each particular protocol run; k_{AB} is a new session key established between A and B with the help of S; K_{AS} and K_{BS} are keys.

Premise

K_{AS} (K_{BS}) is the shared long-term key between A and S (B and S), which is initially established by non-cryptographic, and out-of-band techniques; N_A and N_B are randomly chosen by A and B respectively; k_{AB} is chosen by S as a new session key for A and B.

Protocol actions

1) In Message 1, A randomly chooses M, N_A for this protocol run, and A tells B that A wants to establish a connection with B.

2) In Message 2, B randomly chooses N_B for this run, and tells S that A wants to establish a connection with B.

3) Upon receiving Message 2, S recovers N_A, N_B from the encryptions $\{N_A, M, A, B\}_{K_{AS}}$ and $\{N_B, M, A, B\}_{K_{BS}}$ using K_{AS} and K_{BS} respectively. S checks M to test whether the message $\{N_B, M, A, B\}_{K_{BS}}$ is generated by B for this particular protocol run. If M is the same as that in $\{N_A, M, A, B\}_{K_{AS}}$, then S believes B's presence for this run and M is unique for this protocol run between A and B.

4) In Message 3, S randomly chooses a new session key k_{AB} for A and B, and sends $\{N_A, k_{AB}\}_{K_{AS}}$ to A and $\{N_B, k_{AB}\}_{K_{BS}}$ to B.

5) Upon receiving Message 3, B checks the freshness of the message $\{N_B, k_{AB}\}_{K_{BS}}$ via N_B and then recovers k_{AB} from it. When B receives $\{N_B, k_{AB}\}_{K_{BS}}$, B knows that the uniqueness of M must have been checked by S, otherwise S would not send $\{N_B, k_{AB}\}_{K_{BS}}$ to B.

6) Upon receiving Message 4, A checks the freshness of Message 4 via N_A and recovers k_{AB} from it. When A receives $\{N_A, k_{AB}\}_{K_{AS}}$, A knows that the unique nonce M must have been checked by S to guarantee B's presence before S sends $\{N_A, k_{AB}\}_{K_{AS}}$ to A. Hence, upon receiving $\{N_A, k_{AB}\}_{K_{AS}}$, B's presence is guaranteed to A.

Upon termination of this protocol run, B believes that S is alive from receiving fresh nonce N_B, and A believes that S is alive from receiving fresh nonce N_A, and B's presence is also guaranteed to A.

1. Attack on the Otway-Rees key establishment protocol

In the Otway-Rees key establishment protocol, A's presence is not guaranteed to B. Suppose the adversary I has recorded the messages $M, A, B, \{N_A, M, A, B\}_{K_{AS}}$ in an old protocol run, then I can launch a new protocol run by impersonating A, Fig. 3.9 illustrates the following attack:

Message 1 $I(A) \rightarrow B: \; M, A, B, \{N_A, M, A, B\}_{K_{AS}}$

Message 2 $B \rightarrow S: \; M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$

Message 3 $S \rightarrow B: \; M, \{N_A, k'_{AB}\}_{K_{AS}}, \{N_B, k'_{AB}\}_{K_{BS}}$

Message 4 $B \rightarrow I(A): \; M, \{N_A, k'_{AB}\}_{K_{AS}}$

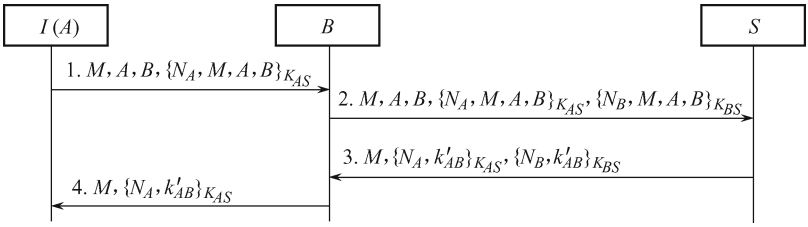


Fig. 3.9 An attack on the Otway-Rees original protocol.

Upon termination of this protocol run, the adversary I makes B believe that B has been talking and sharing a session key k'_{AB} with A, while A has not participated in the run. B will never be notified of any abnormality and B may omit A's subsequent requests for establishing a session key. The implicit problem is that A's presence has never been authenticated to B in this protocol.

2. A simplified version of the Otway-Rees key establishment protocol

In [3], Abadi and Needham suggest that the encryption of N_B in Message 2 is unnecessary, as shown in Fig. 3.10, and many encryptions could be avoided when names are included in S's reply. M is also omitted as a redundancy in this simplified version of the Otway-Ree protocol. The simplification may make the protocol even insecure.

Message 1 $A \rightarrow B: \; A, B, N_A$

Message 2 $B \rightarrow S: \; A, B, N_A, N_B$

Message 3 $S \rightarrow B: \; \{N_A, A, B, k_{AB}\}_{K_{AS}}, \{N_B, A, B, k_{AB}\}_{K_{BS}}$

Message 4 $B \rightarrow A: \; \{N_A, A, B, k_{AB}\}_{K_{AS}}$

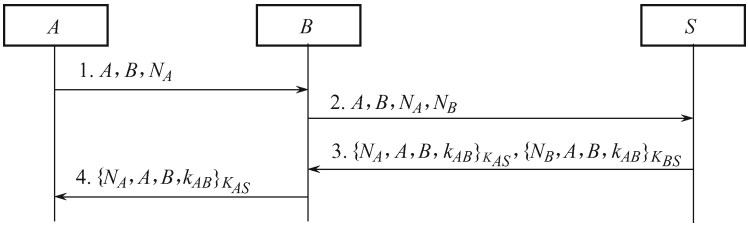


Fig. 3.10 The Abadi-Needham simplified version of the Otway-Ree key establishment protocol.

3. Attack on the simplified version of the Otway-Rees key establishment protocol

Upon termination of the protocol run of the simplified version, neither A's presence nor B's presence is guaranteed to B and A. From the absence of A's presence, there is an attack (similar to the attack presented in Fig. 3.9) on the Abadi-Needham simplified version of the Otway-Ree key establishment protocol.

From the absence of B's presence, we can construct another attack:

Message 1 $A \rightarrow I(B): \ A, B, N_A$

Message 2 $I(B) \rightarrow S: \ A, B, N_A, N'_B$

Message 3 $S \rightarrow I(B): \ \{N_A, A, B, k'_{AB}\}_{K_{AS}}, \{N'_B, A, B, k'_{AB}\}_{K_{BS}}$

Message 4 $I(B) \rightarrow A: \ \{N_A, A, B, k'_{AB}\}_{K_{AS}}$

Upon termination of this protocol run, the adversary I makes A believe that A has been talking and sharing a session key k'_{AB} with B, while B has not participated in the run. Note that the adversary I may also launch denial of service on the online trusted third party S by impersonating B.

As we have seen, the unique nonce M and the encryption of M in Message 2 are necessary for B's presence to S, then to A.

We may remove the flaw in the original Otway-Ree Protocol by adding Message 5 to the protocol, as shown in Fig. 3.11.

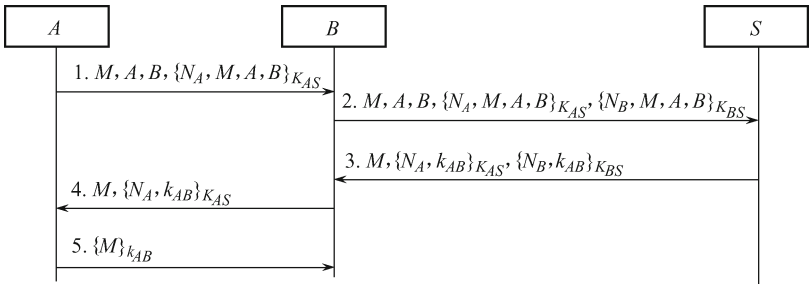


Fig. 3.11 A revision of the original Otway-Ree key establishment protocol.

Message 5 $A \rightarrow B: \ \{M\}_{k_{AB}}$

Upon receiving Message 5, B recovers M from $\{M\}_{k_{AB}}$ to check A's possession of k_{AB}. Since B believes that M is unique for this protocol run between A and B, B can guarantee the liveness of A.

3.3.2 Freshness and association of time-variant parameter

Principle 3.7 If a time-variant parameter is essential to the meaning of a message, it is prudent to authenticate the freshness and association of the

TVP. It is important for protocol designers to understand properly where the TVP is needed and what it is associated with – the freshness of a message or the liveness of a principal.

It is the most important part of authentication and key establishment to deem whether a message is fresh or a principal is present. For each communication principal, the belief about whether a message is fresh or a principal is present is different.

A message which is deemed to have been issued recently is often referred to as a fresh message. In common sense, the freshness of a message implies a good correspondence between the communication principals. The liveness of a principal implies online correspondence.

Time-variant parameters (TVPs) are the most commonly used techniques for timeliness. Time-variant parameters can be nonces, timestamps, sequence numbers, new session keys or share-parts of a new session key.

Verifiable timeliness of a message or a principal may be provided through use of nonces (also new session keys or share-parts of a new session key), timestamps in conjunction with distributed timeclocks, or sequence numbers in conjunction with the maintenance of pairwise (claimant, verifier) state information.

Containing TVP doesn't guarantee the freshness of a message or the liveness of a principal. A TVP in itself may be an old one (as we will illustrate in Example 3.39), so it is better for a TVP to be authenticated fresh by the legitimate participant himself.

The TVP should be associated with the legitimate participants of this run, and bound together with a message, so that the message could not be a replayed one of other run. The freshness and association of a TVP help designers to deem whether a message is fresh for this run. The freshness of a TVP means that the TVP is a new generated one in the current session, and the association of a TVP means that the TVP is associated with the legitimate participants of this run. The freshness and the association of a TVP (e.g., associated with A and B) guarantee that this TVP is a fresh TVP generated for this particular run (e.g., between A and B).

Timestamps in protocols offer the advantage of fewer messages and almost no state information. Timestamps may be used to provide timeliness guarantee and to detect message replay.

It seems charming to use timestamps in protocols. However, timestamps require the maintenance of secure, and synchronized timeclocks, which is difficult in distributed system.

The adversarial modification of distributed timeclocks is difficult to detect in many distributed environments. The adversarial modifications include resetting of a clock backwards so as to restore the validity of old messages, and setting of a clock forward to prepare a message for some future point in time. In both cases, the timeliness provided must be carefully re-evaluated[3]. Timestamps in protocols may typically be replaced by a random number

challenge plus a return message.

From a legitimate participant's point of view, the freshness and association of a TVP can only be deduced from a signed message that contains the TVP trusted by this participant itself. The signed message can be a signature of certain principal, or a message encrypted under a long-term shared key with data integrity protection, etc.

Example 3.16 Fig. 3.12 illustrates the Needham-Schroeder public-key protocol, let's analyze why the protocol is flawed.

$$\begin{array}{ll} \text{Message 1} & A \rightarrow B: \ \{A, N_A\}_{K_B} \\ \text{Message 2} & B \rightarrow A: \ \{N_A, N_B\}_{K_A} \\ \text{Message 3} & A \rightarrow B: \ \{N_B\}_{K_B} \end{array}$$

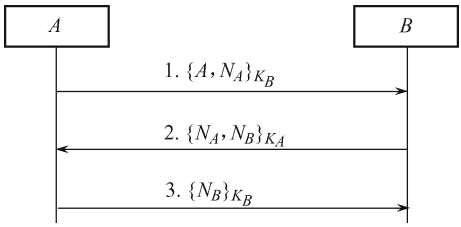


Fig. 3.12 The Needham-Schroeder public-key protocol.

Upon receiving Message 1, B believes that N_A is confidential.

Upon receiving Message 2, A believes that B is alive, and that N_A and N_B are associated with A and B, since only B can decrypt the message $\{A, N_B\}_{K_B}$ to obtain N_A and only A can decrypt the message $\{N_A, N_B\}_{K_A}$.

Upon receiving Message 3, B believes that A is alive and N_B is associated with A and B, but B still could not guarantee the freshness of N_A and the association of N_A with A and B. So an adversary I can launch an attack by confusing N_A. Hence, there exists the attack discovered by Lowe using FDR[33].

In order to fix the absence of the freshness of N_A and the association of N_A with A and B in this protocol, there should exist some evidence for B to authenticate these properties. The flaw can be fixed by changing Message 2 of the protocol to $\{B, N_A, N_B\}_{K_A}$[33]. At the second step, B believes that N_A is associated with N_B in a particular run of B. Upon receiving Message 3, B believes that A is alive and N_B is associated with A and B. Since B believes that N_A is associated with N_B, B believes that N_A is fresh and associated with A and B.

Example 3.17 The Needham-Schroeder shared key protocol[1], as shown in Fig. 3.13, intends to establish a shared new session key k_{AB} between two participants A and B, with the help of a server S.

Message 1 $A \rightarrow S: \ A, B, N_A$

Message 2 $S \rightarrow A: \ \{N_A, k_{AB}, B, \{k_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

Message 3 $A \rightarrow B: \ S, \{k_{AB}, A\}_{K_{BS}}$

Message 4 $B \rightarrow A: \ \{N_B\}_{k_{AB}}$

Message 5 $A \rightarrow B: \ \{N_B - 1\}_{k_{AB}}$

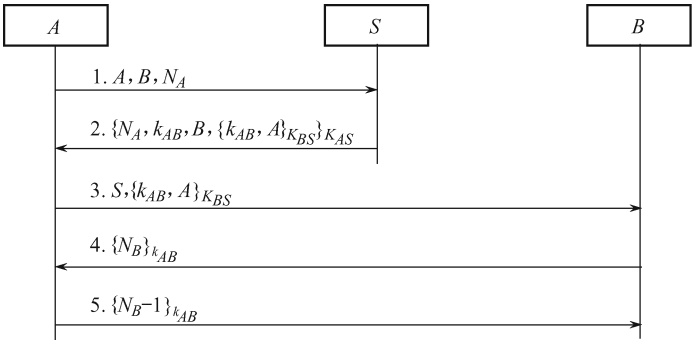


Fig. 3.13 The Needham-Schroeder shared key protocol.

Notation

A and B are principals; S is an online trusted third party; N_A, N_B are nonces; k_{AB} is a new session key established between A and B with the help of S; K_{AS} and K_{BS} are keys.

Premise

K_{AS} (K_{BS}) is the shared long-term key between A and S (B and S), which is initially established by non-cryptographic, and out-of-band techniques; N_A and N_B are randomly chosen by A and B respectively; k_{AB} is chosen by S as a new session key for A and B.

Protocol actions

1) In Message 1, A randomly chooses N_A for this protocol run, and tells S that A wants to establish a connection with B.

2) In Message 2, S randomly chooses k_{AB} for this run between A and B, and sends k_{AB} to A in encryption under K_{AS} and to B in encryption under K_{BS}.

3) Upon receiving Message 2, A checks N_A and believes S's presence for this run. Then A recovers k_{AB} from the encryption $\{N_A, k_{AB}, B\}_{K_{AS}}$ using K_{AS} and forwards $\{k_{AB}, A\}_{K_{BS}}$.

4) Upon receiving Message 3, B recovers k_{AB} from $\{k_{AB}, A\}_{K_{BS}}$ using K_{BS}, and believes the association of k_{AB} with A and B.

5) In Message 4, B randomly chooses N_B for this protocol run, and encrypts N_B to show his possession of the new session key k_{AB} to A.

6) In Message 5, A encrypts $\{N_B - 1\}$ to show his possession of the new session key k_{AB} to B.

7) Upon receiving Message 5, B checks N_B, and believes that A is alive from receiving of fresh nonce N_B.

Attack on the Needham-Schroeder shared key protocol

Upon receiving Message 3, B believes that k_{AB} is associated with A, but B is not sure of the freshness of k_{AB}. Upon receiving Message 5, B knows that someone who has k_{AB} has generated the message $\{N_B - 1\}_{k_{AB}}$, but he still could not authenticate the liveness of the principal A. Hence there exists an attack[4] as shown in Fig. 3.14.

Message 1 $I(A) \rightarrow S$: skip

Message 2 $S \rightarrow I(A)$: skip

Message 3 $I(A) \rightarrow B$: $S, \{k'_{AB}, A\}_{K_{BS}}$

Message 4 $B \rightarrow I(A)$: $\{N_B\}_{k'_{AB}}$

Message 5 $I(A) \rightarrow B$: $\{N_B - 1\}_{k'_{AB}}$

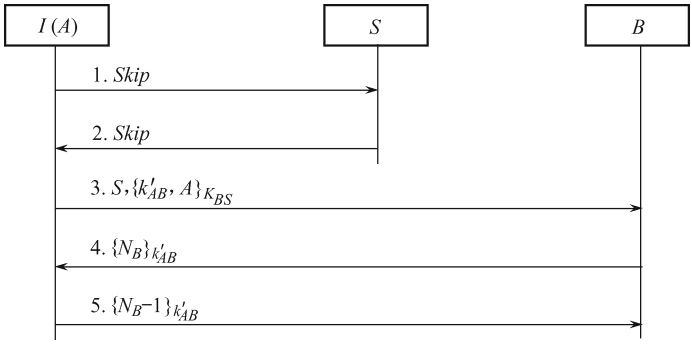


Fig. 3.14 An attack on the Needham-Schroeder shared key protocol.

Premise

k'_{AB} is a compromised session key between A and B. The adversary I has recorded $\{k'_{AB}, A\}_{K_{BS}}$ in Message 3 of an old protocol run.

Protocol actions

1) In Message 3, the adversary I impersonates A and replays the message $\{k'_{AB}, A\}_{K_{BS}}$ to B.

2) Upon receiving Message 3, B recovers k'_{AB} from $\{k'_{AB}, A\}_{K_{BS}}$ using K_{BS}, and believes the association of k'_{AB} with A and B.

3) In Message 4, B randomly chooses N_B for this protocol run, and encrypts N_B to show his possession of the new session key k'_{AB} to A.

4) In Message 5, $I(A)$ encrypts $\{N_B - 1\}$ to show his possession of the new session key k'_{AB} to B.

5) Upon receiving Message 5, B checks N_B, and believes that A (indeed, it is the adversary I) is alive from receiving of fresh nonce N_B and the A's possession of k'_{AB}.

Now B thinks that B is communicating with the opponent partner A, while in fact A knows nothing about it. In Message 3, there is no evidence showing the identification of A although the name of A is explicitly indicated. The absence of the liveness of A and the freshness of k_{AB} makes it possible to record Message 3 and to replay it in the current run.

It is important to understand properly where the TVP is needed and what it is associated with. Misuses of authentication about identities and TVPs are costly. Here is an example:

Example 3.18 Recall the Otway-Rees protocol in Example 3.31. In this protocol, N_A and N_B are associated with a particular run of this protocol between A and B. Since Message 1 is encrypted under the shared key between A and S, so the explicit mention of principal name A in Message 1 is unnecessary. But B should be explicitly mentioned in Message 1, otherwise an adversary I impersonating B can confuse N_A. As for N_B, it is similar to N_A. Figure 3.15 illustrates a fixed simplified version of Otway-Rees Protocol.

$$
\begin{array}{ll}
\text{Message 1} & A \to B: \ A, B, \{N_A, B\}_{K_{AS}} \\
\text{Message 2} & B \to S: \ A, B, \{N_A, B\}_{K_{AS}}, \{N_B, A\}_{K_{BS}} \\
\text{Message 3} & S \to B: \ \{N_A, k_{AB}\}_{K_{AS}}, \{N_B, k_{AB}\}_{K_{BS}} \\
\text{Message 4} & B \to A: \ \{N_A, k_{AB}\}_{K_{AS}}, \{A, B\}_{k_{AB}} \\
\text{Message 5} & A \to B: \ \{k_{AB}\}_{k_{AB}}
\end{array}
$$

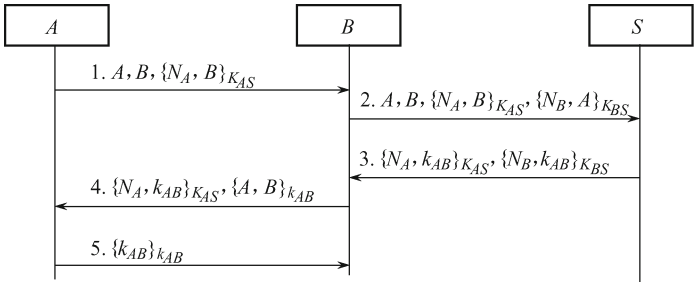


Fig. 3.15 A simplified version of the Otway-Ree key establishment protocol.

Protocol actions

1) In Message 1, A randomly chooses N_A for this protocol run, and A believes that N_A is fresh and associated with B.

2) In Message 2, B randomly chooses N_B for this run, and B believes that N_B is fresh and associated with A.

3) Upon receiving Message 2, S recovers N_A, N_B from the encryptions $\{N_A, B\}_{K_{AS}}$ and $\{N_B, A\}_{K_{BS}}$ using K_{AS} and K_{BS} respectively.

4) In Message 3, S randomly chooses a new session key k_{AB} for A and B, and sends $\{N_A, k_{AB}\}_{K_{AS}}$ to A and $\{N_B, k_{AB}\}_{K_{BS}}$ to B.

5) Upon receiving Message 3, B checks the freshness of the message $\{N_B, k_{AB}\}_{K_{BS}}$ via N_B and then recovers k_{AB} from it. Then B believes that k_{AB} is fresh and associated with A and B.

6) In Message 4, $\{A, B\}_{k_{AB}}$ is an evidence to authenticate B's presence to A.

7) Upon receiving Message 4, A checks the freshness of Message 4 via N_A and recovers k_{AB} from it. A believes that k_{AB} is fresh and associated with A and B. When A receives $\{A, B\}_{k_{AB}}$, A knows that it must be B who creates this encryption using the new session key k_{AB}, so A believes B's presence.

8) In Message 5, $\{k_{AB}\}_{k_{AB}}$ is attached to authenticate A's presence to B, since B believes that k_{AB} is fresh and associated with A.

9) Upon receiving Message 5, B knows that it must be A who creates this encryption using the new session key k_{AB}, so B believes A's presence.

Upon termination of this protocol run, B believes that S is alive from receiving fresh nonce N_B, and A believes that S is alive from receiving fresh nonce N_A, and B's presence is also guaranteed to A, A's presence is guaranteed to B in this protocol.

3.3.3 Data integrity protection of message

Principle 3.8 Correct data integrity protection of a message should be provided to ensure the liveness of principal or the freshness of a message.

A more important aspect of message authentication is data integrity protection. To ensure entity authentication, the freshness and association of a TVP, a proper data integrity protection service should be in the place[3, 13, 32].

Example 3.19 This is an attack discovered by Boyd and Mao[34] on a minor variation of the Otway-Rees Protocol.

The original Message 2 in the Otway-Rees protocol (see Example 3.31):

$$\text{Message 2} \quad B \to S: \quad M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$$

The variation of the Otway-Rees Protocol differs from the original one very slightly:

$$\text{Message 2} \quad B \to S: M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B\}_{K_{BS}}, \{M, A, B\}_{K_{BS}} \tag{3.3}$$

If the encryption primitive does not provide data integrity protection of a message, then the variation can be neglected since a long message is

always implemented in a plural number of blocks while encrypting. Figure 3.16 illustrates the attack by Boyd and Mao.

Message 1 $I(A) \rightarrow B : M, A, B, \{N_I, M, I, B\}_{K_{IS}}$
Message 2 $B \rightarrow I(S) : M, A, B, \{N_I, M, I, B\}_{K_{IS}}, \{N_B\}_{K_{BS}}, \{M, A, B\}_{K_{BS}}$
Message 2′ $I(B) \rightarrow S : M, I, B, \{N_I, M, I, B\}_{K_{IS}}, \{N_B\}_{K_{BS}}, \{M, I, B\}_{K_{BS}}$
Message 3 $S \rightarrow B : \quad M, \{N_I, k_{IB}\}_{K_{IS}}, \{N_B, k_{IB}\}_{K_{BS}}$
Message 4 $B \rightarrow I(A) : M, \{N_I, k_{IB}\}_{K_{IS}}$

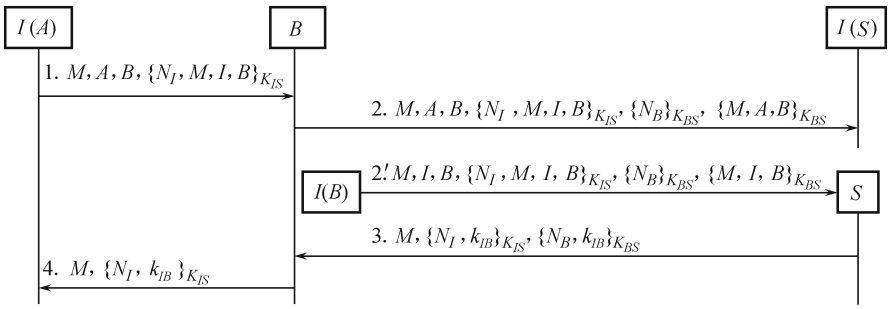


Fig. 3.16 The Boyd-Mao attack on the variation version of Otway-Rees protocol.

Notation

Besides the notations of the protocol in Fig. 3.8, the adversary I is a malicious legitimate principal who runs the Otway-Rees protocol with B in advance, and records some dialogs before. $I(A), I(B)$ and $I(S)$ are the adversary I impersonating the principals A, B and S respectively.

Premise

Besides the premises of the protocol in Fig. 3.8, K_{IS} is the shared long-term key between the adversaries I and S, which is initially established by non-cryptographic, and out-of-band techniques; N_I is randomly chosen by the adversary I; k_{IB} is chosen by S as a new session key for I and B. Note that $\{M, I, B\}_{K_{BS}}$ is an old cipher chunk, a part of Message 2 in (3.3), recorded by I.

Protocol actions

1) In Message 1, I randomly chooses N_I for this protocol run, and I (by impersonating A) tells B that A wants to establish a connection with B (with the same M as in $\{M, I, B\}_{K_{BS}}$). Note that the encryption of $\{N_I, M, I, B\}_{K_{IS}}$ cannot be read by B without the corresponding long-term key K_{IS}.

2) In Message 2, B randomly chooses N_B for this run, and tells S that A wants to establish a connection with B.

3) The adversary intercepts Message 2, and replaces $\{M, A, B\}_{K_{BS}}$ with $\{M, I, B\}_{K_{BS}}$ in Message $2'$.

4) Upon receiving Message $2'$, S recovers N_I, and N_B from the encryptions $\{N_I, M, I, B\}_{K_{IS}}$ and $\{N_B\}_{K_{BS}}$ using K_{IS} and K_{BS} respectively. S checks M to deem whether B is present or not. Since there is no data integrity protection of $\{M, A, B\}_{K_{BS}}$ in (3.3), B's presence cannot be guaranteed by $\{N_B\}_{K_{BS}}$ and $\{M, I, B\}_{K_{BS}}$.

5) In Message 3, S randomly chooses a new session key k_{IB} for I and B, and sends $\{N_I, k_{IB}\}_{K_{IS}}$ via B to $I(A)$ and $\{N_B, k_{IB}\}_{K_{BS}}$ to B.

6) Upon receiving Message 3, B checks the freshness of the message $\{N_B, k_{IB}\}_{K_{BS}}$ by N_B and then recovers k_{IB}.

7) Upon receiving Message 4, I checks the freshness of Message 4 by N_I and recovers k_{IB}.

As a result of this run, B believes that B has been talking to A and shares a session key with A. But in fact, B does this with I. This attack reveals an important point: the liveness of principals, the freshness and association of TVPs could not be achieved without proper data integrity protection of a message.

3.3.4 Stepwise refinement

Principle 3.9 The cryptographic protocol tends to be sound if it is implemented via stepwise refinement.

Stepwise refinement helps designers follow the basic principles in [3]: every message should say what it means. That is, first write down a straight forward English sentence describing the content of a message, then the interpretation of the message should depend only on its content and the whole contents of the cryptographic protocol are carried out step by step.

Boyd and Mao propose a refinement approach that uses two notations to express the precisely needed cryptographic services.

$\{m\}_k$ denotes an encryption of the message m under k, $[m]_k$ denotes a one-way transformation of the message m using the key k[13]. $[m]_k$ serves not only data integrity, but also message source identification.

A principal with k^{-1} which is the verification key matching k can verify the data-integrity correctness of $[m]_k$ and identify the message source. The verification procedure outputs "YES" or "NO": in the "YES" case, $[m]_k$ is deemed to have the correct data integrity and m is deemed to be a recognizable message from the identified source; in the "NO"case, $[m]_k$ is deemed to have an incorrect data integrity and m is deemed to be unrecognizable[13].

Remark

1) In practice, $[m]_k$ can be realized by a signed message, etc. for the case of asymmetric technique realization, a block cipher with MAC, a keyed hash, etc. for the case of symmetric technique realization. Hence, $[m]_k$ denotes a one-way transformation with trapdoor in most cases. So in this book, when we refer to "one-way transformation" we always mean "loose one-way transformation", i.e., it can be one-way transformation with trapdoor.

2) It is noticed that we do not distinguish $[m]_k$ from $\{m\}_k$ in this book, since $[m]_k$ can provide data integrity security and data origin authentication security, otherwise, the cryptographic algorithms without these properties will be no use to guarantee the security of cryptographic protocols. Hence in this book, when we refer to "$\{m\}_k$" we always mean "$[m]_k$", if these cryptographic algorithms in cryptographic protocols are one-way transformations.

The refined specifications are useful in stepwise refinement. Stepwise refinement helps designers achieve soundness of the cryptographic protocol.

Example 3.20 Recall the Needham-Schroeder public-key authentication protocol[1] (ref. Example 1.2), and there exists an attack on this protocol[2]. The above flaw can be fixed via stepwise refinement as follows:

$$\text{Step 1: Message 1} \quad A \to B: \quad \{A, N_A\}_{K_B}$$

Message 1 wants to say that A sends the principal name A and a nonce generated by himself to B, hoping that only B can decrypt it. According to the above Principles 6, 7, 8, principal name A and the nonce N_A should be authenticated and provided with data integrity protection, thus Message 1 can be refined to:

$$\text{Message 0–1} \quad A \to B: \quad \left\{\{A, N_A\}_{K_A^{-1}}\right\}_{K_B}$$

In this message, only N_A should be kept secret, and only B can recover N_A from the encryption. The encryption using K_B could not provide data integrity protection, while the signature of A could, so we can change Message 1 to:

$$\text{Message 1–1} \quad A \to B: \quad \left\{A, \{N_A\}_{K_B}\right\}_{K_A^{-1}}$$

Step 2: Message 2 $\quad B \to A: \quad \{N_A, N_B\}_{K_A}$

Message 2 wants to say that B has got N_A by using B's private key K_B^{-1} (only B can decrypt it, so he is really B), and B wants to send N_A back with a new nonce N_B generated by B himself to A using A's public-key, hoping that A can check N_A and recover N_B from the encryption. According to the above principles 6,7,8, N_A and N_B should be authenticated and provided with data integrity protection, thus Message 2 can be refined to:

$$\text{Message 0–2} \quad B \to A: \quad \left\{N_A, \{N_B\}_{K_B^{-1}}\right\}_{K_A}$$

Likewise in Message 1, we can change Message 2 to:

$$\text{Message 1–2} \quad B \to A: \quad \left\{\{N_A, N_B\}_{K_A}\right\}_{K_B^{-1}}$$

Step 3: Message 3 $A \to B: \{N_B\}_{K_B}$

Message 3 wants to say that A has got N_B by using A's secret key K_A^{-1} (only A can decrypt it, so he is really A), and A wants to send N_B to B using B's public-key, hoping that B can check N_B from the encryption. N_B should be kept secret by using B's public-key K_B, and N_B should be provided with data integrity protection by using A's private key K_A^{-1}, thus Message 3 can be refined to:

$$\text{Message 0–3} \quad A \to B: \quad \left\{\{N_B\}_{K_A^{-1}}\right\}_{K_B}$$

Likewise in Message 1, we can change Message 3 to:

$$\text{Message 1–3} \quad A \to B: \quad \left\{\{N_B\}_{K_B}\right\}_{K_A^{-1}}$$

1. The refinement 0-* of Needham-Schroeder public-key protocol

Now, we have the revised Needham-Schroeder public-key 0-* protocol as shown in Fig. 3.17.

$$\text{Message 0–1} \quad A \to B: \quad \left\{\{A, N_A\}_{K_A^{-1}}\right\}_{K_B}$$

$$\text{Message 0–2} \quad B \to A: \quad \left\{N_A, \{N_B\}_{K_B^{-1}}\right\}_{K_A}$$

$$\text{Message 0–3} \quad A \to B: \quad \left\{\{N_B\}_{K_A^{-1}}\right\}_{K_B}$$

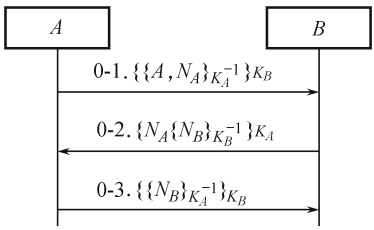


Fig. 3.17 The revised Needham-Schroeder public-key 0-* protocol.

Lowe's attack on the original protocol won't work on 0-* protocol as shown in Fig. 3.18.

$$\text{Message 1} \quad A \to I: \quad \left\{\{A, N_A\}_{K_A^{-1}}\right\}_{K_I}$$

$$\text{Message 1}' \quad I(A) \to B: \quad \left\{\{A, N_A\}_{K_A^{-1}}\right\}_{K_B}$$

$$\text{Message 2}' \quad B \to I(A): \quad \left\{N_A, \{N_B\}_{K_B^{-1}}\right\}_{K_A}$$

$$\text{Message 2} \quad I \to A: \quad \left\{N_A, \{N_B\}_{K_B^{-1}}\right\}_{K_A}$$

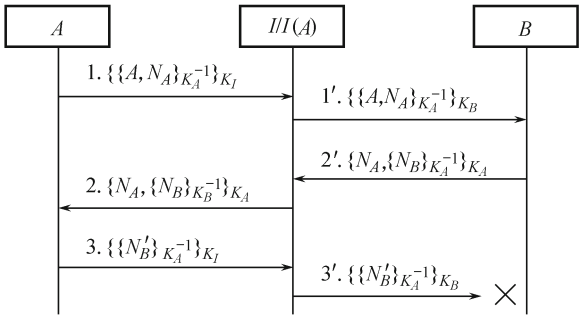


Fig. 3.18 Attack on the Needham-Schroeder public-key 0-* protocol.

$$\text{Message 3} \quad A \rightarrow I: \quad \left\{\{N_B'\}_{K_A^{-1}}\right\}_{K_I}$$

$$\text{Message 3}' \quad I(A) \rightarrow B: \quad \left\{\{N_B'\}_{K_A^{-1}}\right\}_{K_B}$$

Notation

A and B are two protocol principals, and I is an attacker with legitimate identity; N_A and N_B are secret symmetric keying materials chosen by A and B, respectively. K_A^{-1} and K_A are A's private key and public-key, K_B^{-1} and K_B are B's private key and public-key, and K_I^{-1} and K_I are I's private key and public-key. $I(A)$ is the adversary I impersonating the principal A.

Premise

K_A, K_B, and K_I are A, B and I's authentic public-keys respectively; K_A^{-1}, K_B^{-1} and K_I^{-1} are A, B and I's authentic private key only known by itself. They all are initially established by non-cryptographic, and out-of-band techniques.

Actions

1) In step 1, A intends to launch a session between A and I. Firstly, A randomly chooses a nonce N_A, and signs $\{A, N_A\}_{K_A^{-1}}$ using A' private key K_A^{-1} to provide data integrity protection for $\{A, N_A\}$. A encrypts $\{A, N_A\}_{K_A^{-1}}$ using the adversary I's public-key K_I to keep N_A secret to all but I.

2) In step 1′, I recovers $\{\{A, N_A\}_{K_A^{-1}}\}_{K_I}$ using I's private key K_I^{-1} and then encrypts the message using B's public-key K_B, and tries to establish a bogus session with B by impersonating A.

3) Upon receiving Message 1′, B recovers N_A using B's private key K_B^{-1} and A's public-key K_A.

4) In step 2′, B randomly chooses a new nonce N_B and signs it, and then encrypts $\{N_A, \{N_B\}_{K_B^{-1}}\}$ using A's public-key K_A in order to keep N_B secret to all but A.

5) The adversary I intercepts Message $2'$ $\{N_A, \{N_B\}_{K_B^{-1}}\}_{K_A}$, but couldn't recover N_B without A's private key K_A^{-1}. Hence, I just passes $\{N_A, \{N_B\}_{K_B^{-1}}\}_{K_A}$ to A.

6) Upon receiving Message 2, A checks N_A and recovers N'_B (an error N_B from $\{N'_B\}_{K_B^{-1}}$ using I's public-key K_I).

7) A signs N'_B and encrypts $\{N'_B\}_{K_A^{-1}}$ using I's public-key K_I to keep N'_B secret to all but I.

8) I recovers $\{N'_B\}_{K_A^{-1}}$ from $\{\{N'_B\}_{K_A^{-1}}\}_{K_I}$, encrypts $\{N'_B\}_{K_A^{-1}}$ using B's public-key K_B, and sends it to B.

9) Upon receiving Message $3'$, B deems that the session between A and B is bogus since N_B is an error N'_B.

2. The refinement 1-* of Needham-Schroeder public-key protocol

Figure 3.19 illustrates the revised Needham-Schroeder public-key 1-* protocol.

$$\begin{aligned}
&\text{Message 1–1} \quad A \to B: \quad \left\{A, \{N_A\}_{K_B}\right\}_{K_A^{-1}} \\
&\text{Message 1–2} \quad B \to A: \quad \left\{\{N_A, N_B\}_{K_A}\right\}_{K_B^{-1}} \\
&\text{Message 1–3} \quad A \to B: \quad \left\{\{N_B\}_{K_B}\right\}_{K_A^{-1}}
\end{aligned}$$

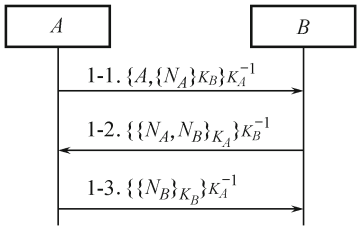


Fig. 3.19 The revised Needham-Schroeder public-key 1-* protocol.

Lowe's attack on the original protocol won't work on 1-* protocol (see Fig. 3.20).

$$\begin{aligned}
&\text{Message 1} \quad A \to I: \quad \left\{\{A, N_A\}_{K_I}\right\}_{K_A^{-1}} \\
&\text{Message 1}' \quad I(A) \to B: \quad \left\{\{A, N_A\}_{K_I}\right\}_{K_A^{-1}}
\end{aligned}$$

Notation

Same as the Needham-Schroeder public-key 0-* protocol.

Premise

Same as the Needham-Schroeder public-key 0-* protocol.

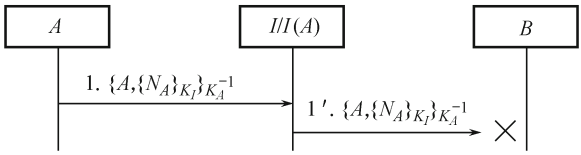


Fig. 3.20 Attack on the Needham-Schroeder public-key 1-* protocol.

Actions

1) In step 1, A intends to launch a session between A and I. Firstly, A randomly chooses a nonce N_A, and encrypts it using the adversary I's public-key K_I, then signs $\{A, \{N_A\}_{K_I}\}$ using A's private key K_A^{-1} and sends the signed message to I. The signature of A can provide data integrity protection for $\{A, \{N_A\}_{K_I}\}$.

2) In step $1'$, I passes the message to B, and tries to establish a bogus session with Bob by impersonating A.

3) Upon receiving Message $1'$, B decrypts $\{A, N_A\}_{K_I}$ using B's private key K_B^{-1}, and B could not recognize the identity name A. Hence B will abort this protocol run.

As we have seen, I could not even launch an denial of service attack on the Needham-Schroeder public-key 1-* protocol.

3.4 Provable security

Principle 3.10 It is prudent to prove security of a cryptographic protocol using formalisms. However, the protocol designers should be clear to the constraints of the formalism used. Weakened assumptions should be used if a proof of security can be derived on it.

Formal analysis of cryptographic protocol security is important in cryptographic protocol design, and it is similar to the software test in software engineering. It helps designers to determine the security strength reached by the cryptographic protocol, to find flaws in the protocol.

Formal analysis, especially mathematical provable security, requires a designer or an analyzer to use correct or more precise cryptographic services, so protocol flaws due to misuse of cryptographic services will become less frequent with the provable security in consideration.

Each formal protocol analysis methodology has its own advantages and limitations, and a protocol analysis approach may be applicable only to a subset of protocols or classes of attacks.

It is important to select an appropriate formalism for each special cryptographic protocol. As for some authentication protocols, formalisms such as Burrows-Abadi-Needham (BAN) logic, strand spaces and model check-

ing are suitable[7, 8, 35]. As for public-key schemes, digital signatures etc., security proofs under ROM are better[9, 14].

Although security validation via formal methods is consistent and useful[36–38], a formal analysis approach in itself may contain implicit assumptions, hence it may be subject to subtle flaws[13, 32]. Protocol designers should be clear to the constraints of the formalism used.

The critical process of converting a concrete protocol into a formal specification still may lead subtle flaws to the analysis[38]. We often assume that analyzers can accurately translate a protocol into rules about security requirements and transmitted messages. But, in fact, we are still lack a method for deciding whether a given set of rules captures "enough" properties of an underlying cryptosystem[22, 23]. A proof of security correctness is often specially tailored for target protocol to be proved, which may introduce some informal assumptions based on human ingenuity[13].

Example 3.21 The initial assumptions of BAN logic usually come from designers' experience and are lack of a formal definition; the protocol specification's idealization for an underlying semantics is also informal, while the security of the protocol idealization is the basis of the soundness of the BAN logic axioms system. Hence people will believe that there are flaws in the protocol if BAN logic has discovered these, but they cannot believe that the protocol is actually secure when BAN logic has proved that. That is, the absence of discovered flaws does not imply the absence of flaws. More, Mao has observed that BAN logic provides a context-free procedure for protocol formulation. For example, suppose a trusted third party S chooses a new session key k_{AB} as a good key for communication between the principals A and B, so the message sending step may be idealized into: S said $A \overset{k_{AB}}{\longleftrightarrow} B$. This protocol formulation is subtle, since the context of the protocol has not been considered. That is, who really sends this message? Is there any authentication mechanism for sender or receiver? Is there any secure mechanism for k_{AB}? We have got no guarantee from this formulation[13].

Example 3.22 Since most of the formalisms suffer from a limitation of fixed time and space resources, formal analysis of a protocol tends to be applied in an isolated run instance. Some formalism requires a formal model of the behavior of adversaries, but it is difficult to be given accurately[35]. The simplifications undoubtedly imply inaccuracies, perhaps mistakes.

Example 3.23 Computational security defines notions of security so that we know what to aim for and what to expect, and we can prove how the security of a scheme relates to the security of its primitives. Provable security in itself is based on some assumptions. In Random Oracle Model, we usually assume that all random values are indeed random, adversary does not exploit any properties of a hash function, and hash functions behave idealistically. In essence, to hash a message has only added quality redundancy to the message in a deterministically verifiable manner. While a deterministic hash

function can never "amplify" entropy according to Shannon's entropy theory, so random oracle does not exist in the real world. Thus, a real-world hash function only emulates the random oracle behavior to a precision where the difference is hopefully a negligible quantity[32]. Hence, a rigorous argument on the security of a cryptographic scheme requires a formal model of the behavior of a hash function, while it is difficult to give a formal hash model accurately. Assumptions reduce the strength of a provable security proof.

Perhaps a cryptographic scheme is secure in practice even though a reduction may not exist. Perhaps the scheme is in fact insecure, but an attack has not been discovered. Perhaps a reduction can be found by modifying the scheme slightly, and we can regard this reduction as a type of assurance about the original cryptographic protocol.

A lot of assumptions are made about the abilities of an adversary, and the behavior of the adversary are hard to be expressed in provable security. The adversary may guess password via off-line password dictionary guessing, replay a message from a last run of a protocol, execute protocols concurrently, etc. It is difficult to express this adversary's knowledge accurately since the abilities of an adversary are incremental.

Cryptographic scheme using weaker assumptions provides a higher security confidence than that using stronger assumptions. For example, a proof under standard intractability assumptions is a formal proof of security for a public-key cryptosystem relying solely on the intractability of the underlying one-way trapdoor transformation. It is a strong security established in the real world. That is to say, if the underlying intractability assumptions could not be broken, then a cryptosystem will not be broken too. This is an exact security property.

References

[1] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[2] Lowe G (1995) An Attack on the Needham-Schroeder Public Key Authentication Protocol. Information Processing Letters 56(3): 131 – 133

[3] Abadi M, Needham R (1996) Prudent Engineering Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering 22(1): 6 – 15

[4] Denning DE, Sacco GM (1981) Timestamps in Key Distribution Protocols. Communication of the ACM 24(8): 533 – 536

[5] Boyd C (1990) Hidden Assumptions in Cryptographic Protocols. IEE Proceedings Part E: Computers and Digital Techniques 137(6): 433 – 436

[6] Clark J and Jacob J (1997) A Survey of Authentication Protocol Literature: Version 1.0. http://www.win.tue.nl/∼ecss/downloads/clarkjacob.pdf. Accessed Nov 1997

[7] Burrows M, Abadi M, Needham R (1990) A Logic of Authentication. ACM Transactions on Computer Systems 8(1): 18 – 36

[8] Lowe G (1999) Towards a Completeness Result for Model Checking of Security Protocols. Journal of Computer Security 7(2 – 3): 89 – 146

[9] Bellare M, Rogaway P (1993) Entity Authentication and Key Distribution. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 773. Springer-Verlag, pp 232 – 249

[10] Canetti1 R, Meadows C, Syverson P (2002) Environmental Requirements for Authentication Protocols. In: ISSS'02 Proceedings of the 2002 Mext-NSF-JSPS International Conference on Software Security: Theories and Systems, Tokyo, 8 – 10 Nov 2002

[11] Abadi M (1998) Two Facets of Authentication. In: Proceedings of the 11th IEEE Computer Security Foundations Workshop, Rockport, 9 – 11 June 1998

[12] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[13] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[14] Goldwasser S, Micali S (1984) Probabilistic Encryption. Journal of Computer and System Sciences 28(2): 270 – 299

[15] Bellare M, Rogaway P (1994) Optimal Asymmetric Encryption. In: EUROCRYPT'94 Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology, Perugia, 9 – 12 May 1994. Lecture Notes in Computer Science, vol 950. Springer-Verlag, pp 92 – 111

[16] Shoup V (2001) OAEP Reconsidered. In: CRYPTO'01 Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 19 – 23 Aug 2001. Lecture Notes in Computer Science, vol 2139, pp 239 – 259, Springer

[17] Fujisaki E, Okamoto T, Pointcheval D, Stern J (2001) RSA-OAEP is Secure Under the RSA Assumption. In: CRYPTO'01 Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 19 – 23 Aug 2001. Lecture Notes in Computer Science, vol 2139, pp 260 – 274, Springer

[18] Cramer R, Shoup V (1998) A Practical Public Key Cryptosystem Provably Secure Against Chosen Ciphertext Attack. In: CRYPTO'98 Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 23 – 27 Aug 1998. Lecture Notes in Computer Science, vol 1462. Springer-Verlag, pp 13 – 25

[19] ElGamal T (1985) A Publi – Key Cryptosystem and Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory 31(4): 469 – 472

[20] Fouque PA, Howgrave-Graham N, Martinet G, Poupard G (2003) The Insecurity of Esign in Practical Implementations. In: ASIACRYPT'03 Proceedings of 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, 30 Nov – 4 Dec 2003. Lecture Notes in Computer Science, vol 2894, pp 492 – 506, Springer

[21] Dolev D, Yao AC (1983) On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2): 198 – 208

[22] Canetti R, Krawczyk H, Nielsen JB (2003) Relaxing Chosen-Ciphertext Security. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 2729, pp 565 – 582, Springer

[23] Herzog J, Liskov M, Micali S (2003) Plaintext Awareness via Key Registration. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 2729, pp 548 – 564, Springer

[24] Katz J, Ostrovsky R, Yung M (2001) Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In: EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045, pp 475 – 494, Springer

[25] Vaudenay S (2002) ecurity Flaws Induced by CBC Padding – Application to SSL, IPSEC, WTLS. In: EUROCRYPT'02 Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology, Amsterdam, 28 Apr – 2 May, 2002. Lecture Notes in Computer Science, vol 2332, pp 534 – 546, Springer

[26] GPG. The GNU privacy guard. http://www.gnupg.org. Accessed 18 Nov 2010

[27] Nguyen PQ (2004) Can We Trust Cryptographic Software Cryptographic Flaws in GNU Privacy Guard v1.2.3. In: EUROCRYPT'04 Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology, Interlaken, 2 – 6 May 2004. Lecture Notes in Computer Science, vol 3027, pp 555 – 570, Springer

[28] Mitchell JC, Shmatikov V, Stern U (1998) Finite State Analysis of SSL 3.0. In: Proceedings of the 7th USENIX Security Symposium, San Antonio, 26 – 29 Jan 1998

[29] Freier AO, Karlton P, Kocher PC (1996) The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt. Accessed 18 Nov 2010

[30] Woo TYC, Lam SS (1992) Authentication for Distributed Systems. Computer 25(1): 39 – 52

[31] Carman DW, Kruus PS, Matt BJ (2000) Constraints and Approaches for Distributed Sensor Network Security. NAI Labs Technical Report # 00–010, 1 Sept 2000

[32] Otway D, Rees O (1987) Efficient and Timely Mutual Authentication. Operating Systems Review 21(1): 8 – 10

[33] Lowe G (1996) Breaking and Fixing the Needham-Schroeder Public-key Protocol Using FDR. In: TACAS'96 Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Passau, 27 – 29 Mar 1996. Lecture Notes in Computer Science (Lecture Notes in Software Configuration Management), vol 1055, pp 147 – 166, Springer

[34] Boyd C, Mao W (1993) On a Limitations of BAN Logic. In: EUROCRYPT'93 Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology, Lofthus, 23 – 27 May 1993. Lecture Notes in Computer Science, vol 765, pp 240 – 247, Springer

[35] Fabrega FJT, Herzog JC, Guttman JD (1998) Strand Spaces: Why is a Security Protocol Correct? In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, 3 – 6 May 1998

[36] Canetti R, Krawczy H (2001) Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In: EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045, pp 453 – 474, Springer

[37] Stern J (2003) Why Provable Security Matters. In: EUROCRYPT'03 Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology, Warsaw, 4 – 8 May 2003. Lecture Notes in Computer Science, vol 2656, pp 449 – 461, Springer

[38] Mao W, Boyd C (1993) Towards a Formal Analysis of Security Protocols. In: Proceedings of the Computer Security Foundations Workshop VI, Franconia, 15 – 17 June 1993. IEEE Computer Society Press, Washington, pp 147 – 158

4 Informal Analysis Schemes of Cryptographic Protocols

Abstract Four security definitions about unilateral authentication secure, mutual authentication secure, unilateral session key secure, or mutual session key secure are given respectively under the computational model of matching conversation and indistinguishability. An informal analysis approach based on trusted freshness is presented, and the analysis results suggest the correctness of a protocol or the way to construct attacks intuitively from the absence of security properties. Then, the reasons why typical attacks on authentication protocols exist are discussed based on trusted freshness, and corresponding examples are illustrated to corroborate the discussion.

Due to the asynchronous nature of contemporary networks, establishing whether the security of an authentication protocol is adequate or not is far more difficult than one may have initially imagined[1—15]. A variety of useful rigorous ways have been developed for analyzing and reasoning about cryptographic protocols and they have been proved much useful while designing and analyzing a cryptographic protocol[16—21].

However, there are some important issues that still lack satisfactory treatment. (1) What's the efficient way to distinguish whether a message is fresh or not, to prevent replay attacks, parallel attacks and interleaving attacks. For example, Burrows, Abadi and Needham presented the famous BAN logic[7], which states: the formula X has not been sent in a message at any time before the current run of the protocol, then X is fresh. This is subtle, hence there exists an interleaving attack which is also a replay attack on Needham-Schroeder public key protocol (Needham-Schroeder protocol for short) even if the Needham-Schroeder protocol could be proved secure by BAN logic[1—3, 7, 8, 22]. (2) How to avoid the dependency of analysis on the idealization of a protocol, the concrete formalization of attackers' possible behaviors and the formalization of concurrent runs of protocols. (3) What are the precise specifications of the guarantee for the security of an authentication protocol, which prove the correctness of the protocol sufficiently and necessarily.

In this chapter, 4 security definitions about unilateral authentication se-

cure, mutual authentication secure, unilateral session key secure, or mutual session key secure are given respectively under the computational model of matching conversation and indistinguishability. The presented conditions to guarantee the security adequacy of unilateral entity authentication protocols, mutual entity authentication protocols, unilateral key establishment protocols and mutual key establishment protocols are proved. A novel idea of protocol security analysis is presented based on trusted freshness, which is called the freshness principle. Security analysis based on trusted freshness can efficiently distinguish whether a message is fresh or not, and the analysis results suggest the correctness of a protocol or the way to construct attacks intuitively from the absence of security properties. The reasons why typical attacks on authentication protocols exist are discussed based on trusted freshness, and corresponding examples are illustrated to corroborate the discussion.

4.1 The security of cryptographic protocols

Recall the security assumptions of the cryptographic protocols. Suppose there exists a probabilistic polynomial time (PPT) attacker I that has full control of the communication links as described in Dolev-Yao threat model[23]. Besides this, suppose that the Dolev-Yao attacker I can also launch the Adaptive Chosen Ciphertext Attacks (CCA2) without limitations. Suppose the cryptographic primitives are secure against Indistinguishable Adaptive Chosen Ciphertext Attack (IND-CCA2). That is, in IND-CCA2 security strength, the failures in cryptographic protocols are not in any way related to the strength or weakness of a particular cryptographic primitive used (that is, the cryptographic primitives are perfect in this attack model), but related to the protocol logic flaws, which permit the attacker to break the security goals of cryptographic protocols without necessarily breaking the particular cryptographic primitives used. Suppose that a legitimate party is either totally corrupted or totally secure. Suppose that each participant has his own private key and all other parties' public keys (respectively, the shared long-term key between co-operative principals or trusted third parties) in public-key case (respectively, in symmetric-key case), which are deployed safely before the cryptographic protocol run via non-cryptographic, and out-of-band techniques. Furthermore, private keys and shared keys are commonly assumed to be too long to guess in a computationally feasible way. In general, an authentication protocol is considered flawed if a principal concludes a normal run of the protocol with its intended communication partners while the intended partner would have a different conclusion. This book mainly discusses Challenge-Response authentication protocols.

4.1.1 Authenticity and confidentiality under computational model

The security definition under computational model provides a high confidence of the security of a cryptosystem.

Definition 4.1 A conversation is a sequence of timely ordered messages that a participant sent out (respectively, received), and as consequent responses, received (respectively, sent). Let $\tau_1 < \tau_2 < \cdots \tau_n$ be a time sequence recorded by the participant when it converses. The conversation can be denoted by the following sequence: $conv = (\tau_1, m_1, m_1'), (\tau_2, m_2, m_2'), \cdots, (\tau_n, m_n, m_n')$[17, 24].

This sequence encodes that at time τ_1, the participant was asked m_1 and responded with m_1', and then, at some later time $\tau_2 > \tau_1$, the participant was asked m_2, and responded with m_2', and so on, until, finally, at time τ_n it was asked m_n, and responded with m_n'. If $m_1 =$"", that is no message input, then the participant is the initiator, otherwise, it is called the responder. If $m_n =$"", there is no message output, then the participant ends the conversation. At the end of a protocol run, each participant makes a decision about the authentication of the intended partner: accept, reject, or is undetermined.

Definition 4.2 Suppose there exists a cryptographic protocol run between principals A and B. Let $conv = (\tau_0, "", m_1), (\tau_2, m_1', m_2), (\tau_4, m_2', m_3), \cdots, (\tau_{2(t-1)}, m_{t-1}', m_t)$ be a conversation of A. It is said that B has a conversation $conv'$ which matches $conv$ if there exists time sequence $\tau_1 < \tau_2 < \cdots \tau_n$ and $conv' = (\tau_1, m_1, m_1'), (\tau_3, m_2, m_2'), (\tau_5, m_3, m_3'), \cdots, (\tau_{2t-1}, m_t, m_t')$ where $m_t =$"no message output". These two conversations are called matching conversations[17, 22, 24, 25].

Given a protocol Π between principals A and B, if a principal like A (or B) with a conversation $conv$ believes that B (or A) always has a conversation $conv'$ which matches $conv$ whenever it is allowed to complete a protocol run, then this authentication protocol is secure from the point of view of A (or B). Here the attacker wins if the principal A or B has reached "accept" decision while A or B does not have a matching conversation in B or A.

We follow the probabilistic indistinguishability definitional approach[25] presented by Goldwasser and Micali to define confidentiality security. Here, that the attacker has broken the scheme means that: without breaking any cryptographic primitive and knowing the corresponding key, the attacker can still learn something about the established new session key under the run of a cryptographic protocol. Here "learn" is defined as distinguishing the value of a key generated by the cryptographic protocol from an independent randomly chosen key.

4.1.2 Security definitions

Based on the security definition of authenticity, the Unilateral entity Authentication Secure definition (UA-Secure) and Mutual entity Authentication Secure definition (MA-Secure) are presented; based on the security definition of authenticity and confidentiality, the Unilateral authenticated Key Secure (UK-Secure) and Mutual authenticated Key Secure (MK-Secure) are presented[19].

Definition 4.3 (UA-Secure) Unilateral entity Authentication Secure: an authentication protocol Π is called UA-Secure from the point of view of A if the attacker cannot win with a non-negligible probability for any attacker I in Dolev-Yao threat model with the assistance of cryptanalysis training course. Here the attacker wins if the principal A has reached "accept" decision while A does not have a matching conversation in B.

Actually, according to the notion matching conversation, we can only prove that the intended principal has correctly responded to the challenge of the current protocol run, that is, the intended principal is present, but we can not prove that the intended principal has participated in this protocol run where the challenge is generated. Hence, there exists Man-in-the-Middle attack even the presence of the intended principal has been proved. To correct this problem, the challenge should be associated with all the participants in the protocol, then the identity of the intended principal could not be authenticated.

Definition 4.4 (MA-Secure) Mutual entity Authentication Secure: an authentication protocol Π is called MA-Secure if the attacker cannot win with a non-negligible probability for any attacker I in Dolev-Yao threat model with the assistance of cryptanalysis training course. Here the attacker wins if any principal A or B has reached "accept" decision while A or B does not have a matching conversation in B or A.

Definition 4.5 (UK-Secure) Unilateral authenticated Key Secure: let k be the value of the corresponding new session key. We toss a coin b, $b \xleftarrow{R} \{0,1\}$. If $b = 0$, we provide the attacker I with the value k. Otherwise we provide the attacker I with a value r randomly chosen from the probability distribution of keys generated by protocol Π. At the end of its run, the attacker I outputs a bit b' (as its guess for b). An authentication protocol Π is called UK-Secure from the point of view of A if the following properties hold for any attacker I in Dolev-Yao threat model with the assistance of cryptanalysis training course:

1) If uncorrupted party A believes that A has completed a session with the intended opposite party B, then A trusts that the uncorrupted party B must have responded to the same session, and they both output the same key k.

2) The probability that the attacker I guesses correctly the bit b (i.e., outputs $b' = b$) is no more than $1/2$ plus a negligible fraction in the security parameter.

Definition 4.6 (MK-Secure) Mutual authenticated Key Secure (SK-Secure[19]): let k be the value of the corresponding session key. We toss a coin b, $b \xleftarrow{R} \{0,1\}$. If $b = 0$, we provide the attacker I with the value k. Otherwise we provide the attacker I with a value r randomly chosen from the probability distribution of keys generated by protocol Π. At the end of its run, the attacker I outputs a bit b' (as its guess for b). An authentication protocol Π is called MK-Secure if the following properties hold for any attacker I in Dolev-Yao threat model with the assistance of cryptanalysis training course:

1) Protocol Π satisfies the property that if two uncorrupted parties complete matching sessions then they both output the same key.

2) The probability that the attacker I guesses correctly the bit b (i.e., outputs $b' = b$) is no more than $1/2$ plus a negligible fraction in the security parameter.

Recall that each party creates and maintains a local state for a particular protocol run until a session ends its run. To corrupt a party means that the attacker learns all the internal memory of that party including long-term secrets (such as private keys or master shared keys used across different sessions) and session-specific information contained in the party's memory (such as internal state of incomplete sessions and session-keys corresponding to completed sessions).

Mutual authenticated key secure includes mutual authenticated key transport secure (for mutual authenticated key transport) and mutual authenticated key agreement secure (for mutual authenticated key agreement).

4.2 Security mechanism based on trusted freshenss

A novel idea of protocol security analysis based on trusted freshness will be presented in this section[26]. The presentations include the freshness principle, which is the key of the security analysis based on trusted freshness; the substantial and necessary requirements to meet four computational security definitions; a manual analysis method based on the freshness principle, and an illustration of the analysis method based on trusted freshness.

4.2.1 Notions

Some notions used in the security analysis method based on trusted freshness will be given in this subsection.

Definition 4.7 (Freshness) From the point of view of a participant in a protocol run, freshness means that a freshness identifier or a message is confirmed to be new for a particular run of the protocol. If a freshness identifier is new generated by the principal itself for this run or a message is conveyed with this new generated identifier via a one-way transformation or a trapdoor one-way transformation, then the freshness of the freshness identifier or the message is confirmed.

Definition 4.8 (Trusted freshness) A trusted freshness identifier (also called trusted freshness) is a freshness identifier whose freshness has been confirmed via generation of the principal itself or via a one-way transformation or a trapdoor one-way transformation including a trusted freshness identifier. The trusted freshness identifiers include trusted nonces, trusted timestamps, trusted session keys or trusted shared parts of a session key. Note that the trusted freshness identifiers are different for different protocol runs, are different for each participant in the same protocol run, and are different for the same principal in different protocol runs.

Definition 4.9 (Fresh message) From the point of view of a participant in a protocol run, a fresh message is a sent or received message that includes a trusted freshness.

Definition 4.10 (Term) A term $\hat{m}$ is a fresh message owned by a principal that may be exchanged in a particular protocol run. A term set $\hat{M}$ is the collection of all terms in a protocol run. Terms can be recursively defined as:

1) If $\hat{m}$ is a trusted freshness identifier, then $\hat{m}$ is a term.

2) If $\hat{m}$ is a term, o is a principal identity or a freshness identifier, and $\{\hat{m}, o\}$, or $\{o, \hat{m}\}$ is encrypted or should be decrypted via a one-way transformation or a trapped door one-way transformation, then $\{\hat{m}, o\}$, or $\{o, \hat{m}\}$ is a term. $\{\hat{m}, o\}$ and $\{o, \hat{m}\}$ are regarded as the same term in the security analysis approach based on trusted freshness.

3) If $\hat{m}$ is a term, k is a cryptographic key known by the principal, and $\{\hat{m}\}_k$ is encrypted or should be decrypted via a one-way transformation or a trapped door one-way transformation, then $\{\hat{m}\}_k$ is a term. If $\hat{m}$ is a term, o is a principal identity or a freshness identifier, then $\{o\}_{\hat{m}}$ is a term.

A maximal term is the longest term which is constructed from a single message of a particular protocol run via the applications of Definition 4.12 as many times as possible.

There may be more than one maximal terms in a message, for example, the message 4 in a protocol:

$$\text{Message 4} \quad A \to B : \{A, k_{AB}, T_B\}_{K_{BS}}, \{N_B\}_{k_{AB}}$$

Both $\{A, k_{AB}, T_B\}_{K_{BS}}$ and $\{N_B\}_{k_{AB}}$ are the maximal term of the message.

If the maximal term of a message in a protocol run is the same as that of another message of this protocol run, then we say this protocol has similar term in these two messages.

That is, one single message's maximal term should not be the same as another single message's maximal term in a particular protocol run. Otherwise, one of these two messages may be interleaving replayed in some cases. Recall the similar term in Message 4 and Message 5 in the Clark-Jacob attack on the Woo-Lam protocol of Example 3.28.

A signed term is a binary group $(\delta, \hat{m})$, where δ is a sign, $\hat{m} \in \hat{M}$. A signed term is stated as $+\hat{m}$ or $-\hat{m}$. $+\hat{m}$ and $-\hat{m}$ states a sent out or received fresh message.

Example 4.1 Figure 4.1 illustrates an example of terms. The protocol intends to establish a new session key k_{AB} between A and B, with the help of the trusted server S. N_A, N_B are nonces generated by A and B respectively; K_{AS} and K_{BS} are shared long-term keys between A and S, and B and S respectively.

$$\begin{aligned}
&\text{Message 1} \quad A \to S : A, B, N_A \\
&\text{Message 2} \quad S \to A : \{N_A, k_{AB}, B, \{k_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
&\text{Message 3} \quad A \to B : S, \{k_{AB}, A\}_{K_{BS}} \\
&\text{Message 4} \quad B \to A : \{N_B\}_{k_{AB}} \\
&\text{Message 5} \quad A \to B : \{N_B - 1\}_{k_{AB}}
\end{aligned}$$

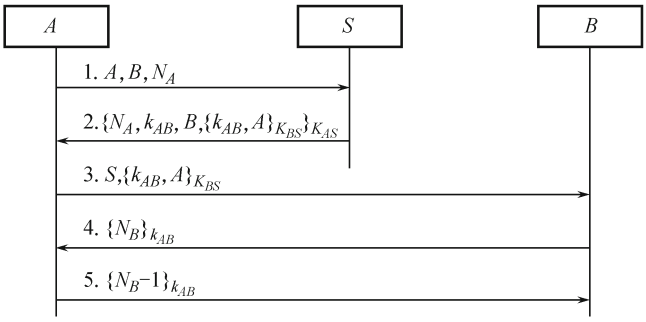


Fig. 4.1 Example of terms.

Suppose the principal B believes that N_B is a trusted freshness identifier (N_B is generated by B in this protocol run). Suppose the principal A believes

that N_A is a trusted freshness identifier (N_A is generated by A in this protocol run).

$$\text{Message 1} \quad A \rightarrow S : A, B, N_A$$

Upon sending Message 1, A has gotten the terms as in Table 4.1, where $\{...N_A...\}$ is the maximal term of Message 1.

Upon receiving Message 1, B has not gotten any terms since there does not exist a trusted freshness identifier for B.

$$\text{Message 2} \quad S \rightarrow A : \{N_A, k_{AB}, B, \{k_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$

Table 4.1 The terms owned by A from Message 1

No.	Term	Signed Term
1	$\{...N_A...\}$	$+\{...N_A...\}$
Maximal	Term	
	$\{...N_A...\}$	

Upon sending Message 2, S has gotten the terms as in Table 4.2. Upon receiving Message 2, A has gotten the terms as in Table 4.3.

Table 4.2 The terms owned by S from Message 2

No.	Term	Signed Term
1	$\{...k_{AB}...\}$	$+\{...k_{AB}...\}$
2	$\{...k_{AB}, A...\}$	$+\{...k_{AB}, A...\}$
3	$\{...k_{AB}...\}_{K_{BS}}$	$+\{...k_{AB}...\}_{K_{BS}}$
4	$\{...k_{AB}, A...\}_{K_{BS}}$	$+\{...k_{AB}, A...\}_{K_{BS}}$
5	$\{...N_A, k_{AB}...\}$	$+\{...N_A, k_{AB}...\}$
6	$\{...k_{AB}, B...\}$	$+\{...k_{AB}, B...\}$
7	$\{...k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$+\{...k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}$
8	$\{...N_A, k_{AB}, B...\}$	$+\{...N_A, k_{AB}, B...\}$
9	$\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$+\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}$
10	$\{...k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$+\{...k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}$
11	$\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$+\{...N_A, k_{AB}, B, \{...N_A, k_{AB}, A...\}_{K_{BS}}...\}$
12	$\{...N_A, k_{AB}...\}_{K_{AS}}$	$+\{...N_A, k_{AB}...\}_{K_{AS}}$
13	$\{...k_{AB}, B...\}_{K_{AS}}$	$+\{...k_{AB}, B...\}_{K_{AS}}$
14	$\{...k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$+\{...k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
15	$\{...N_A, k_{AB}, B...\}_{K_{AS}}$	$+\{...N_A, k_{AB}, B...\}_{K_{AS}}$
16	$\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$+\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
17	$\{...k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$+\{...k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
18	$\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$+\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
Maximal	Term	
	$\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	

Table 4.3 The terms owned by A from Message 2

No.	Term	Signed Term
1	$\{...N_A...\}$	$-\{...N_A...\}$
2	$\{...N_A, k_{AB}...\}$	$-\{...N_A, k_{AB}...\}$
3	$\{...N_A, B...\}$	$-\{...N_A, B...\}$
4	$\{...N_A, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$-\{...N_A, \{...k_{AB}, A...\}_{K_{BS}}...\}$
5	$\{...N_A, k_{AB}, B...\}$	$-\{...N_A, k_{AB}, B...\}$
6	$\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$-\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}$
7	$\{...N_A, B, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$-\{...N_A, B, \{...k_{AB}, A...\}_{K_{BS}}...\}$
8	$\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}$	$-\{...N_A, k_{AB}, B, \{...N_A, k_{AB}, A...\}_{K_{BS}}...\}$
9	$\{...N_A, k_{AB}...\}_{K_{AS}}$	$-\{...N_A, k_{AB}...\}_{K_{AS}}$
10	$\{...N_A, B...\}_{K_{AS}}$	$-\{...N_A, B...\}_{K_{AS}}$
11	$\{...N_A, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$-\{...N_A, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
12	$\{...N_A, k_{AB}, B...\}_{K_{AS}}$	$-\{...N_A, k_{AB}, B...\}_{K_{AS}}$
13	$\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$-\{...N_A, k_{AB}, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
14	$\{...N_A, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$-\{...N_A, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
15	$\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	$-\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$
Maximal	Term	
	$\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$	

Note that from the point of view of A, $\{...N_A, k_{AB}, A...\}_{K_{BS}}$ is the same as a randomly chosen nonce since A does not have possession of the long-term key K_{BS}. Here, $\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$ is the maximal term of Message 2.

As we will show in the following part of this chapter, upon receiving Message 2, A has the belief that k_{AB} is a trusted freshness from a trapdoor one-way transformation $\{...N_A, k_{AB}, B, \{...k_{AB}, A...\}_{K_{BS}}...\}_{K_{AS}}$ including the trusted freshness identifier N_A.

$$\text{Message 3} \quad A \rightarrow B : S, \{k_{AB}, A\}_{K_{BS}}$$

Upon sending Message 3, A has not gotten any terms without the knowledge of the long-term key K_{BS}. Upon receiving Message 3, B has not gotten any terms since there does not exist a trusted freshness identifier for B. So, the maximal term in Message 3 is null.

$$\text{Message 4} \quad B \rightarrow A : \{N_B\}_{k_{AB}}$$

Upon sending Message 4, B has gotten the terms as in Table 4.4.

Table 4.4 The terms owned by B from Message 4

No.	Term	Signed Term
1	$\{...N_B...\}$	$+\{...N_B...\}$
2	$\{...N_B...\}_{k_{AB}}$	$+\{...N_B...\}_{k_{AB}}$
Maximal	Term	
	$\{...N_B...\}_{k_{AB}}$	

Upon receiving Message 4, note that k_{AB} is a trusted freshness, so A has gotten the terms as in Table 4.5.

Table 4.5 The terms owned by A from Message 4

No.	Term	Signed Term
1	$\{...N_B...\}_{k_{AB}}$	$-\{...N_B...\}_{k_{AB}}$
Maximal	Term $\{...N_B...\}_{k_{AB}}$	

So, the maximal term of Message 4 is $\{...N_B...\}_{k_{AB}}$.

$$\text{Message 5} \quad A \rightarrow B : \{N_B - 1\}_{k_{AB}}$$

Upon sending Message 5, note that k_{AB} is a trusted freshness, so A has gotten the terms as in Table 4.6.

Table 4.6 The terms owned by A from Message 5

No.	Term	Signed Term
1	$\{...N_B...\}_{k_{AB}}$	$+\{...N_B...\}_{k_{AB}}$
Maximal	Term $\{...N_B...\}_{k_{AB}}$	

Upon receiving Message 5, B has gotten the terms as in Table 4.7.

Table 4.7 The terms owned by B from Message 5

No.	Term	Signed Term
1	$\{...N_B...\}$	$+\{...N_B...\}$
2	$\{...N_B...\}_{k_{AB}}$	$+\{...N_B...\}_{k_{AB}}$
Maximal	Term $\{...N_B...\}_{k_{AB}}$	

Here, $\{...N_B...\}_{k_{AB}}$ is the maximal term in Message 5.

Example 4.2 Here is an example of similar term. Recall the Example 4.16, the maximal term $\{...N_B...\}_{k_{AB}}$ in Message 4 is the same as that in Message 5, so we say this protocol in Example 4.16 has similar term $\{...N_B...\}_{k_{AB}}$ in Message 4 and Message 5.

Definition 4.11 (Liveness of a principal) From the point of view of a participant in a protocol run, the intended opposite participant is in lively correspondence with him in this session.

Note From the point of view of a participant in a protocol run, the intended opposite participant is specially in lively correspondence with this origin participant in this session. The liveness of a principal with origin is also called origin liveness of a principal.

In lively correspondence with a origin participant means that the origin participant has corroborative evidence that the intended opposite participant is in lively correspondence with this origin participant but not any other participant.

Definition 4.12 (Confidentiality of a freshness identifier From the point of view of a participant in a protocol run, the freshness identifier is transmitted in the form of an encryption that cannot be decrypted by the attacker.

If the freshness identifier is transmitted in the form of a plaintext or an encryption that may be decrypted by the attacker, then the freshness identifier is open. Note that the signature of a freshness identifier is not confidential.

Definition 4.13 (Freshness of a freshness identifier) From the point of view of a participant in a protocol run, the freshness identifier is new generated for this particular protocol run, not an old one or a compromised one.

A principal believes the freshness of the freshness identifier generated by the principal itself.

Definition 4.14 (Association of a freshness identifier) From the point of view of a participant in a protocol run, the freshness identifier is bound to some legitimate participants of this particular protocol run.

In the security analysis of a cryptographic protocol based on trusted freshness, the security properties are described by beliefs, which are the beliefs about the security of a cryptographic protocol owned by each participant in a particular protocol run. The beliefs are about liveness of principal, confidentiality of a freshness identifier, freshness of a freshness identifier and association of a freshness identifier.

4.2.2 Freshness principle

Definition 4.15 (Freshness Principle) For each participant of a cryptographic protocol, the security of the protocol depends only on the sent or received "loose" one-way transformation of a message which includes a trusted freshness.

The security goals are the security objects at the end of the protocol run. The set of security goals constructs the security properties of a cryptographic protocol to achieve.

In practice, a one-way transformation $[M]_k$ can be realized by a pair $(M,prf_k(M))$ where prf_k denotes a keyed pseudorandom function (e.g., a message authentication code in cipher-block-chaining mode of operation, CBC-MAC, or a keyed cryptographic hash function, HMAC) for the case of symmetric technique realization, or a digital signature algorithm for the case of

asymmetric technique realization. These are practically efficient realization[6]. These practical one-way transformations are indeed trapped door one-way transformations for most cases, hence we use "loose" one-way transformation to refer to them. As we have stated in the last chapter, when we refer to "one-way transformation", we usually mean "loose one-way transformation".

Let A, B be the participants of an authentication protocol, we have the following lemmas:

Lemma 4.1 (Liveness Lemma) The liveness of a principal B can be achieved by a participant A via a sent or received "loose" one-way transformation that includes a trusted freshness identifier owned by A, where the "loose" one-way transformation can only be accomplished by the principal B.

Lemma 4.2 (Confidentiality Lemma) The confidentiality of a freshness identifier can be achieved by a participant A if the identifier is transmitted in the form of an encryption that cannot be decrypted by the attacker; if the freshness identifier is transmitted in the form of a plaintext or an encryption that may be decrypted by the attacker, then the freshness identifier is open.

Lemma 4.3 (Freshness Lemma) The freshness of a freshness identifier can be achieved by a participant A via a sent or received "loose" one-way transformation that includes:

1) a new freshness identifier which is generated by the principal A itself;

2) a new freshness identifier which is bound together with A's another trusted freshness identifier in a "loose" one-way transformation, where the "loose" one-way transformation can only be accomplished by the intended participant B.

Lemma 4.4 (Association Lemma) The association of a freshness identifier can be achieved by a participant A via a sent or received "loose" one-way transformation that includes a trusted freshness identifier owned by A, where the "loose" one-way transformation can only be accomplished by the intended principal B, or the identity of the participant is explicitly stated in the "loose" one-way transformation.

Lemma 4.5 (Origin liveness Lemma) The liveness of a principal B with origin can be achieved by a participant A via a sent or received "loose" one-way transformation that includes a trusted freshness identifier owned by A, where the "loose" one-way transformation can only be accomplished by the principal B specially for the origin participant A.

In security analysis approach based on trusted freshness, only "loose" one-way transformation that includes a trusted freshness identifier is considered as an efficient message of a conversation, so terms could be deduced from the transformation. That is to say, only the fresh messages are concerned in security analysis based on trusted freshness, and the message parts that do not contribute to the protocol security property analysis in the trusted

freshness method are omitted.

4.2.3 Security of authentication protocol

Suppose there exists a protocol Π between A and B, the security goal of Π is to authenticate the liveness of a principal entity or establish a new session key to build a secure channel in an insecure network. The new session key k_{ab} can either be generated by any of the authenticated participants or a trusted third party S, or be the output of a function of all protocol participants' random input like N_A and N_B.

Table 4.8 lists the security property requirements to guarantee the security goals of a cryptographic protocol, and we will show that the listed security

Table 4.8 The guarantee of the security adequacy of a cryptographic protocol

Security Goals	Security Properties Achieved by A					Security Properties Achieved by B				
	B	S	N_A	N_B	k_{AB}	A	S	N_A	N_B	k_{AB}
UA-secure (Authenticate B)	1[a]									
UA-secure (Authenticate A)						1				
MA-secure (Authenticate both)	1					1				
Origin UA-secure (Origin Authenticate B)	A1[b]									
Origin UA-secure (Origin Authenticate A)						B1				
Origin MA-secure (Origin Authenticate both)	A1					B1				
UK-secure (Authenticate B)	1				1[c]1[d]AB[e]					
UK-secure (Authenticate A)						1				11AB
MK-secure (Key Transport)	1				11AB	1				11AB
MK-secure (Key Transport)	1		11AB	11AB		1		11AB	11AB	

[a] "1" or "" means that the liveness of the principal is authenticated or unknown respectively.

[b] "A1" or "" means that the liveness of the principal is authenticated with origin or unknown respectively.

[c] "?" or "" means that the confidentiality of the freshness identifier is unknown; "1" means that the confidentiality of the freshness identifier is confidential; "0" means that the freshness identifier is open, which is a plaintext, or a compromised one or an old one.

[d] "?" or "" means that the freshness of the freshness identifier is unknown; "1" means that the freshness identifier is fresh.

[e] "#" or "" means that the freshness identifier is not associated with any principals; "11A" (or "11B") means that the freshness identifier is associated with the principal A (or B); "11AB" or "11BA" means that the freshness identifier is associated with both A and B.

goals are not only necessary but also substantial. Here, when an authentication protocol is MK-secure is proved, and other similar proofs of UA-secure, UK-secure, MK-secure are omitted for interest of concision.

The security property requirements of some cryptographic protocols are given in [26]. And they are:

Theorem 4.1 An authentication-only protocol Π is called UA-Secure if and only if a participant like A believes the liveness of the intended opposite principal B.

Theorem 4.2 An authentication-only protocol Π is called MA-Secure if and only if each participant believes the liveness of both communication principals.

Theorem 4.3 A key establishment authentication protocol Π is called UK-Secure if and only if a participant like A believes the liveness of the intended opposite principal B, and believes the confidentiality, the freshness and also the association of the new session key k with the principal A and the principal B.

Theorem 4.4 A key establishment authentication protocol Π is called MK-Secure if and only if each participant like A (or B) believes the liveness of the intended opposite principal B (or A), and believes the confidentiality, the freshness and also the association of the new session key k with the principal A and the principal B.

Theorem 4.5 A authentication-only protocol Π is called Origin UA-Secure if and only if a participant like A believes the liveness of the intended opposite principal B with origin.

Theorem 4.6 A authentication-only protocol Π is called Origin MA-Secure if and only if each participant believes the liveness of both communication principals with origin.

Here, the liveness of the principal is able to determine the true identity of the other(s) which could possibly gain access to the resulting key; the confidentiality of the new session key k implies secrecy of the key; the freshness of the key k requires that the key is new for this protocol run and it could not be a replay of a compromised one; the association of the k implies preclusion of any unauthorized additional parties from deducing the same key.

Note that the liveness of the intended opposite principal differs subtly, but in a very important manner, from the association of the new session key k. The liveness of a principal implies entity authentication, and an actual communication has been established with such party (or parties); the association of the new session key k is knowledge of the identity of parties which may gain access to the key, rather than corroboration of the entity authentication.

4.2.4 Manual analysis based on trusted freshness

Based on the freshness principle and the accurately presented security goals, an analysis method as shown in Fig. 4.2 based on trusted freshness will be presented, which can be accomplished easily even by hand.

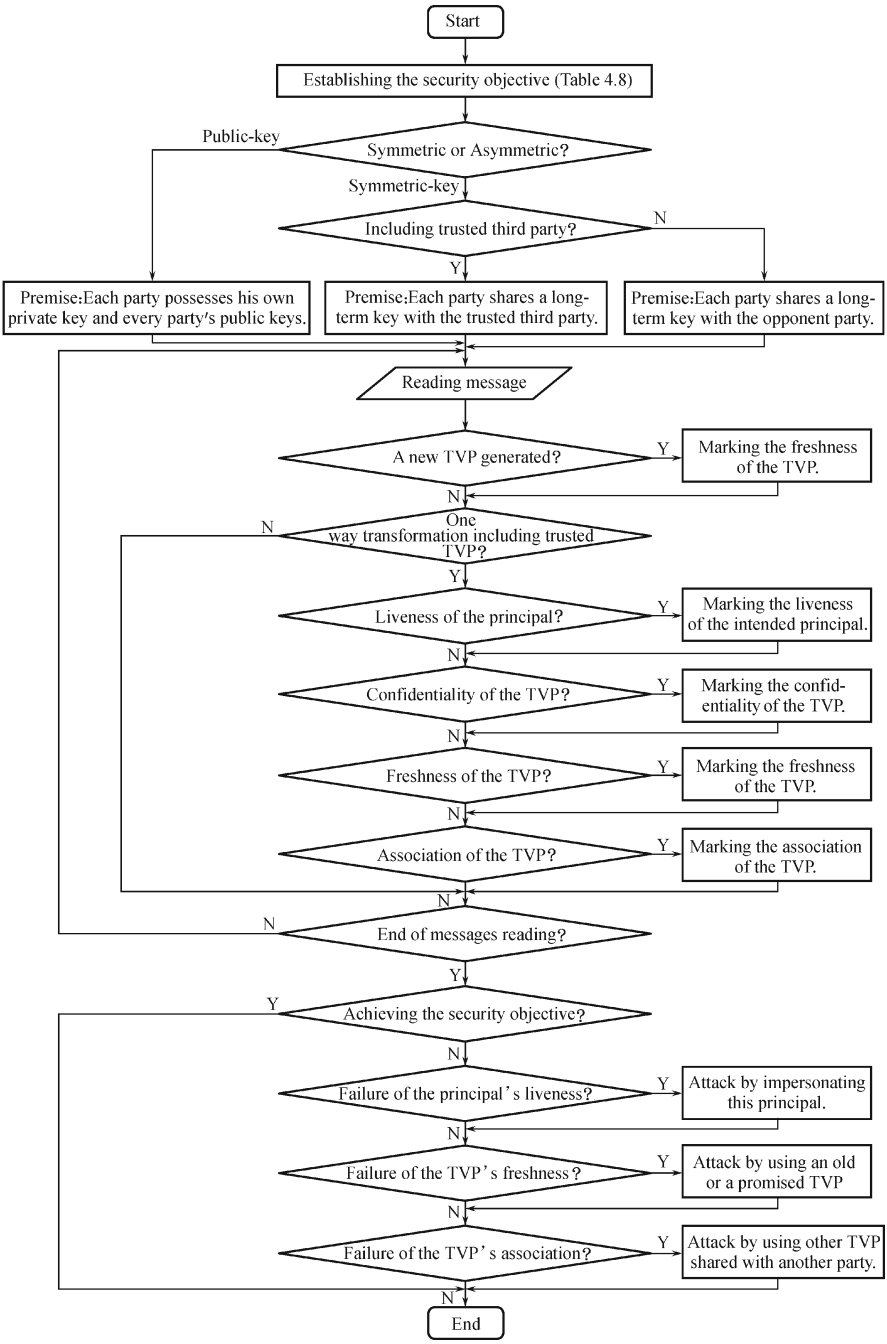


Fig. 4.2 Security analysis of cryptographic protocol based on trusted freshness.

The manual freshness analysis method is refined as follows for the same authentication protocol Π in Section 4.2.3.

1) The security goals to be reached is specified based on Table 4.8.

2) Table 4.9 specifies the premise before the start of the protocol.

Recall that each participant has his own private key and all other parties' public-keys (respectively, the shared long-term key between co-operative principals or trusted third parties) in public-key case (respectively, in symmetric-key case).

Table 4.9 The premise of a cryptographic protocol

Cryptographic schemes	Premise knowledge of A	Premise knowledge of B	Premise knowledge of I
public-key	K_A^{-1}, K_A, K_B, K_I	K_B^{-1}, K_B, K_A, K_I	K_I^{-1}, K_A, K_B, K_I
symmetric-key without TTP	K_{AB}	K_{AB}	
symmetric-key with TTP	K_{AS}	K_{BS}	

3) From the point of view of each legitimate participant, the security properties of a cryptographic protocol are established based on the freshness principle and Lemmas 4.1, 4.2, 4.3, and 4.4 while sending or receiving a "loose" one-way transformation that includes a trusted freshness.

4) Comparing with the security goals established in step 1, the analysis results can either establish the correctness of the protocol when it is in fact correct, or identify the absence of the security properties and the structure to construct attacks based on the absence. From the absence of the security properties of an authentication protocol, various attacks could be directly constructed:

(1) Absence of the liveness of a principal like A: impersonate A to launch an attack, e.g., Otway-Rees protocol[27], Woo-Lam protocol[5, 10].

Absence of the origin liveness of a principal like A: impersonate A by replaying the corroborative evidence from A which may be generated for any other participants but the original participant B.

(2) Absence of the freshness of a freshness identifier: launch an attack by replaying the recorded one-way transformation with a compromised session key, e.g., Needham-Schroeder shared key protocol[9].

(3) Absence of the association of a freshness identifier: launch an attack by confusing a legitimate principal like B to believe a session key k' between I and A (or B) to be the key between A and B, e.g., the Needham-Schroeder public-key protocol[9].

4.2.5 Application of security analysis based on trusted freshness

Example 4.3 Recall the analysis of Needham-Schroeder public-key protocol in Example 3.16 and Example 3.20, Fig. 4.3 illustrates the analysis pro-

cedure of the Needham-Schroeder public-key protocol based on the trusted freshness analysis method.

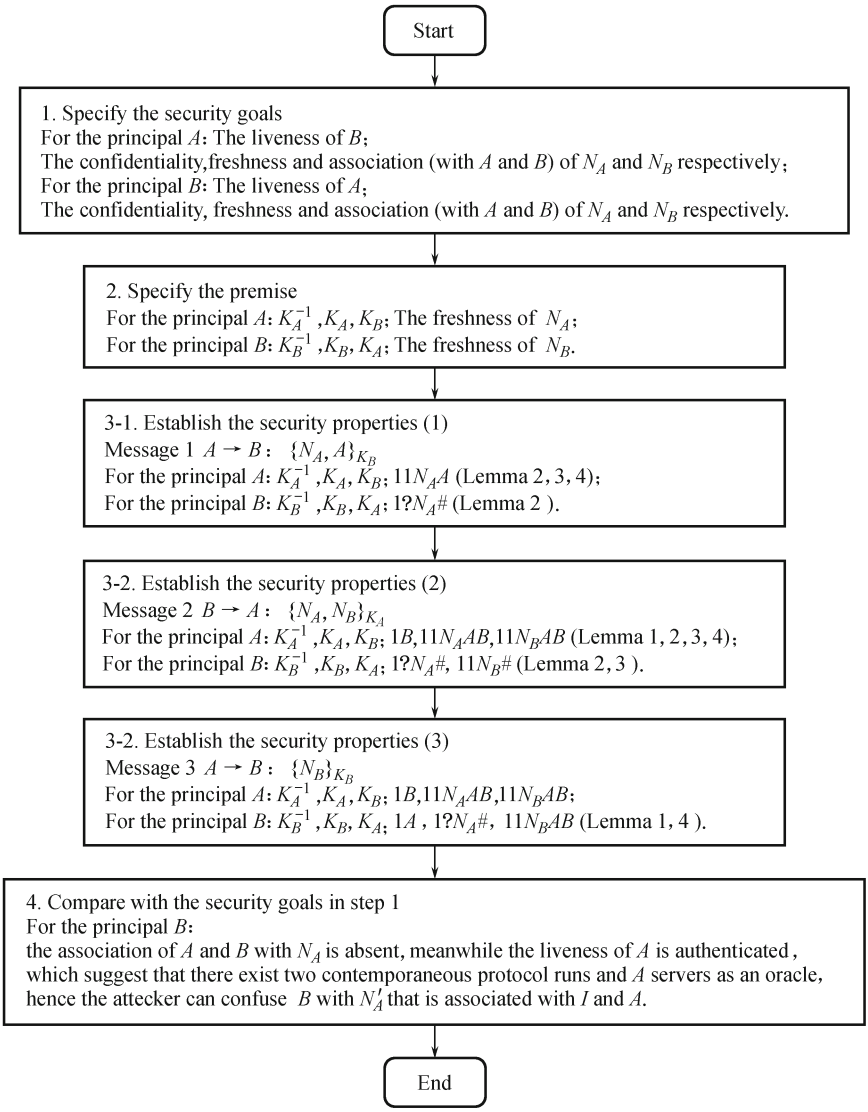


Fig. 4.3 Security analysis of Needham-Schroeder public-key protocol based on trusted freshness.

By analyzing, we get result on security properties, as shown in Tables 4.10, 4.11, and 4.12.

Table 4.10 Security properties achieved by A in the Needham-Schroeder public-key protocol

	Presence of B	N_A			N_B		
		Confidentiality	*Freshness*	*Association*	*Conf.*	*Fresh.*	*Asso.*
Message 1		1	1	A			
Message 2	1			AB	1	1	AB
Message 3							
End of run	1	11AB			11AB		

Table 4.11 Security properties achieved by B in the Needham-Schroeder public-key protocol

	Presence of A	N_A			N_B		
		Confidentiality	*Freshness*	*Association*	*Conf.*	*Fresh.*	*Asso.*
Message 1		1	?	#			
Message 2					1	1	#
Message 3	1						AB
End of run	1	1?#			11AB		

For the sake of ease, the security properties achieved by A and B are simplified as in Table 4.12.

Table 4.12 Security analysis of the Needham-Schroeder public key protocol

	A			B		
	B	N_A	N_B	A	N_A	N_B
Message 1		11A			1?#	
Message 2	1	11AB	11AB			11#
Message 3				1		11AB
End of run	1	11AB	11AB	1	1?#	11AB

4.3 Analysis of classic attacks

A successful attack on an authentication or key establishment protocol usually does not refer to breaking a cryptographic algorithm, e.g., via a complexity theory-based cryptanalysis technique. Instead, it usually refers to Malice's unauthorized and undetected acquisition of a cryptographic credential or nullification of a cryptographic service without breaking a cryptographic algorithm. It actually does not require very sophisticated techniques for an adversary to mount these attacks on a lower-layer communication protocol, as we have seen in Example 1.3.

Over several years, many different types of attacks on cryptographic primitives and protocols have been identified, and it is impossible for us to know all the protocol attacking techniques an adversary may use since the adversary will constantly devise new techniques. In this section, several typical attacks on cryptographic protocols will be analyzed, and the reasons why

these protocols are flawed will be discussed based on trusted freshness to provide us with insight into how to develop stronger protocols. Notice that an adversary may actually launch attacks via a combined way of the listed well-known protocol attacking techniques[22].

In the specific examples below, A and B are the legitimate parties Alice and Bob, and I is the adversary Malice who could also be a legitimate participant in some cases.

4.3.1 Man in the middle attack

In man-in-the-middle attack (often abbreviated to MITM), the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection while in fact the entire conversation is controlled by the attacker. A man-in-the-middle attack can only be successful when the attacker can impersonate each entity to the satisfaction of the other. Most cryptographic protocols include some form of entity authentication specifically to prevent MITM attacks. In essence, man-in-the-middle attack is generally applicable to a communication protocol where mutual entity authentication is absent.

Example 4.4 Diffie-Hellman key agreement[28] provides the first practical solution to the key distribution problem, allowing two parties, never having met in advance or having shared keying material, to establish a shared secret by exchanging messages over an insecure network, as shown in Fig. 4.4. The security rests on the intractability of the Diffie-Hellman problem and the related problem of computing discrete logarithms.

$$\begin{aligned} &\text{Message 1} \quad A \rightarrow B : \alpha^x \bmod p \\ &\text{Message 2} \quad B \rightarrow A : \alpha^y \bmod p \end{aligned}$$

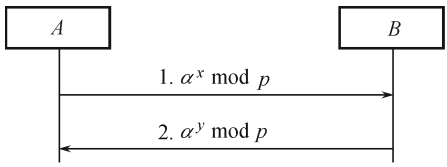


Fig. 4.4 The basic version of Diffie-Hellman key agreement protocol.

Notation

A and B are two protocol principals, x and y are randomly chosen as their private keys by A and B, respectively.

Premise

An appropriate large prime p and a generator element α of $\mathbb{Z}_p^*$ ($2 \leqslant \alpha \leqslant p-2$) are selected and published. x and y are randomly chosen hence could not be found out.

Protocol actions

1) In Message 1, A randomly chooses a secret "x, $1 \leqslant x \leqslant p-2$", and sends B message "$\alpha^x \bmod p$".

2) Upon receiving Message 1, B gets "α^x" and computes the shared key as "$k = (\alpha^x)^y \bmod p = \alpha^{xy} \bmod p$".

3) In Message 2, B randomly chooses a secret "y, $1 \leqslant y \leqslant p-2$", and sends A message "$\alpha^y \bmod p$".

4) Upon receiving Message 2, A gets "α^y" and computes the shared key as "$k = (\alpha^y)^x \bmod p = \alpha^{yx} \bmod p$".

Note that "$\alpha^{xy} \bmod p = \alpha^{yx} \bmod p$", hence A and B have computed the same key k. This is how the Diffie-Hellman key exchange protocol achieves a shared key between two communication parties.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and freshness assurances of the TVP x; from Lemma 4.2, B has the confidentiality assurance of the TVP x, but B could not deduce the freshness of the TVP x.

2) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and freshness assurances of the TVP y; from Lemma 4.2, A has the confidentiality assurance of the TVP y, but A could not deduce the freshness of the TVP y.

3) At the end of the protocol run, A and B have computed the same key "$k = \alpha^{xy} \bmod p$", and have gotten the confidentiality and freshness assurances of k from the confidentiality and freshness of the TVP x and the TVP y respectively. However, the whole transmitted messages in this protocol could not provide the assurance of the association of A with k for B, and B with k for A.

The analyzing result is indicated in Table 4.13.

Table 4.13 Security analysis of the Diffie-Hellman key agreement protocol

	A				B			
	B	x	y	k	A	x	y	k
Message 1		11#				1?#		
Message 2			1?#				11#	
End of run		11#	1?#	11A		1?#	11#	11B

This basic Diffie-Hellman protocol version provides none authentication, entity authentication and key confirmation, hence it can only provide secrecy protection of the resulting key from eavesdroppers, but not from active

adversaries. Fig. 4.5 illustrates the "man-in-the-middle" attack on the basic unauthenticated Diffie-Hellman key establishment protocol.

$$
\begin{array}{ll}
\text{Message 1} & A \rightarrow I(B) : \alpha^x \bmod p \\
\text{Message 1}' & I(A) \rightarrow B : \alpha^{x'} \bmod p \\
\text{Message 2}' & B \rightarrow I(A) : \alpha^y \bmod p \\
\text{Message 2} & I(B) \rightarrow A : \alpha^{y'} \bmod p
\end{array}
$$

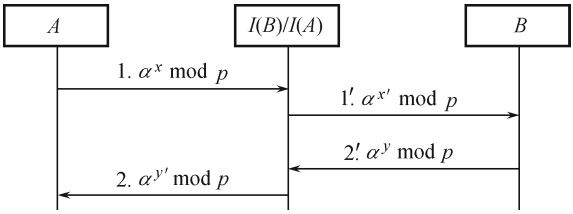


Fig. 4.5 The attack on the basic Diffie-Hellman key agreement.

Protocol actions

1) In Message 1, A randomly chooses a secret "x, $1 \leqslant x \leqslant p-2$", and sends B message "$\alpha^x \bmod p$".

2) The adversary I intercepts A's exponential α^x and replaces it with $\alpha^{x'}$ where x' is a secret chosen by I; meanwhile I intercepts B's exponential α^y and replaces it with $\alpha^{y'}$ where y' is a secret chosen by I.

3) At the end of protocol run, A forms session key "$k_1 = (\alpha^{y'})^x \bmod p = \alpha^{y'x} \bmod p = \alpha^{xy'} \bmod p$", and B forms session key "$k_2 = (\alpha^{x'})^y \bmod p = \alpha^{x'y} \bmod p$", while I can compute both keys k_1 and k_2.

4) When A subsequently sends a message to B encrypted under k_1, I decrypts it, re-encrypts the plaintext under k_2, and forwards it to B. Similarly, I decrypts the message encrypted by B (for A) under k_2, and re-encrypts it under k_1. Both A and B believe that they communicate securely, while I can read all traffic.

4.3.2 Source-substitution attack

In source-substitution attack, the attacker makes a substitution of a source entity identity with the identity of the adversary, making the victim believe that it is talking directly to the intended entity while in fact the entire conversation is controlled by the adversary. e.g., in Example 4.5, the adversary registers source entity's public-key as its own. A source-substitution attack can only be successful when the adversary can impersonate an entity to the satisfaction of the other. In essence, source-substitution attack is generally applicable to a communication protocol where entity authentication is absent.

Example 4.5 The ElGamal key agreement is a Diffie-Hellman variant providing a one-pass protocol with unilateral key authentication[29]. This protocol is more simply Diffie-Hellman key agreement wherein the public exponential of the recipient is fixed and has verifiable authenticity, as shown in Fig. 4.6.

Message 1 $A \rightarrow B : \alpha^x \bmod p$

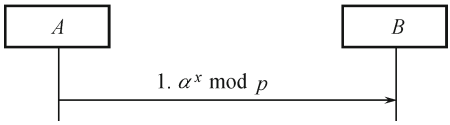


Fig. 4.6 ElGamal key agreement in one-pass.

Notation

A and B are two protocol principals. x is randomly chosen as its private key by A. b is a preselected secret random integer.

Premise

An appropriate large prime p and a generator element α of $\mathbb{Z}_p^*$ ($2 \leqslant \alpha \leqslant p-2$) are selected and published. The public-key of the recipient is known to the originator.

Protocol actions

1) One-time setup (public-key generation and publication). B picks an appropriate large prime p and a generator element "α of $\mathbb{Z}_p^*$", selects a random integer "b, $2 \leqslant b \leqslant p-2$", and computes "$\alpha^b \bmod p$", then B publishes p, α and α^b, keeping private key b secret.

2) Each time a shared key is required.

(1) A obtains an authentic copy of B's public-key α^b. A randomly chooses a secret "$x, 1 \leqslant x \leqslant p-2$", and sends B message "$\alpha^x \bmod p$".

(2) A computes the shared key as "$k = (\alpha^b)^x \bmod p = \alpha^{bx} \bmod p = \alpha^{xb} \bmod p$"; B receives α^x and computes the same shared key as "$k = (\alpha^x)^b \bmod p = \alpha^{xb} \bmod p$".

Protocol security analysis

1) Before the protocol run, suppose A and B have the assurance that b is a confidential and long-term key which is known only by B.

2) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and freshness assurances of the TVP x; from Lemma 4.2, B has the confidentiality assurance of the TVP x, but B could not deduce the freshness of the TVP x.

3) At the end of the protocol run, A has computed the key $k = \alpha^{xb} \bmod p$. From Lemma 4.2, Lemma 4.3 and Lemma 4.4, A has gotten the confidentiality and freshness assurances of k from the confidentiality and freshness of the

TVP x, and A has gotten the association of A and B with k from the TVP x and B's long-term key b respectively. From Lemma 4.2, B has the confidentiality and the association (B with k) assurances, from B's long-term private key b, but B could not deduce the freshness of k and also the association assurance of A with k.

The analyzing result is indicated in Table 4.14. The recipient B in this protocol has neither corroboration that it shares the secret key k, nor any key freshness assurance. Neither party obtains entity authentication or key confirmation. The adversary can launch an attack by impersonating A directly.

Table 4.14 Security analysis of the ElGamal key agreement in one-pass

	A				B			
	B	x	b	k	A	x	b	k
Message 1		11#	11B			1?#	11B	
End of run		11#	11B	11AB		1?#	11B	1?B

Example 4.6 The MTI two-pass key agreement protocol as shown in Fig. 4.7 is a variant of Diffie-Hellman key agreement to yield time-variant session key with mutual key authentication against passive attacks[28–30].

$$\text{Message 1} \quad A \to B : \alpha^x \bmod p$$
$$\text{Message 2} \quad B \to A : \alpha^y \bmod p$$

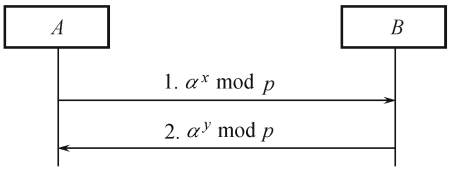


Fig. 4.7 The MTI key agreement protocol in two-pass.

Notation

A and B are two protocol principals, a ($1 \leqslant a \leqslant p-2$) and b ($1 \leqslant b \leqslant p-2$) are randomly chosen integers as its long-term private keys by A and B respectively.

Premise

An appropriate large prime p and a generator element α of "$\mathbb{Z}_p^*$ ($2 \leqslant \alpha \leqslant p-2$)" are selected and published. A has the assurance that only B knows the corresponding long-term private key b of "$\alpha^b \bmod p$", B has the assurance that only A knows the corresponding long-term private key a of "$\alpha^a \bmod p$".

Protocol actions

1) In Message 1, A randomly chooses a secret "x, $1 \leqslant x \leqslant p-2$", and sends B message "$\alpha^x \bmod p$".

2) In Message 2, B randomly chooses a secret "y, $1 \leqslant y \leqslant p-2$", and sends A message "$\alpha^y \bmod p$".

3) B receives "$\alpha^x \bmod p$" and computes the shared key as "$k = (\alpha^x)^b(\alpha^a)^y = \alpha^{bx+ay} \bmod p$".

4) A receives α^y and computes the shared key as "$k = (\alpha^y)^a(\alpha^b)^x = \alpha^{ay+bx} = \alpha^{bx+ay} \bmod p$".

Protocol security analysis

1) Before the protocol run, suppose A and B have the assurance that a is a confidential and long-term key which is known only by A, and that b is a confidential and long-term key which is known only by B.

2) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and freshness assurances of the TVP x; from Lemma 4.2, B has the confidentiality assurance of the TVP x, but B could not deduce the freshness of the TVP x.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and freshness assurances of the TVP y; from Lemma 4.2, A has the confidentiality assurance of the TVP y, but A could not deduce the freshness of the TVP y.

4) At the end of the protocol run, A has computed the key "$k = \alpha^{bx+ay} \bmod p$". From Lemma 4.2, Lemma 4.3, and Lemma 4.4, A has gotten the confidentiality and freshness assurances of k from the confidentiality and freshness of the TVP x, and A has gotten the association of A and B with k from the TVP x and B's long-term key b respectively. Similar cases exist for B. At last, neither party obtains entity authentication, hence the adversary could launch an attack with a masquerade.

From Table 4.15 we get security properties of the protocol. The freshness assurance of the session key k depends on the fresh input x or y from each party, and the association of k depends on the long-term private key a and b. Neither party obtains entity authentication or key confirmation. The adversary can launch an attack by impersonating A directly.

Table 4.15 Security analysis of the MTI key agreement in two-pass

	A				B			
	B	x	y	k	A	x	y	k
Message 1		11#				1?#		
Message 2			1?#				11#	
End of run		11#	1?#	11AB		1?#	11#	11AB

Hence, the protocol in Example 4.6 may suffer an attack[29], as shown in Fig. 4.8.

$$\begin{aligned}
&\text{Message 1} \quad A \to I(B): \quad \alpha^x \bmod p\\
&\text{Message 1}' \quad I \to B: \quad \alpha^x \bmod p\\
&\text{Message 2}' \quad B \to I: \quad \alpha^y \bmod p\\
&\text{Message 2} \quad I(B) \to A: \quad \alpha^{ey} \bmod p
\end{aligned}$$

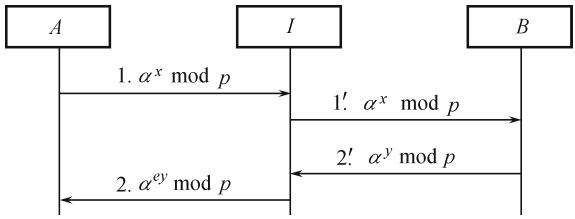


Fig. 4.8 The attack on the MTI key agreement protocol in two-pass.

Premise

A selects a random integer "a $(1 \leqslant a \leqslant p-2)$" as A's long-term private key, and registers the public-key "$\alpha^a \bmod p$"; B selects a random integer "b $(1 \leqslant b \leqslant p-2)$" as B's long-term private key, and registers the public-key "$\alpha^b \bmod p$"; the adversary I selects an integer e, computes "$\alpha^{ae} \bmod p$", and registers the public-key "$\alpha^{ae} \bmod p$".

Protocol actions

1) In Message 1, A randomly chooses a secret "x, $1 \leqslant x \leqslant p-2$", and sends B message "$\alpha^x \bmod p$".

2) In Message 1′, I launches a new protocol run between I and B by forwarding A′s exponential α^x to B.

3) B receives α^x and computes the shared key between I and B as "$k = (\alpha^x)^b(\alpha^{ae})^y = \alpha^{bx+aey} \bmod p$".

4) In Message 2′, I intercepts B's exponential α^y, then modifies "α^y to α^{ey}" and sends it to A.

5) A receives α^{ey} and computes the shared key with B as "$k = (\alpha^{ey})^a(\alpha^b)^x = \alpha^{aey+bx} \bmod p = \alpha^{bx+aey} \bmod p$".

At the end of the protocol run, A believes it is shared key "α^{bx+aey}" with B, while B believes it is shared key "α^{bx+aey}" with I. In this attack, I is not actually able to compute k itself, but rather causes B to have false beliefs. B concludes that subsequently received messages encrypted by the key "$k = \alpha^{bx+aey} \bmod p$" originated from I, whereas, in fact, it is only A who knows k and can originate such messages. This attack may be detected by key confirmation and prevented by modifying the protocol so that the exponentials or the identities of the intended entities are authenticated, e.g., through a digital signature.

4.3.3 Message replay attack

Message replay is a classic attack on authentication and authenticated key establishment protocols. A message replay attack is where a previous legitimate data transmission is captured or recorded and then replayed by an attacker in a new protocol run attempting to gain unauthorized access to data or re-

sources. A replay attack can be used in conjunction with a masquerade where an unauthorized user pretends to be somebody else.

A replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack by IP packet substitution (such as stream cipher attack).

Suppose A wants to prove A's identity to B. B requests A's password as proof of identity A dutifully provides (possibly after some transformation like a hash function); meanwhile, Malice is eavesdropping the conversation and keeps the password. After the interchange is over, Malice connects with B posing as A; when asking for a proof of identity, Malice sends A's password read from the last session which B will accept.

A way to avoid replay attacks is using session tokens: B sends a one-time token to A, which A uses to transform the password and sends the result to B (e.g., computing a hash function of the session token appended to the password). On B's side, B performs the same computation; if and only if both values match, the login is successful. Now suppose Malice has captured this value and tries to use it on another session; B sends a different session token, and when Malice replies with the captured value it will be different from B's computation. Session tokens should be chosen by a (pseudo-) random process. Otherwise Malice may be able to guess some future token and convince A to use that token in A's transformation. Malice can then replay A's reply at a later time, which B will accept.

B can also send nonces but should then include a message authentication code (MAC), which A should check in order to avoid replay attacks.

Timestamping is another way of preventing a replay attack. Synchronization should be achieved when using timestamp in a secure protocol. For example, B periodically broadcasts the time on B's clock together with a MAC. When A wants to send B a message, A includes A's best estimate of the time on A's clock in A's message, which is also authenticated. B only accepts messages for which the timestamp is within a reasonable tolerance. The advantage of this scheme is that B does not need to generate (pseudo-) random numbers.

It seems that we have already established a good awareness of message-replay attacks. This can be evidently seen from the ubiquitous use of TVPs (nonces, timestamps) in the basic and standard protocol constructions. However, simply using a timestamp on data or a message, or using tokens to verify timestamps of messages does not guarantee the key freshness. In essence, replay attack is generally applicable to a communication protocol where key freshness assurance is absent. This is why mistakes can be made repeatedly even when the designers know the errors very well in a different context. The freshness assurance could be achieved as in Lemma 4.3.

Example 4.7 Recall Needham-Schroeder shared key protocol in Example

3.17. The protocol as shown in Fig. 4.9 intends to establish a new session key k_{AB} between A and B, with the help of the trusted server S. N_A, N_B are nonces generated by A and B respectively; K_{AS} and K_{BS} are shared long-term keys between A and S, and B and S respectively.

$$
\begin{aligned}
&\text{Message 1} \quad A \rightarrow S : A, B, N_A \\
&\text{Message 2} \quad S \rightarrow A : \{N_A, k_{AB}, B, \{k_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
&\text{Message 3} \quad A \rightarrow B : S, \{k_{AB}, A\}_{K_{BS}} \\
&\text{Message 4} \quad B \rightarrow A : \{N_B\}_{k_{AB}} \\
&\text{Message 5} \quad A \rightarrow B : \{N_B - 1\}_{k_{AB}}
\end{aligned}
$$

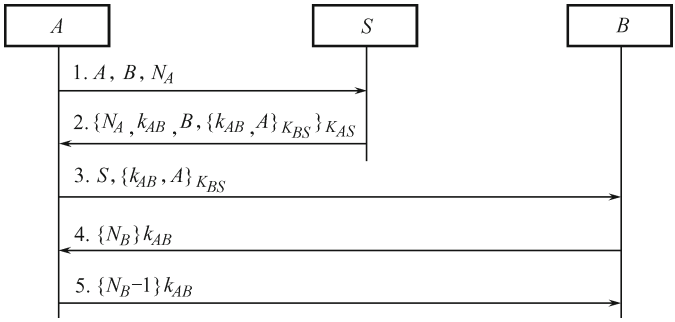


Fig. 4.9 The Needham-Schroeder shared key protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.3, A has the freshness assurance of the TVP N_A.

2) Upon receiving Message 2, A has the assurance that Message 2 including trusted freshness N_A must be encrypted by the trusted third party S, and could not be a replay one. From Lemma 4.2 and Lemma 4.3, A has the confidentiality and freshness assurances of the new chosen session key k_{AB}; from Lemma 4.4, A has the association assurance of the new session key k_{AB} with A and B since Message 2 could not be a replay one, and only S could encrypt N_A and k_{AB}. From Lemma 4.1, A has gotten the entity authentication of S.

3) Upon receiving Message 3, from Lemma 4.2, B has the assurance that the new chosen session key k_{AB} is confidential, but B does not know whether k_{AB} is a new generated key for this protocol run or a promised one, and B does not know whether k_{AB} is associated with A and B in this protocol run.

4) Upon receiving Message 4, A decrypts N'_B via using k_{AB}, but A is not sure whether N'_B is exactly the randomly chosen TVP N_B by B, or just a value decrypted using k_{AB} from a random data selected by the adversary I.

5) Upon receiving Message 5, B could not get any new assurance about the protocol security.

From Table 4.16, the absence of the security properties, various attacks, most of which are message replay attacks, could be constructed.

Table 4.16 Security analysis of the Needham-Schroeder shared key protocol

	A					B				
	B	S	N_A	N_B	k_{AB}	A	S	N_A	N_B	k_{AB}
Message 1			01#					0?#		
Message 2		1			11AB					
Message 3										1?#
Message 4				1?#					11#	
Message 5										
End of run		1	01#	1?#	11AB			0?#	11#	1?#

1. Attack on Needham-Schroeder shared key protocol by impersonating B

Example 4.8 From the absence of the B's liveness, in the point of view of A, an attack by impersonation B could be launched, as shown in Fig. 4.10.

$$\begin{aligned}
&\text{Message 1} \quad A \to S: \quad A, B, N_A \\
&\text{Message 2} \quad S \to A: \quad \{N_A, k_{AB}, B, \{k_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
&\text{Message 3} \quad A \to I(B): S, \{k_{AB}, A\}_{K_{BS}} \\
&\text{Message 4} \quad I(B) \to A: N \\
&\text{Message 5} \quad A \to I(B): \{N'_B - 1\}_{k_{AB}}, \text{ where } N'_B = \{N\}_{k_{AB}^{-1}}
\end{aligned}$$

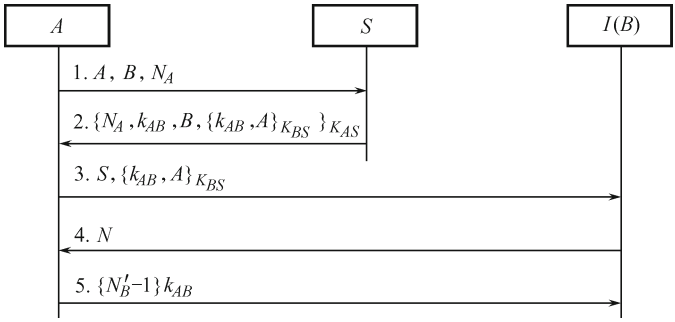


Fig. 4.10 An attack on the Needham-Schroeder shared key protocol by impersonating B.

Protocol actions

1) The message exchanges from Message 1 to Message 3 are the same as in the original Needham-Schroeder shared key protocol.

2) In Message 4, B sends a random nonce N to A as $\{N_B\}_{k_{AB}}$, then A decrypts N using the new session key k_{AB} and gets $N'_B = \{N\}_{k_{AB}^{-1}}$.

3) In Message 5, A encrypts $\{N'_B - 1\}$ using k_{AB} as a response to B's

challenge N'_B (Actually N'_B is not a challenge from B, it is only a nonce from the attacker).

Upon termination of the protocol run in Example 4.8, the adversary I is not actually able to get k_{AB} itself, but rather causes A to have false beliefs: A has completed a successful protocol run with B, and is sharing a new session key k_{AB} with B. A concludes that subsequently messages could be encrypted using k_{AB} and safely transmitted to B, whereas in fact, B knows nothing about the key establishment procedure.

2. An attack on the Needham-Schroeder shared key protocol by using compromised key

Example 4.9 From the absence of the key freshness assurance of k_{AB}, the Needham-Schroeder protocol is vulnerable to an attack discovered by Denning and Sacco[31], as illustrated in Fig. 4.11. The attacker I intercepts A's messages sent by and to A in the message lines 3, 4 and 5, and replays old session key material $S, \{k'_{AB}, A\}_{K_{BS}}$ which the attacker may record from a previous run of the protocol between A and B. An old session key k'_{AB} could possibly be promised since a careless communication principal may put it in an insecure place, or discard it etc., while the attacker has unlimited time to spend on finding an old data encryption key and then reusing it as though it were new.

$$
\begin{array}{ll}
\text{Message 1} & A \rightarrow S: \quad A, B, N_A \\
\text{Message 2} & S \rightarrow A: \quad \{N_A, k_{AB}, B, \{k_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\
\text{Message 3} & A \rightarrow B: \quad S, \{k_{AB}, A\}_{K_{BS}} \\
\text{Message 3}' & I(A) \rightarrow B: S, \{k'_{AB}, A\}_{K_{BS}} \\
\text{Message 4}' & B \rightarrow I(A): \{N_B\}_{k'_{AB}} \\
\text{Message 5}' & I(A) \rightarrow B: \{N_B - 1\}_{k'_{AB}}
\end{array}
$$

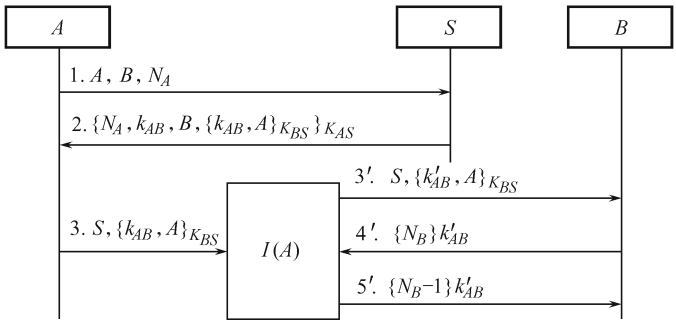


Fig. 4.11 An attack on the Needham-Schroeder shared key protocol using a compromised key k'_{AB}.

Protocol actions

1) The message exchanges from Message 1 to Message 3 are the same as in the original Needham-Schroeder shared key protocol.

2) In Message $3'$, the attacker I sends a recorded old message "$S, \{k'_{AB}, A\}_{K_{BS}}$" to B where I has the knowledge of the old session key k'_{AB}.

3) In Message $4'$, B sends A a challenge N_B to confirm the new session key.

4) In Message $5'$, A makes response $\{N_B - 1\}_{k'_{AB}}$ to B by using a compromised old session key k'_{AB}.

Upon termination of the protocol run in Example 4.9, A believes that the key establishment with B fails, whereas, B believes that he has successfully established a new session key k'_{AB} with A, and B may ignore the subsequently key establishment requirement for a new session key. Actually, k'_{AB} is a promised key known by the attacker. In deed, this attack could be launched by I from sending Message $3'$ directly, and the principal A will not participate in this protocol run at all (Example 4.10).

3. Attack on Needham-Schroeder shared key protocol by impersonating A

Example 4.10 From the absence of the A's liveness, the attacker may launch an attack without the presence of A, as shown in Fig. 4.12.

$$\begin{aligned}
&\text{Message } 3' \quad I(A) \rightarrow B : S, \{k'_{AB}, A\}_{K_{BS}} \\
&\text{Message } 4' \quad B \rightarrow I(A) : \{N_B\}_{k'_{AB}} \\
&\text{Message } 5' \quad I(A) \rightarrow B : \{N_B - 1\}_{k'_{AB}}
\end{aligned}$$

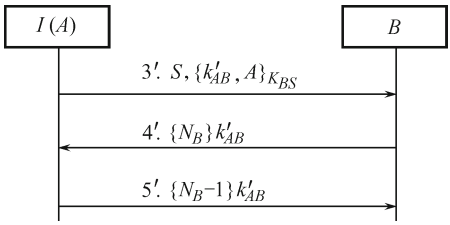


Fig. 4.12 An attack on the Needham-Schroeder shared key protocol by impersonating A.

4.3.4 Parallel session attack

A parallel session attack occurs when two or more protocol runs are executed concurrently and messages from one run (the reference session) are used to form spoofed messages in another run (the attack session). Following are examples of the parallel session attack.

Example 4.11 Figure 4.13 is a simple one-way authentication protocol. A wants to check the liveness of B by using a new chosen challenge N_A.

$$\begin{array}{ll} \text{Message 1} & A \rightarrow B : \{N_A\}_{K_{AB}} \\ \text{Message 2} & B \rightarrow A : \{N_A + 1\}_{K_{AB}} \end{array}$$

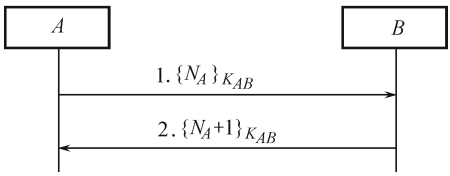


Fig. 4.13 A simple one-way authentication protocol.

Notation

A and B are two protocol principals.

Premise

K_{AB} is the shared long-term key between A and B, which is initially established by non-cryptographic, and out-of-band techniques; N_A is a nonce randomly chosen by A.

Protocol actions

1) In Message 1, A randomly chooses a new nonce N_A as a challenge for this protocol run, and sends it to B encrypted under the shared long-term key K_{AB} between A and B.

2) Upon receiving Message 1, B gets N_A from the encryption $\{N_A\}_{K_{AB}}$, and responses $\{N_A + 1\}_{K_{AB}}$ to show that B is operational.

Successful execution of the protocol should convince A that B is present since only B could have formed the appropriate response $\{N_A + 1\}_{K_{AB}}$ to the challenge N_A issued in Message 1.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of the TVP N_A.

2) Upon receiving Message 1, B could not determine whether $\{N_A\}_{K_{AB}}$ is a nonce chosen by the attacker or a challenge selected by the opponent party A.

3) Upon receiving Message 2, A could not determine whether $\{N_A+1\}_{K_{AB}}$ is an appropriate response to the challenge N_A from B or not, since there does not exist the evidence that $\{N_A + 1\}_{K_{AB}}$ is accomplished by the principal B, hence it even may be a trapped one-way transformation from A itself. Table 4.17 indicates the analyzing result.

Table 4.17 Security analysis of the simple one-way authentication protocol

	A	
	B	N_A
Message 1		11#
Message 2		
End of run		11#

Example 4.12 Figure 4.14 illustrates that an intruder can play the role of B as responder and initiator. The attack works by starting another protocol run in response to the initial challenge $\{N_A\}$.

$$\begin{aligned}
&\text{Message 1} \quad A \rightarrow I(B) : \{N_A\}_{K_{AB}} \\
&\text{Message 1}' \quad I(B) \rightarrow A : \{N_A\}_{K_{AB}} \\
&\text{Message 2}' \quad A \rightarrow I(B) : \{N_A + 1\}_{K_{AB}} \\
&\text{Message 2} \quad I(B) \rightarrow A : \{N_A + 1\}_{K_{AB}}
\end{aligned}$$

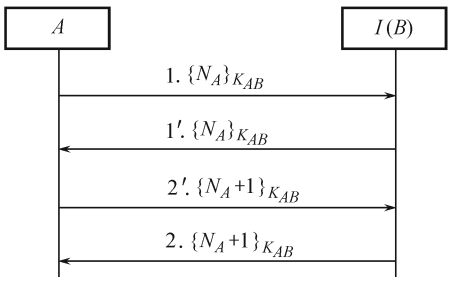


Fig. 4.14 A parallel session attack on the simple one-way authentication protocol.

Notation

A and B are two protocol principals, I is the attacker.

Premise

K_{AB} is the shared long-term key between A and B, which is initially established by non-cryptographic, and out-of-band techniques; N_A is a nonce randomly chosen by A.

Protocol actions

1) To initiate the attack, the adversary waits for A to initiate the first protocol session with B. A does the same thing as in Message 1 of Example 4.11.

2) I intercepts the Message 1 and pretends to be B, starting a second run of the protocol by replaying the intercepted message $\{N_A\}_{K_{AB}}$.

3) A replies to $I(B)$'s challenge in Message 2′ with the exact value $\{N_A + 1\}_{K_{AB}}$ that $I(B)$ requires to accurately complete the attack session.

4) I intercepts the Message $2'$ and replays the intercepted message $\{N_A + 1\}_{K_{AB}}$ to A as B's (indeed, it is I) response to Message 1.

Successful execution of the attack on this simple One-Way protocol could convince A that B is present since only B could have formed the appropriate response $\{N_A + 1\}_{K_{AB}}$ to the challenge N_A issued in Message 1.

Such attacks may be prevented via modifying the protocol so that the challenge response messages could show the identity of the party performing encryption or decryption. For example, change $\{N_A + 1\}_{K_{AB}}$ to $\{A, N_A + 1\}_{K_{AB}}$, or $\{B, N_A + 1\}_{K_{AB}}$.

Example 4.13 The Woo-Lam protocol[5] is an authentication protocol based on symmetric-key cryptography, as shown in Fig. 4.15. The protocol intends to authenticate A to B with the aid of a trusted third party S. The nonce N_B servers as challenge for authenticating A to B.

$$
\begin{array}{ll}
\text{Message 1} & A \rightarrow B : A \\
\text{Message 2} & B \rightarrow A : N_B \\
\text{Message 3} & A \rightarrow B : \{N_B\}_{K_{AS}} \\
\text{Message 4} & B \rightarrow S : \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}} \\
\text{Message 5} & S \rightarrow B : \{N_B\}_{K_{BS}}
\end{array}
$$

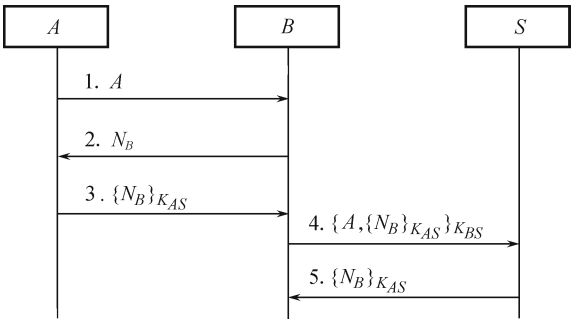


Fig. 4.15 The Woo-Lam authentication protocol.

Notation

A and B are two protocol principals, and S is the trusted third party. K_{AS} and K_{BS} are keys that A and B shared with S respectively.

Premise

K_{AS} and K_{BS} are shared long-term keys between A and S, and B and S respectively, which are initially established by non-cryptographic, and out-of-band techniques. N_B is a nonce randomly chosen by B.

Protocol actions

1) In Message 1, A launches a new protocol run.

2) In Message 2, B randomly chooses a new nonce N_B as a challenge to this protocol run and sends it to A.

3) In Message 3, A sends B a response to the challenge N_B using the shared long-term only known by A and the trusted third party S, to show that it is A who has encrypted B's challenge N_B.

4) In Message 4, B encrypts A's response with A's identity using the shared long-term key only known by B and the trusted third party S, to show that it is B who has encrypted A's response $\{N_B\}_{K_{AS}}$.

5) Upon receiving Message 4, S checks A's response $\{N_B\}_{K_{AS}}$ to confirm A's identity. Then, S sends B the message $\{N_B\}_{K_{BS}}$ showing that A's identity has been authenticated by S.

6) Upon receiving Message 5, B checks S's response $\{N_B\}_{K_{BS}}$ to get N_B. If N_B is correct, then B believes that A's identity has been authenticated by the trusted third party S.

Successful execution should convince B that A is present with the help of the trusted party S.

Protocol security analysis

1) In Message 1, neither A nor B can draw any useful assurance from it.

2) In Message 2, from Lemma 4.3, B has the freshness assurance of the TVP N_B.

3) Upon receiving Message 3, B could not determine whether $\{N_B\}_{K_{AS}}$ is a nonce chosen by the attacker or a response from the opponent party A, and B can only treat it as an unrecognizable foreign cipher chunk. Neither A nor B can draw any new assurance from Message 3.

4) Upon receiving Message 4, S could not determine whether $\{N_B\}_{K_{AS}}$ or $\{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$ is a fresh message or not since S doesn't have any trusted freshness. Actually, S could not authenticate the liveness of A and B, but S proves that N_B is recovered from an encryption under the shared long-term key between A and S.

5) Upon receiving Message 5, from Lemma 4.1, B believes that it must be S who has decrypted the message $\{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$ and $\{N_B\}_{K_{AS}}$ from recovering N_B which is trusted by B. However, B cannot authenticate the liveness of A since there does not exist any evidence that N_B is the challenge from B to A.

Upon termination of the Woo-Lam protocol, B has not gotten the assurance that A is present. The analyzing result is indicated in Table 4.18.

From the absence of the liveness property of A, an attacker can play the role of A as responder or initiator. In addition to the attacks discovered in [10] and [12] on the Woo-Lam Protocol, a new attack is illustrated in Example 3.13. In essence, a parallel session attack continually exists in the Woo-Lam Protocol, since the absence of A's liveness property has not been fixed.

Table 4.18 Security analysis of the Woo-Lam protocol

	A			B			S		
	B	S	N_B	A	S	N_B	A	B	N_B
Message 1									
Message 2			0?#			01#			0?#
Message 3									
Message 4						01#			0?#
Message 5					1				
End of run					1				0?#

4.3.5 Reflection attack

Reflection attack is a type of replay attack in which transmitted data is sent back to its originator. In a basic authentication scheme, a secret is known to both the originator and the target (or the trusted server), this allows them to be authenticated and they may verify this shared secret without sending it in plaintext over the wire. The originator initiates a connection to a target, and the target attempts to authenticate the originator by sending it a challenge, then the originator utilizes the shared secret to process this randomly chosen challenge to show his identity. The essential idea of the reflection attack is to trick the target into providing the answer to its own challenge (Example 4.11). That is, the same challenge-response protocol is used by each side to authenticate the other side, thereby leaving the attacker with fully-authenticated channel connection.

Example 4.11 could be fixed by sending the responder's identity within the response. Then, if the originator receives a response that has its own identity in it, then the originator will reject the response; if the originator receives a response that has the opponent's identity in it, and if the nonce is the same as the one the originator has sent in his challenge, then the originator will accept the message.

Example 4.14 Recall the fixed Woo-Lam protocol by Abadi and Needham[10]. This fixed version of the Woo-Lam Protocol suffers a reflection attack discovered by Clark and Jacob[12]. Here, the attacker mounts reflection attack twice: Message 3 is a reflection of Message 2, and Message 5 is that of Message 4. First, the random chunk that B receives in Message 3 is actually B's nonce sent out in Message 2. Again, the cipher chunk that B receives in Message 5 is actually the one created by himself and sent out in Message 4. B cannot detect this attack discovered by Clark and Jacob.

As Mao has stated in [22], a series of fixes for the Woo-Lam Protocol[32] are also flawed in a similar way: they all suffer reflection attack in various ways[10, 12, 22]. The key reason for this flaw is the absence of the liveness

property of A as we have illustrated in Table 4.18.

4.3.6 Interleaving attack

Interleaving attack is a type of replay attack in which transmitted data is from outside the current run of the protocol. The attacker may compose a message and sends it out to a principal in one run, from which he expects to receive an answer; the answer may be useful for another principal in another run, and in the latter run, the answer obtained from the former run may further stimulate the latter principal to answer a question which in turn is further used in the first run, and so on[22]. In essence, interleaving attack is generally applicable to a communication protocol where a principal's liveness or the session key's association assurance is absent. Here are two examples:

Example 4.15 Figure 4.16 illustrates a refined Woo-Lam protocol in [22]. The absence of the A's liveness has not been solved yet (see Table 4.18).

$$
\begin{aligned}
&\text{Message 1} \quad A \rightarrow B : A \\
&\text{Message 2} \quad B \rightarrow A : N_B \\
&\text{Message 3} \quad A \rightarrow B : \{N_B\}_{K_{AS}} \\
&\text{Message 4} \quad B \rightarrow S : \{A, N_B, \{N_B\}_{K_{AS}}\}_{K_{BS}} \\
&\text{Message 5} \quad S \rightarrow B : \{N_B\}_{K_{BS}}
\end{aligned}
$$

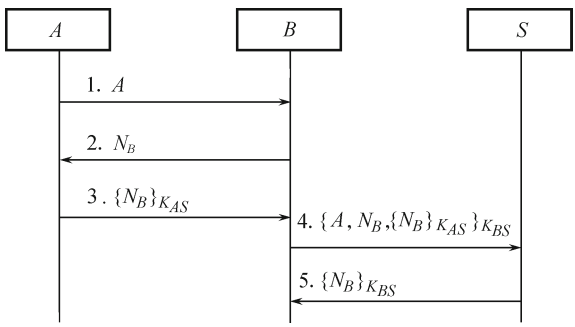


Fig. 4.16 The refined Woo-Lam authentication protocol version of Mao.

Protocol actions

1) The message exchanges from Message 1 to Message 3 are the same as in the original Woo-Lam authentication protocol.

2) In Message 4, $\{N_B\}_{K_{AS}}$ is an encryption which B could not recover, so B includes the trusted freshness N_B in Message 4 to guarantee the freshness of Message 4.

3) Upon receiving Message 4, S recovers N_B using K_{BS}, checks whether A has responded to the same challenge N_B, and then re-encrypts N_B using the long-term key K_{BS} and sends N_B back to B in Message 5.

4) Upon receiving Message 5, If S replies to $\{N_B\}_{K_{BS}}$, then B is convinced that A is active in this protocol run.

Successful execution should convince B that A is present with the help of the trusted party S.

Protocol security analysis

1) The assurances from Message 1 to Message 3 are the same as in the original Woo-Lam authentication protocol.

2) Upon receiving Message 4, since there is none trusted TVP for S, S still could not determine whether $\{N_B\}_{K_{AS}}$ or $\{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$ is a fresh message or not. Actually, S still could not authenticate the liveness of A and B.

Hence, upon termination of the Woo-Lam protocol, B has not gotten the assurance that A is present. The security properties of the Woo-Lam protocol are the same as those in Table 4.18. Hence there exists an interleaving attack on the refined Woo-Lam protocol, as shown in Fig. 4.17.

$$
\begin{array}{ll}
\text{Message } 1 & A \rightarrow I : A \\
\text{Message } 1' & I(A) \rightarrow B : A \\
\text{Message } 2' & B \rightarrow I(A) : N_B \\
\text{Message } 2 & I \rightarrow A : N_B \\
\text{Message } 3 & A \rightarrow I : \{N_B\}_{K_{AS}} \\
\text{Message } 3' & I(A) \rightarrow B : \{N_B\}_{K_{AS}} \\
\text{Message } 4' & B \rightarrow S : \{A, N_B, \{N_B\}_{K_{AS}}\}_{K_{BS}} \\
\text{Message } 5' & S \rightarrow B : \{N_B\}_{K_{BS}} \\
\text{Message } 4 & I \rightarrow S : \{I, N_B, \{N_B\}_{K_{AS}}\}_{K_{IS}} \\
\text{Message } 5 & S \rightarrow I : \{N_B\}_{K_{IS}}
\end{array}
$$

Protocol actions

1) In Message 1, A tells I that A wants to establish a connection with I; upon receiving Message 1, I establishes a connection with B instantly by impersonating A.

2) In Message $2'$, B provides challenge N_B to the session between A (indeed, it is I) and B; upon receiving Message $2'$, I provides the same challenge N_B to A for the session between A and I.

3) In Message 3, A returns this challenge N_B encrypted under K_{AS} to I; upon receiving Message 3, by impersonating A, I passes $\{N_B\}_{K_{AS}}$ to B as A's response to the challenge N_B for the session between A and B. As we have seen, in Message 3, A servers as an encryption oracle in the session between $I(A)$ and B.

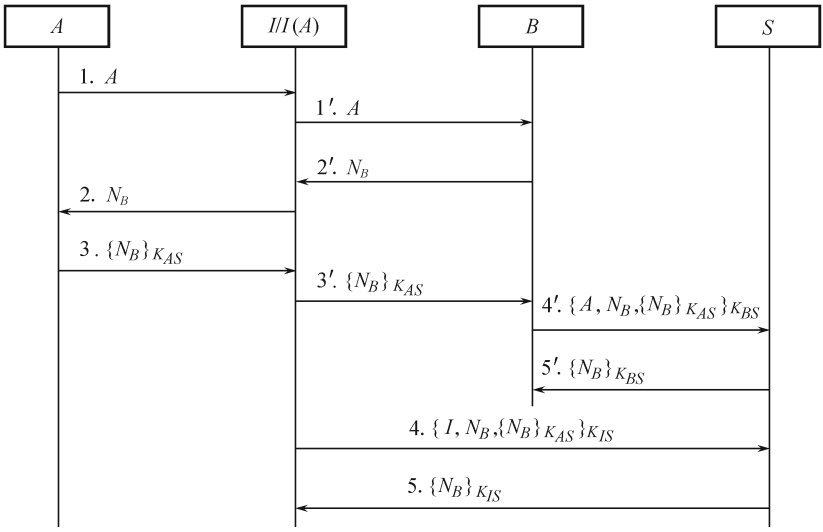


Fig. 4.17 An attack on the refined Woo-Lam protocol version of Mao.

4) Upon receiving Message 3′, B passes the encryption $\{N_B\}_{K_{AS}}$ on to S in Message 4′ for future verification.

5) Upon receiving Message 4′, S could not find any abnormality since the received message is an encryption under K_{BS} and K_{AS}.

6) Message 5′ is the reply from S which contains the challenge N_B intended for A and B. On the basis of the reply containing N_B, B believes that A is active in this protocol run.

7) Upon receiving Message 3 containing $\{N_B\}_{K_{AS}}$, I can continue his session with A, and at last, successfully complete the protocol run.

This is a perfect attack, all principals including A, B and S could not find any abnormality. Upon termination of the run of this refined Woo-Lam protocol, B accepts "the run with A", but in fact, A has not launched the run with B at all, and A thinks that A has completed a protocol run with I.

As we have shown above, although the principal name A is explicitly mentioned in Message 4, the absence of A's liveness still causes this flaw.

Example 4.16 Recall the Needham-Schroeder public-key authentication protocol and the security analysis of the Needham-Schroeder public-key protocol in Table 4.12. Since B could not guarantee the freshness of N_A and the association of N_A with A and B, there exists the interleaving attack discovered by Lowe[9].

Example 4.17 Recall Mao's revised Needham-Schroeder public-key 0-* protocol as shown in Fig. 4.18.

$$\text{Message 1} \quad A \to B : \left\{\{A, N_A\}_{K_A^{-1}}\right\}_{K_B}$$

Message 2 $B \rightarrow A : \left\{N_A, \{N_B\}_{K_B^{-1}}\right\}_{K_A}$

Message 3 $A \rightarrow B : \left\{\{N_B\}_{K_A^{-1}}\right\}_{K_B}$

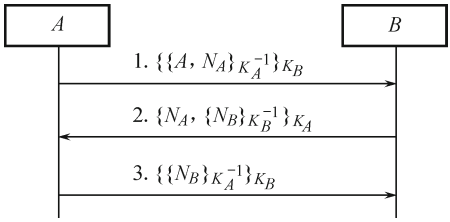


Fig. 4.18 The revised Needham-Schroeder public-key 0-* protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of the TVP N_A.

2) Upon receiving Message 1, B could not determine whether $\{\{A, N_A\}_{K_A^{-1}}\}_{K_B}$ is a replay message from the attacker or a new message generated by the opponent party A. B could not get any security assurances.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the TVP N_B.

4) Upon receiving Message 2, according to the freshness assurance of N_A, A is sure that only B could get N_A, using B's private key K_B^{-1}, and send N_A back to A in Message 2, hence, from Lemma 4.1, A has the liveness assurance of B. Since only A could decrypt $\{N_A, \{N_B\}_{K_B^{-1}}\}_{K_A}$ to get N_B, and N_B is signed by B's private key K_B^{-1}, from Lemma 4.3 and Lemma 4.4, A has the freshness assurance and the association assurance of the TVP N_B with A and B. So does N_A.

5) Similar case exists as in Message 3: upon receiving Message 3, from Lemma 4.2, Lemma 4.3, and Lemma 4.4, B has the confidentiality assurance, the freshness assurance, and the association assurance of the TVP N_A and N_B with A and B; from Lemma 4.1, B has the liveness assurance of A.

Upon termination of the Needham-Schroeder public-key 0-* protocol run, both A and B achieve the security objects of the Needham-Schroeder public-key. The analyzing result is indicated in Table 4.19.

Table 4.19 Security analysis of the refined Needham-Schroeder public-key 0-* protocol

	A			B		
	B	N_A	N_B	A	N_A	N_B
Message 1		11A			1?#	
Message 2	1	11AB	11AB			11#
Message 3				1	11AB	11AB
End of run	1	11AB	11AB	1	11AB	11AB

Example 4.18 Recall Mao's another revised Needham-Schroeder public-key 1-* protocol as shown in Fig. 4.19.

$$\begin{aligned}
&\text{Message 1} \quad A \rightarrow B : \{A, \{N_A\}_{K_B}\}_{K_A^{-1}} \\
&\text{Message 2} \quad B \rightarrow A : \{\{N_A, N_B\}_{K_A}\}_{K_B^{-1}} \\
&\text{Message 3} \quad A \rightarrow B : \{\{N_B\}_{K_B}\}_{K_A^{-1}}
\end{aligned}$$

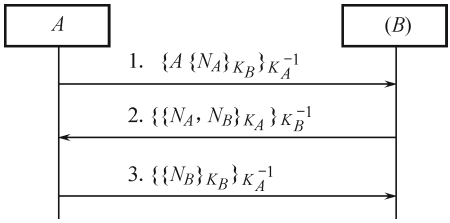


Fig. 4.19 The revised Needham-Schroeder public-key 1-* protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of the TVP N_A.

2) Upon receiving Message 1, B could not determine whether $\{A, \{N_A\}_{K_B}\}_{K_A^{-1}}$ is a replay message from the attacker or a new message generated by the opponent party A. B could not get any security assurances.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the TVP N_B.

4) Upon receiving Message 2, according to the freshness assurance of N_A, A is sure that only B could get N_A, using B's private key K_B^{-1}, and send N_A back to A in Message 2. Hence, from Lemma 4.3 and Lemma 4.4, A has the freshness assurance and the association assurance of the TVP N_A and N_B with A and B; from Lemma 4.1, A has the liveness assurance of B.

5) Similar case exists in Message 3: upon receiving Message 3, from Lemma 4.2, Lemma 4.3, and Lemma 4.4, B has the confidentiality assurance, the freshness assurance, and the association assurance of the TVP N_A and N_B with A and B; from Lemma 4.1, B has the liveness assurance of A.

Upon termination of the Needham-Schroeder public-key 1-* protocol run, both A and B achieve the security objects of the Needham-Schroeder public-key. The analyzing result is indicated in Table 4.20.

Table 4.20 Security analysis of the refined Needham-Schroeder public-key 1-* protocol

	A			B		
	B	N_A	N_B	A	N_A	N_B
Message 1		11#			1?#	
Message 2	1	11AB	11AB			11#
Message 3				1	11AB	11AB
End of run	1	11AB	11AB	1	11AB	11AB

4.3.7 Attack due to type flaw

An attack is stated due to type flaw in [22]: typical type flaws include a principal tricked to misinterpret a nonce, a timestamp or an identity into a key, etc. Misinterpretations are likely to occur when a protocol is poorly designed in that the type information of message components is not explicit, then type flaws can be very common in implementation. Let us check why type flaws exist using the security analysis apporach based on trusted freshness. In essence, type flaw attacks are generally applicable to a communication protocol where entity authentication or the freshness assurance of a TVP is absent.

Example 4.19 Neuman and Stubblebine[33] propose an authentication protocol, as shown in Fig. 4.20, to achieve mutual authentication and authenticated key establishment between A and B with the help of a trusted third party S.

Message 1 $A \rightarrow B : A, N_A$
Message 2 $B \rightarrow A : B, \{A, N_A, T_B\}_{K_{BS}}, N_B$
Message 3 $A \rightarrow B : \{B, N_A, k_{AB}, T_B\}_{K_{AS}}, \{A, k_{AB}, T_B\}_{K_{BS}}, N_B$
Message 4 $A \rightarrow B : \{A, k_{AB}, T_B\}_{K_{BS}}, \{N_B\}_{k_{AB}}$

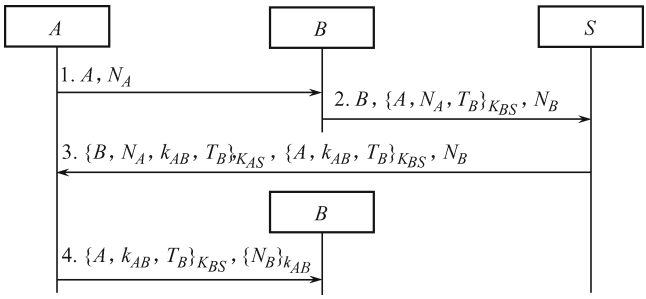


Fig. 4.20 The Neuman-Stubblebine authentication protocol.

Notation

A and B are two protocol principals, and S is a trusted third party. N_A and N_B are nonces, and T_B is a timestamp generated by B referring to an absolute time. K_{AS} and K_{BS} are shared long-term keys, and k_{AB} is a new session key between A and B to be established in this authentication protocol.

Premise

K_{AS} and K_{BS} are shared long-term keys between A and S, and B and S respectively, which are initially established by non-cryptographic, and out-of-band techniques. N_A or N_B is a nonce randomly chosen by A and B respectively, and T_B is a timestamp generated by B.

Protocol actions

1) In Message 1, A launches a new protocol run by sending the identity A and a randomly chosen nonce N_A.

2) In Message 2, B randomly chooses a nonce N_B, and then sends the identity B, and the nonce N_B with the encryption $\{A, N_A, T_B\}_{K_{BS}}$ to S to indicate a new protocol run between A and B.

3) Upon receiving Message 2, S gets N_A and N_B for this run between A and B, and then S randomly chooses a new session key k_{AB} for this run and keeps it secret via an encryption under K_{AS} and K_{BS} respectively.

4) Upon receiving Message 3, A gets the new session key k_{AB} via the shared long-term key K_{AS} known only by A and S.

5) In Message 4, A forwards $\{A, k_{AB}, T_B\}_{K_{BS}}$ received in Message 3 to B, and encrypts N_B under k_{AB} to show A's knowledge of k_{AB} to B.

6) Upon receiving Message 4, B gets the new session key k_{AB} via the shared long-term key K_{BS} known only by B and S, and confirms A's knowledge of k_{AB} via the encryption $\{N_B\}_{k_{AB}}$.

Successful execution should convince A and B that both entities are present and k_{AB} is a new session key between A and B.

Unfortunately, this protocol has not achieved these security objects as it intends to.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A believes that the TVP N_A is no longer confidential, but it is fresh for this run.

2) Upon receiving Message 1, B could not get any assurance about this protocol.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B believes that the TVP N_B is no longer confidential, but it is fresh for this run.

4) Upon receiving Message 2, since T_B is an absolute timestamp, S believes that the message $\{A, N_A, T_B\}_{K_{BS}}$ is fresh, and it could not be a replay one. Furthermore, since K_{BS} is only known by B and S, from Lemma 4.1 and Lemma 4.3, S believes that B is present and it must be B who has just generated this message $\{A, N_A, T_B\}_{K_{BS}}$. From Lemma 4.4, S believes that

N_A and N_B are new established for the protocol run between A and B from the explicitly identity of A and the possession of the key K_{BS} by B.

5) Upon receiving Message 3, from the point of view of A, since N_A is a trusted fresh TVP and K_{AS} is only known by A and S, from Lemma 4.2 and Lemma 4.3, A gets the confidentiality and freshness assurances of the new session key k_{AB}. From Lemma 4.4, A believes that k_{AB} is for this protocol run between A and B from the explicitly identity of B and the possession of the key K_{AS} by A. According to the protocol semantic cue, the principal in $\{B, N_A, k_{AB}, T_B\}_{K_{AS}}$ is the one who has confirmed his identity and liveness to the third trusted party S. From Lemma 4.1, since N_A is a trusted fresh TVP from the point of view of A, A believes that it could only be S who has generated and sent the message $\{B, N_A, k_{AB}, T_B\}_{K_{AS}}$ after S has checked the liveness of B via the message of $\{A, N_A, T_B\}_{K_{BS}}$, hence A believes the liveness of S and B.

6) Upon receiving Message 4, since T_B is an absolute timestamp, B believes that the message $\{A, k_{AB}, T_B\}_{K_{BS}}$ is fresh, but the maximal term $\{A, k_{AB}, T_B\}_{K_{BS}}$ in Message 4 is similar to the maximal term $\{A, N_A, T_B\}_{K_{BS}}$ in Message 2, so $\{A, k_{AB}, T_B\}_{K_{BS}}$ may be a replay one of Message 2. Hence, B could not get any new assurance about this protocol.

The maximal term $\{A, k_{AB}, T_B\}_{K_{BS}}$ includes an identity of A, a random session key and an absolute timestamp (or we can call it a trusted freshness TVP), and it can be constructed via term definition, terms are T_B, $\{T_B\}_{K_{BS}}$, $\{k_{AB}, T_B\}_{K_{BS}}$, $\{A, T_B\}_{K_{BS}}$, then $\{A, k_{AB}, T_B\}_{K_{BS}}$. Similar case exists in the maximal term $\{A, N_A, T_B\}_{K_{BS}}$ of Message 2. Hence, $\{A, N_A, T_B\}_{K_{BS}}$ could be replayed in Message 4 as Mao has stated in [22]. Note that $\{N_B\}_{k_{AB}}$ is also a maximal term in Message 4.

Table 4.21 Security analysis of the Neuman-Stubblebine authentication protocol

	A					B				
	B	S	N_A	N_B	k_{AB}	A	S	N_A	N_B	k_{AB}
Message 1			01#					0?#		
Message 2									01#	
Message 3	1	1	01AB		11AB					
Message 4										
End of run	1	1	01AB		11AB			0?#	01#	

Upon termination of the protocol run, by analyzing of Table 4.21shows that B is not sure whether the opponent principal is present or not, and whether k_{AB} is a new session key for A and B or not.

From the absence of the liveness of the principal A and the association of k_{AB} with A and B in the point of view of B, the adversary I could launch an attack as shown in Fig. 4.21 by impersonating A[22].

$$\begin{array}{lll} \text{Message 1} & I(A) \rightarrow B: & A, N_A \\ \text{Message 2} & B \rightarrow I(S): & B, \{A, N_A, T_B\}_{K_{BS}}, N_B \end{array}$$

Message 3 $I(S) \rightarrow I(A)$: none

Message 4 $I(A) \rightarrow B$: $\{A, N_A, T_B\}_{K_{BS}}, \{N_B\}_{N_A}$

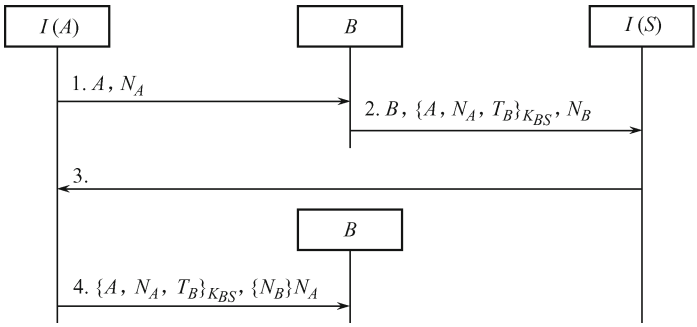


Fig. 4.21 An attack on the Neuman-Stubblebine authentication protocol.

Notation

$I(A)$ and $I(S)$ are adversaries impersonating A and S respectively.

Protocol actions

1) In Message 1, the adversary launches a new protocol run with a randomly chosen nonce N_A by impersonating A.

2) In Message 2, B does the same as in the original Neuman-Stubblebine authentication protocol.

3) I intercepts Message 2 to get N_B, then encrypts N_B using I's randomly chosen nonce N_A and sends $\{N_B\}_{N_A}$ to B to show A's knowledge of k_{AB} by confusing N_A with k_{AB}.

Upon termination of the protocol run, B believes that B has performed a successful protocol run with A, and N_A is a new session key for A and B, while A knows nothing about this key establishment procedure.

4.3.8 Attack due to name omission

Name omission is often the case in authentication protocols for the name information about a message can be deduced from other data parts in the context, or from what encryption keys have been applied to. To obtain an elegant protocol that contains little redundancy, protocol designers may omit the identities of the participants, which may lead to name-omission flaws. In security analysis based on trusted freshness, entity authentication or the association assurance of a TVP may be absent due to name omission. Hence, attack due to name omission can be constructed from these absences.

Example 4.20 Denning and Sacco propose a public-key protocol as an

alternative to their fix of the Needham-Schroeder shared key protocol[1, 31]. The protocol of Denning and Sacco is as shown in Fig. 4.22.

Message 1 $A \rightarrow S : A, B$

Message 2 $S \rightarrow A : \text{Cert}_A, \text{Cert}_B$

Message 3 $A \rightarrow B : \text{Cert}_A, \text{Cert}_B, \left\{\{k_{AB}, T_A\}_{K_A^{-1}}\right\}_{K_B}$

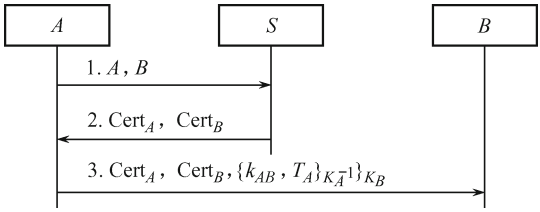


Fig. 4.22 The Denning and Sacco authentication protocol.

Notation

A and B are two protocol principals, and S is a trusted third party. T_A is a timestamp generated by A referring to an absolute time. K_A (K_B) and K_A^{-1} (K_B^{-1}) are public-key and private key of A (B) respectively. Cert_A (Cert_B) is a certification of A's (or B's) identity and corresponding public-key (K_A) signed by a trusted certification authority center CA. k_{AB} is a new session key between A and B to be established in this authentication protocol.

Premise

Both A and B know the public-key of the trusted certification authority center CA to get K_A and K_B. Each principal knows the key pair of himself, that is, K_A and K_A^{-1} for A, and K_B and K_B^{-1} for B.

Protocol actions

1) In Message 1, A launches a new protocol run of mutual authentication and authenticated key establishment between A and B.

2) Upon receiving Message 2, A gets the public-key K_B of B.

3) In Message 3, $\{\{k_{AB}, T_A\}_{K_A^{-1}}\}_{K_B}$ is encrypted for confidentiality (under K_B) and authenticity (under K_A^{-1}).

4) Upon receiving Message 3, B gets the public-key K_A of A, and B sees that the new session key k_{AB} should be exclusively shared between A and B from B's private key K_B^{-1} and A's public-key K_A being applied. Note k_{AB} is randomly chosen by A for this protocol run.

Successful execution should convince A and B that k_{AB} is a new session key between A and B.

Unfortunately, the Denning and Sacco authentication protocol has not achieved the security objects as it intends to.

Protocol security analysis

1) In Message 1 and Message 2, neither A nor B could get any assurance about this protocol.

2) In Message 3, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of the new session key k_{AB}. From Lemma 4.4, A has the association assurance of k_{AB} with A, since the "loose" one-way transformation $\{k_{AB}, T_A\}_{K_A^{-1}}$ could only be generated by A.

3) Upon receiving Message 3, since T_B is an absolute timestamp, B believes that the message $\{k_{AB}, T_A\}_{K_A^{-1}}$ is fresh, and it could not be a replay one. Hence, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the new session key k_{AB}. From Lemma 4.4, B has the association assurance of k_{AB} with A, since the "loose" one-way transformation $\{k_{AB}, T_A\}_{K_A^{-1}}$ could only be generated by A.

Table 4.22 means that upon termination of the protocol run B believes A is present, and the new session key k_{AB} is confidential, fresh, and associated with A but B.

Table 4.22 Security analysis of the Denning-Sacco protocol

	A		B	
	B	k_{AB}	A	k_{AB}
Message 1				
Message 2				
Message 3		11A	1	11A
End of run		11A		11A

Example 4.21 From the absence of the k_{AB}'s association with B in the point of view of B, the adversary can perform an attack by confusing the session key k'_{AB}, as shown in Fig. 4.23, intended for A and I to be the session key between A and B[10].

$$
\begin{aligned}
&\text{Message 1} \quad A \to S: && A, I\\
&\text{Message 2} \quad S \to A: && \mathrm{Cert}_A,\ \mathrm{Cert}_I\\
&\text{Message 3} \quad A \to I: && \mathrm{Cert}_A,\ \mathrm{Cert}_I,\ \left\{\{k'_{AB}, T_A\}_{K_A^{-1}}\right\}_{K_I}\\
&\text{Message 1}' \quad I(A) \to S: && A, B\\
&\text{Message 2}' \quad S \to I(A): && \mathrm{Cert}_A,\ \mathrm{Cert}_B\\
&\text{Message 3}' \quad I(A) \to B: && \mathrm{Cert}_A,\ \mathrm{Cert}_B,\ \left\{\{k'_{AB}, T_A\}_{K_A^{-1}}\right\}_{K_B}
\end{aligned}
$$

Notation

$I(A)$ is an adversary I impersonating A. K_I and K_I^{-1} are the public-key and private key of I. Cert_I is a certification of I's identity and corresponding public-key (K_I) signed by a trusted certification authority center CA.

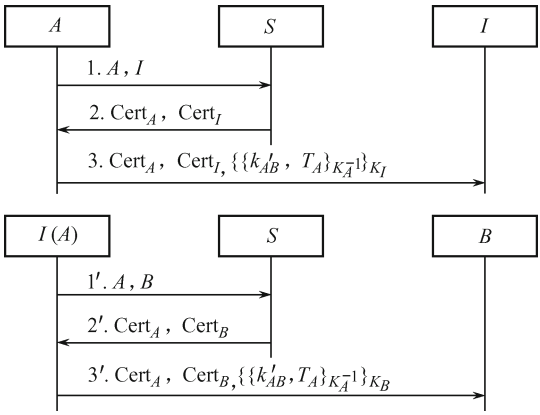


Fig. 4.23 An attack on the Denning-Sacco authentication protocol.

Premise

All principals A, B and I know the public-key of the trusted certification authority center CA to get K_A, K_B and K_I. Each principal knows the key pair of himself, that is, K_A and K_A^{-1} for A, K_B and K_B^{-1} for B, and K_I and K_I^{-1} for I.

Protocol actions

1) In Message 1, A launches a new protocol run of mutual authentication and makes authenticated key establishment between A and I.

2) Upon receiving Message 2, A gets the public-key K_I of I.

3) In Message 3, $\{\{k'_{AB}, T_A\}_{K_A^{-1}}\}_{K_I}$ is encrypted for confidentiality (under K_I) and authenticity (under K_A^{-1}).

4) Upon receiving Message 3, I gets the new session key k'_{AB}. Then I could launch an attack by impersonating A and use the transformation $\{k'_{AB}, T_A\}_{K_A^{-1}}$ to generate Message 3′ $\{\text{Cert}_A, \text{Cert}_B, \{\{k'_{AB}, T_A\}_{K_A^{-1}}\}_{K_B}\}$.

Upon termination of the protocol attack, B believes that B has performed a successful run with A, and k'_{AB} is a new session key for A and B, while A knows nothing about this key establishment between A and B, and k'_{AB} is actually shared by A and I.

4.3.9 Attack due to misuse of cryptographic services

The attack due to misuse of cryptographic services is a very common design flaw as stated in [22]. Misuse of cryptographic services means that a cryptographic algorithm used in a protocol provides an incorrect protection so that the needed protection is absent. This type of flaw may lead to various attacks.

Example 4.22 Recall Example 3.15, the Otway and Rees key establishment protocol is illustrated as in Fig. 4.24.

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

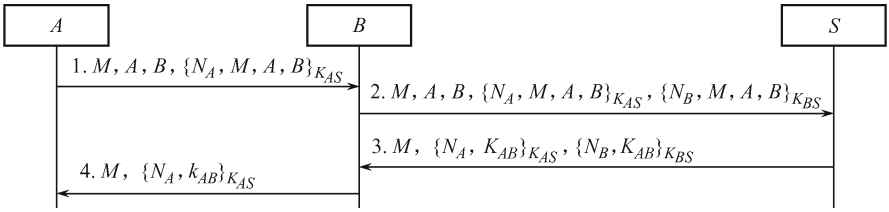


Fig. 4.24 The Otway-Rees key establishment protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of the TVP N_A and the freshness assurance of the TVP M. From Lemma 4.4, A has the association assurance of N_A with A and B, since the transformation $\{N_A, M, A, B\}_{K_{AS}}$ generated by A includes the identities of both A and B, which implies the association of both A and B.

2) Upon receiving Message 1, B has the confidentiality assurance of the TVP N_A.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the TVP N_B. From Lemma 4.4, B has the association assurance of N_B with A and B, since the transformation $\{N_B, M, A, B\}_{K_{BS}}$ generated by B includes the identities of both A and B, which implies the association of both A and B.

4) Upon receiving Message 2, S could not get any assurance about this protocol. A could not get any new assurance since $\{N_B, M, A, B\}_{K_{BS}}$ is an encryption for B without corresponding decrypted key K_{BS}.

5) Upon receiving Message 3, from Lemma 4.3, B has the liveness assurance of the trusted third party S from the transformation $\{N_B, k_{AB}\}_{K_{BS}}$ including B's trusted freshness component N_B, since only S could get N_B and generate this message $\{N_B, k_{AB}\}_{K_{BS}}$. From the one-way transformation $\{N_B, k_{AB}\}_{K_{BS}}$ including the trusted freshness N_B, and from Lemmas 4.2, 4.3, and 4.4, B has the confidentiality assurance, freshness assurance of k_{AB}, and also the association assurance of k_{AB} with both A and B.

6) Similarly, upon receiving Message 4, from Lemma 4.2, Lemma 4.3, and Lemma 4.4, A has the liveness assurance of the trusted third party S from the receiving fresh nonce N_A, and A also has the confidentiality assurance,

freshness assurance of k_{AB}, and also the association assurance of k_{AB} with A and B. From Lemma 4.3, A has the liveness assurance of B from the receiving fresh nonce N_A and M.

Upon termination of this protocol run, B believes that S is alive from receiving fresh nonce N_B, and A believes that S is alive from receiving fresh nonce N_A, and B's liveness is also guaranteed to A via M, but A's liveness is not guaranteed to B in this protocol. The analyzing result is indicated in Table 4.23.

Table 4.23 Security analysis of the Otway-Rees protocol

	A					B				
	B	S	N_A	N_B	k_{AB}	A	S	N_A	N_B	k_{AB}
Message 1			$11AB$					1?#		
Message 2				1?#					$11AB$	
Message 3							1			$11AB$
Message 4	1	1	$11AB$		$11AB$					
End of run	1	1			$11AB$		1			$11AB$

From the absence of the A's liveness, there exists an attack as stated in Example 3.15: the adversary I has recorded the message $\{M, A, B, \{N_A, M, A, B\}_{K_{AS}}\}$ in an old protocol run, then I can launch a new protocol run by impersonating A.

4.3.10 Security analysis of other protocols

Example 4.23 The KryptoKnight protocol[34] as shown in Fig. 4.25 is a mutual authentication protocol based on symmetric-key cryptography. N_A serves as nonce for entity authentication of B and N_B serves as nonce for entity authentication of A.

$$\begin{aligned}
&\text{Message 1} \quad A \rightarrow B : A, B, N_A \\
&\text{Message 2} \quad B \rightarrow A : B, A, N_B, H(N_A, N_B, B, K_{AB}) \\
&\text{Message 3} \quad A \rightarrow B : A, B, H(N_A, N_B, K_{AB})
\end{aligned}$$

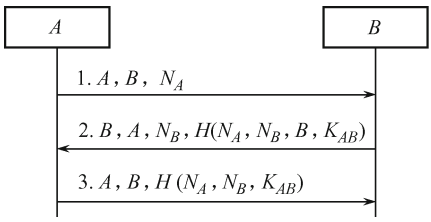


Fig. 4.25 The KryptoKnight mutual authentication protocol.

Notation

A and B are two protocol principals. N_A and N_B are nonces. K_{AB} is a shared long-term key. $H(x_1, x_2, ..., k)$ is a keyed cryptographic hash function.

Premise

K_{AB} is the shared long-term key between A and B, which is initially established by non-cryptographic, and out-of-band techniques; N_A, N_B are nonces generated by A and B respectively.

Protocol actions

1) In Message 1, A launches a new protocol run by sending the identities of A, B and a randomly chosen nonce N_A.

2) B randomly chooses a nonce N_B, and sends Message 2 to A to indicate the liveness of B. Message 2 includes the identities of A, B and the hash value $H(N_A, N_B, B, K_{AB})$.

3) Upon receiving Message 2, A gets N_B, and then recalculates and compares $H(N_A, N_B, B, K_{AB})$ with the received hash value. If it matches, then A confirms the liveness of B.

4) In Message 3, A generates and sends the hash value $H(N_A, N_B, K_{AB})$ to B to indicate the liveness of A.

5) Upon receiving Message 3, B recalculates and compares $H(N_A, N_B, K_{AB})$ with the received hash value. If it matches, then B knows that it must be A who has generated and sent the hash value $H(N_A, N_B, K_{AB})$ using their shared long-term key K_{AB}.

Successful execution should convince A and B that both entities are present.

Protocol security analysis

1) In Message 1, from Lemma 4.3, A has the freshness assurance of the TVP N_A.

2) Upon receiving Message 1, B couldn't get any assurance about this protocol.

3) In Message 2, from Lemma 4.3, B has the freshness assurance of the TVP N_B.

4) Upon receiving Message 2, A gets N_B, recalculates $H(N_A, N_B, B, K_{AB})$, and compares the recalculation value with the received hash value. If it matches, from Lemma 4.2, then A knows that it must be B who has generated and sent the hash value $H(N_A, N_B, B, K_{AB})$ including the trusted freshness N_A using their shared long-term key K_{AB}. Hence, the liveness of B is authenticated.

5) Similarly, upon receiving Message 3, B recalculates $H(N_A, N_B, K_{AB})$, and compares the recalculation value with the received hash value. If it matches, from Lemma 4.2, then B knows that it must be A who has generated and sent the hash value $H(N_A, N_B, K_{AB})$ including the trusted freshness N_B

using their shared long-term key K_{AB}. Hence, the liveness of A is authenticated.

Upon termination of the protocol run, A believes that B is alive from comparing the hash value $H(N_A, N_B, K_{AB})$ including the trusted fresh nonce N_A, and B believes that A is alive from comparing the hash value $H(N_A, N_B, K_{AB})$ including the trusted fresh nonce N_B.

The security analysis result in Table 4.24 based on the trusted freshness shows that the KryptoKnight protocol has achieved the mutual authentication objectives as it intends to.

Table 4.24 Security analysis of the KryptoKnight protocol

	A			B		
	B	N_A	N_B	A	N_A	N_B
Message 1		01#			0?#	
Message 2	1	01AB	01AB			01AB
Message 3				1	01AB	
End of run	1			1		

Example 4.24 The Yahalom protocol[35] is a classical authenticated key establishment protocol as shown in Fig. 4.26. The protocol intends to establish a new session key k_{AB} between A and B with the help of a server S, and to achieve mutual authentication.

$$\begin{aligned}
&\text{Message 1} \quad A \rightarrow B: \quad A, N_A \\
&\text{Message 2} \quad B \rightarrow S: \quad B, \{A, N_A, N_B\}_{K_{BS}} \\
&\text{Message 3} \quad S \rightarrow A: \quad \{B, k_{AB}, N_A, N_B\}_{K_{AS}}, \{A, k_{AB}\}_{K_{BS}} \\
&\text{Message 4} \quad I(A) \rightarrow B: \{A, k_{AB}\}_{K_{BS}}, \{N_B\}_{k_{AB}}
\end{aligned}$$

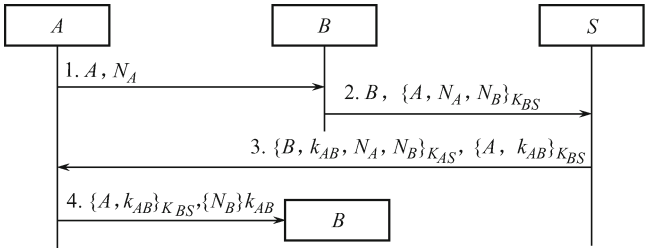


Fig. 4.26 The Yahalom protocol.

Notation

A and B are two protocol principals, and S is a trusted third party. N_A and N_B are nonces. K_{AS} and K_{BS} are shared long-term keys. k_{AB} is a new session key between A and B to be established in this authentication protocol.

Premise

K_{AS} and K_{BS} are shared long-term keys between A and S, and B and S respectively, which are initially established by non-cryptographic, and out-of-band techniques. N_A, N_B are nonces generated by A and B respectively.

Protocol actions

1) In Message 1, A launches a new protocol run by sending the identity of A and a randomly chosen nonce N_A.

2) In Message 2, B randomly chooses a nonce N_B, and sends the encryption $\{A, N_A, N_B\}_{K_{BS}}$ to S to show the ownership of the long-term key K_{BS}.

3) Upon receiving Message 2, S gets N_A and N_B.

4) In Message 3, S calculates $\{B, k_{AB}, N_A, N_B\}_{K_{AS}}$, $\{A, k_{AB}\}_{K_{BS}}$ using the shared long-term key K_{AS} and K_{BS} and sends Message 3 to A.

5) Upon receiving Message 3, A gets N_B and the new session key k_{AB} from the encryption $\{B, k_{AB}, N_A, N_B\}_{K_{AS}}$.

6) In Message 4, A encrypts N_B using the new session key k_{AB} to show the liveness of A and the knowledge of k_{AB}.

7) Upon receiving Message 4, B gets the new session key k_{AB} from the encryption $\{A, k_{AB}\}_{K_{BS}}$, and then B checks A's liveness and A's knowledge of k_{AB} via $\{N_B\}_{k_{AB}}$.

Successful execution should convince A and B that both entities are present and k_{AB} is the new session key for A and B.

Protocol security analysis

1) In Message 1, from Lemma 4.3, A has the freshness assurance of the TVP N_A.

2) Upon receiving Message 1, B couldn't get any assurance about this protocol.

3) In Message 2, from Lemma 4.2, both A and B have the confidentiality assurance of N_B. From Lemma 4.3, B has the freshness assurance of the TVP N_B. From Lemma 4.4, B has the association assurance of N_B with A and B.

4) Upon receiving Message 2, S gets N_A and N_B using the long-term key K_{BS}. From Lemma 4.2, S has the confidentiality assurance of N_B, but S couldn't get any assurance about the liveness of B and the freshness of N_A and N_B for there is not any trusted freshness of S in the transformation $\{A, N_A, N_B\}_{K_{BS}}$.

5) Upon receiving Message 3, from Lemma 4.1, A has the liveness assurances of both B and S. From Lemma 4.2, A has the confidentiality assurance of N_B and k_{AB}. From Lemma 4.3, A has the freshness assurance of N_B and k_{AB}. From Lemma 4.4, A has the association assurance of N_A, N_B and k_{AB} with A and B.

6) Upon receiving Message 4, from Lemma 4.1, B has the liveness assurance of A. From Lemma 4.2, B has the confidentiality assurance of k_{AB}.

From Lemma 4.3, B has the freshness assurance of k_{AB} from the trusted freshness N_B. From Lemma 4.4, B has the association assurance of k_{AB} with A and B from the trusted freshness N_B's association with A and B.

Upon termination of this protocol run, the security analysis result in Table 4.25 based on the trusted freshness shows that the Yahalom protocol has achieved the mutual authentication and key establishment objectives.

Table 4.25 Security analysis of the Yahalom protocol

	A					B					S		
	B	S	N_A	N_B	k_{AB}	A	S	N_A	N_B	k_{AB}	N_A	N_B	k_{AB}
Message 1			$01\#$					$0?\#$			$0?\#$		
Message 2				$1?\#$					$11AB$			$1?\#$	
Message 3	1	1	$01AB$	$11AB$	$11AB$								$1?\#$
Message 4						1				$11AB$			
End of run	1				$11AB$	1				$11AB$			

Example 4.25 The TMN protocol[36] is a key establishment protocol based on asymmetric-key cryptography with trusted third party, as illustrated in Fig. 4.27.

$$
\begin{aligned}
&\text{Message 1} \quad A \to S : A, S, B, \{k_A\}_{K_S} \\
&\text{Message 2} \quad S \to B : S, B, A \\
&\text{Message 3} \quad B \to S : B, S, A, \{k_B\}_{K_S} \\
&\text{Message 4} \quad S \to A : S, A, B, V(k_A, k_B)
\end{aligned}
$$

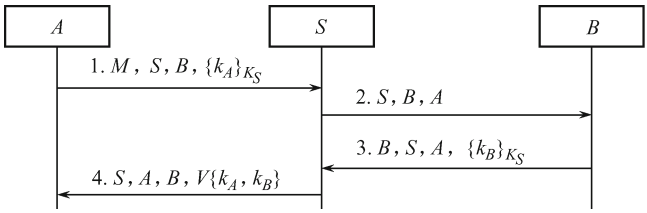


Fig. 4.27 The TMN protocol.

Notation

A and B are two protocol principals, and S is a trusted third party. K_S is a public-key of S. k_A and k_B are randomly chosen input by A and B respectively, and k_B is also the new session key to be established between A and B in the TMN protocol.

The Vernam encryption $V(k_1, k_2)$ is the bit XOR of the two keys k_1 and k_2 where $V(k_1, V(k_1, k_2)) = k_2$. Suppose the randomly input keys k_1 and k_2 are redundancy, hence the receiver could determine whether the received encryption $V(k_1, k_2)$ is correctly decrypted or not.

Premise

K_S and K_S^{-1} are the public-key and the private key of the trusted third party S, which is initially established by non-cryptographic, and out-of-band techniques. k_B is the new session key to be established between A and B in this protocol, and k_B can be recovered by A from the Vernam encryption $V(k_A, k_B)$.

Protocol actions

1) In Message 1, A launches a new protocol run by sending the identities of A, S, B and the encryption of a randomly chosen key input k_A under the public-key K_S.

2) Upon receiving Message 1, S gets k_A from the encryption of $\{k_A\}_{K_S}$ using S's private key K_S^{-1}.

3) In Message 2, S notices B launching a new session between A and B by sending the identities of A, B and S.

4) In Message 3, B randomly chooses a new key k_B for this session between A and B, and sends S the encryption of k_B under S's public-key K_S.

5) Upon receiving Message 3, S gets k_B from the encryption of $\{k_B\}_{K_S}$ using S's private key K_S^{-1}.

6) In Message 4, S sends A the new session key k_B via the Vernam encryption $V(k_A, k_B)$.

7) Upon receiving Message 4, A resumes the new session key k_B via the ownership of k_A and the recalculation of $V(k_A, V(k_A, k_B)) = k_B$.

Successful execution should establish a new session key k_B between A and B.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and freshness assurances of the TVP k_A.

2) Upon receiving Message 1, from Lemma 4.2, B has the confidentiality assurance of the TVP k_A.

3) In Message 2, neither A nor B could get any new assurance about this protocol.

4) In Message 3, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and freshness assurances of the TVP k_B.

5) Upon receiving Message 3, from Lemma 4.2, S has the confidentiality assurance of the TVP k_B. A also has the confidentiality assurance of the TVP k_B.

6) Upon receiving Message 4, from Lemma 4.2, A has the confidentiality assurance of the TVP k_B. From Lemma 4.3, A has the freshness assurance of the TVP k_B from the trusted freshness k_A.

Upon termination of the protocol run, the security analysis result in Table 4.26 shows that: both A and B are not sure whether the opponent principal is present or not, and whether k_B is a new session key for A and B or not.

From the absence of the security properties of an authentication protocol, various attacks could be directly constructed[37].

Table 4.26 Security analysis of the TMN protocol

	A				B			
	B	S	k_A	k_B	A	S	k_A	k_B
Message 1			11#				1?#	
Message 2								
Message 3				1?#				11#
Message 4				11#				
End of run				11#				11#

Example 4.26 (Attack 1 on the TMN protocol) From the absence of A's liveness, the attacker may launch an attack without the presence of A, as illustrated in Fig. 4.28.

$$\begin{aligned}
&\text{Message 1} \quad I(A) \rightarrow S : A, S, B, \{k'_A\}_{K_S} \\
&\text{Message 2} \quad S \rightarrow B : S, B, A \\
&\text{Message 3} \quad B \rightarrow S : B, S, A, \{k_B\}_{K_S} \\
&\text{Message 4} \quad S \rightarrow I(A) : S, A, B, V(k'_A, k_B)
\end{aligned}$$

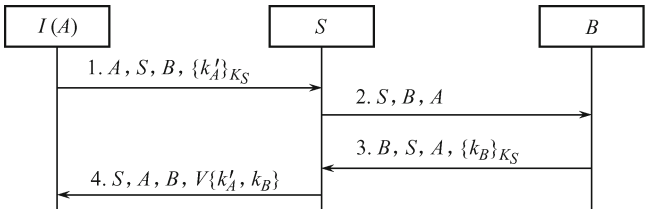


Fig. 4.28 An attack on the TMN protocol by impersonating A.

Notation

$I(A)$ is the adversary I impersonating A.

Premise

k'_A is randomly chosen by I as the nonce to launch a new session between A and B by impersonating A. k_B is randomly chosen by B as the new session key to be established between A and B.

Protocol actions

1) In Message 1, the adversary I randomly chooses a TVP k'_A to launch a new session between A and B by impersonating A.

2) Upon receiving Message 1, S gets k'_A from the encryption of $\{k'_A\}_{K_S}$ using S's private key K_S^{-1}.

3) In Message 2, S notices B launching a new session between A and B by sending the identities of A, B and S.

4) In Message 3, B randomly chooses a new key k_B for this session between A and B (actually between I and B), and sends S the encryption of k_B under S's public-key K_S.

5) Upon receiving Message 3, S gets k_B from the encryption of $\{k_B\}_{K_S}$ using S's private key K_S^{-1}.

6) In Message 4, S sends A the new session key k_B via the Vernam encryption $V(k'_A, k_B)$.

7) Upon receiving Message 4, $I(A)$ resumes the new key k_B via the ownership of k'_A and the recalculation of $V(k'_A, V(k'_A, k_B)) = k_B$.

Upon termination of the attack on the TMN protocol, the adversary I causes B to have false beliefs: B has completed a successful protocol run with A, and is sharing a new session key k_B with A, but actually shares the key k_B with I. Furthermore, B concludes that subsequently messages could be encrypted using k_B and safely transmitted to A (actually known by I), whereas in fact, A knows nothing about the key establishment.

Example 4.27 (Attack 2 on the TMN protocol) From the absence of the B's liveness, the attacker may launch an attack without the presence of B, as illustrated in Fig. 4.29.

$$
\begin{aligned}
&\text{Message 1} \quad A \to S: \quad A, S, B, \{k_A\}_{K_S} \\
&\text{Message 2} \quad S \to I(B): S, B, A \\
&\text{Message 3} \quad I(B) \to S: B, S, A, \{k'_B\}_{K_S} \\
&\text{Message 4} \quad S \to A: \quad S, A, B, V(k_A, k'_B)
\end{aligned}
$$

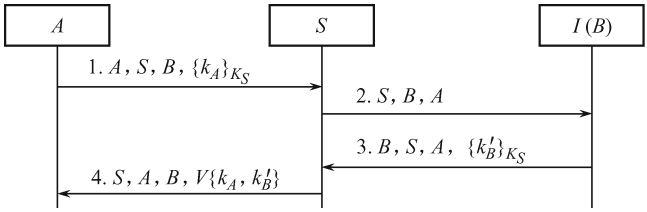


Fig. 4.29 An attack on the TMN protocol by impersonating B.

Notation

$I(B)$ is the adversary I impersonating B.

Premise

k'_B is randomly chosen by I as the new session key to be established between A and B by impersonating B.

Protocol actions

1) In Message 1, A launches a new protocol run by sending the identities of A, S, B and the encryption of a randomly chosen key input k_A under S's public-key K_S.

2) Upon receiving Message 1, S gets k_A from the encryption of $\{k_A\}_{K_S}$ using S's private key K_S^{-1}.

3) In Message 2, S notices B launching a new session between A and B by sending the identities of A, B and S.

4) In Message 3, the adversary I randomly chooses a TVP k'_B as the new session key between A and B by impersonating B (actually between I and B), and sends S the encryption of k'_B under S's public-key K_S.

5) Upon receiving Message 3, S gets k'_B from the encryption of $\{k'_B\}_{K_S}$ using S's private key K_S^{-1}.

6) In Message 4, S sends A the new session key k'_B via the Vernam encryption $V(k_A, k'_B)$.

7) Upon receiving Message 4, A resumes the new key k'_B via the ownership of k_A and the recalculation of $V(k_A, V(k_A, k'_B)) = k'_B$.

Upon termination of the attack on the TMN protocol, the adversary I causes A to have false beliefs: A has completed a successful protocol run with B, and is sharing a new session key k'_B with B, but actually shares the key k'_B with I. Furthermore, A concludes that subsequently messages could be encrypted using k'_B and safely transmitted to B (actually known by I), whereas in fact, B knows nothing about the key establishment.

Example 4.28 (Attack 3 on the TMN protocol) From the absence of the key association assurance of k_B, the adversary I could get the secret new session key k_B which is intended only for A and B, as illustrated in Fig. 4.30.

Message 1	$A \to I(A):$	$A, S, B, \{k_A\}_{K_S}$
Message 1′	$I(A) \to S:$	$A, S, B, \{k_A\}_{K_S}$
Message 2′	$S \to I(B):$	S, B, A
Message 3′	$I(B) \to S:$	$B, S, A, \{k'_B\}_{K_S}$
Message 4′	$S \to I(A):$	$S, A, B, V(k_A, k'_B)$
		(Now I knows k_A from $V(k'_B, V(k_A, k'_B)) = k_A$)
Message 1″	$A \to S:$	$A, S, B, \{k_A\}_{K_S}$
Message 2	$S \to B:$	S, B, A
Message 3	$B \to S:$	$B, S, A, \{k_B\}_{K_S}$
Message 4″	$S \to I(A):$	$S, A, B, V(k_A, k_B)$
		(Now I knows k_B from $V(k_A, V(k_A, k_B)) = k_B$)
Message 4	$I(S) \to A:$	$S, A, B, V(k_A, k_B)$

Notation

$I(A)$ or $I(B)$ is the adversary I impersonating A and B independently.

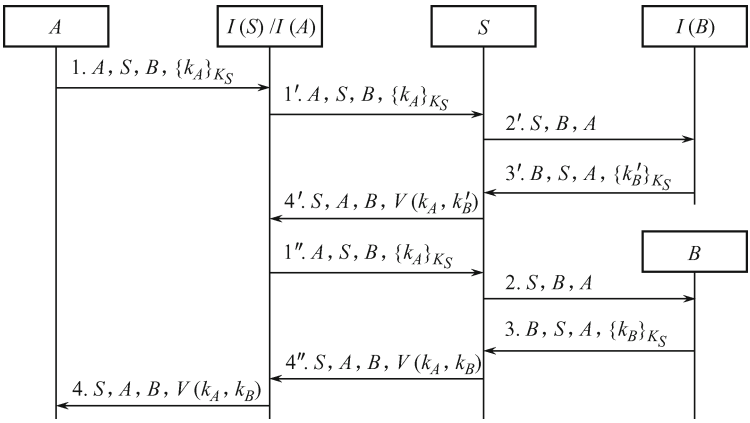


Fig. 4.30 An attack on the TMN protocol to get the secret new session key k_B between A and B by impersonating both A and B.

Premise

k'_B is randomly chosen by I as the new session key to be established between A and B by impersonating B.

Protocol actions

1) In Message 1, A launches a new protocol run by sending the identities of A, S, B and the encryption of a randomly chosen key input k_A under the public-key K_S of S.

2) The adversary I intercepts Message 1 and replays this message as Message $1'$ to S by impersonating A.

3) In Message $2'$, S notices B launching a new session between A and B by sending the identities of A, B and S.

4) The adversary I intercepts Message $2'$, and randomly chooses a TVP k'_B as the new session key between A and B by impersonating B (actually between I and A), and sends S the encryption of k'_B under S's public-key K_S.

5) Upon receiving Message $3'$, S gets k'_B from the encryption of $\{k'_B\}_{K_S}$ using S's private key K_S^{-1}.

6) In Message $4'$, S sends A (Indeed I) the new session key k'_B via the Vernam encryption $V(k_A, k'_B)$.

7) Upon receiving Message $4'$, the adversary I intercepts Message $4'$ and resumes k_A via the ownership of the TVP k'_B and the recalculation of $V(k'_B, V(k_A, k'_B)) = k_A$.

8) In Message $1''$, the adversary I forwards Message 1 to S, indicating a new protocol run between A and B.

9) Upon receiving Message $1''$, S gets k_A from the encryption of $\{k_A\}_{K_S}$ using S's private key K_S^{-1}.

10) In Message 2, S notices B the new session between A and B by sending the identities of A, B and S.

11) In Message 3, B randomly chooses a new key k_B for this session between A and B, and sends S the encryption of k_B under S's public-key K_S.

12) Upon receiving Message 3, S gets k_B from the encryption of $\{k_B\}_{K_S}$ using S's private key K_S^{-1}.

13) In Message $4''$, S sends A the new session key k_B via the Vernam encryption $V(k_A, k_B)$.

14) The adversary I intercepts Message $4''$, and resumes the new key k_B via the ownership of k_A and the recalculation of $V(k_A, V(k_A, k_B)) = k_B$. Up to now, I knows the session key k_B which is intended to be secret and only known by A and B.

15) In Message 4, the adversary I forwards Message $4''$ as Message 4 to A.

16) Upon receiving Message 4, A resumes the new key k_B via the ownership of k_A and the recalculation of $V(k_A, V(k_A, k_B)) = k_B$.

Upon termination of the attack on the TMN protocol, the adversary I has gotten the secret new session key k_B between A and B, which is intended to be secret and only known by A and B. As a result of this, the adversary I could get the subsequent sensitive encryption communicated between A and B, via using the decryption key k_B.

Note that Message $1''$ and Message 1 are actually the same message from A to S, and the adversary I has intercepted this message. Similar case exists in Message $4''$ and Message 4.

Example 4.29 The big mouth frog protocol[38] is a key transport protocol based on symmetric-key cryptography with trusted third party, as illustrated in Fig. 4.31.

$$\begin{array}{ll} \text{Message 1} & A \to S : A, \{B, k_{AB}\}_{K_{AS}} \\ \text{Message 2} & S \to B : \{A, k_{AB}\}_{K_{BS}} \end{array}$$

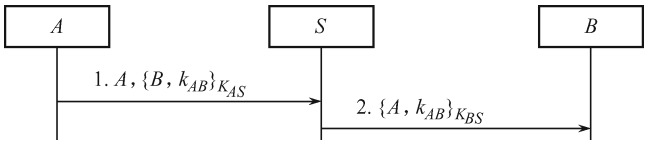


Fig. 4.31 The big mouth frog protocol.

Notation

A and B are two protocol principals, and S is a trusted third party. k_{AB} is randomly chosen by A as the new session key to be established between A and B.

Premise

K_{AS} and K_{BS} are shared long-term keys between A and S, and B and S respectively, which are initially established by non-cryptographic, and out-

of-band techniques.

Protocol actions

1) In Message 1, A randomly chooses a new key k_{AB} for this session between A and B, and sends S the encryption of $\{B, k_{AB}\}_{K_{AS}}$ using the shared long-term key K_{AS} to indicate that A wants to establish a subsequent communication key with B, and k_{AB} is intended for A and B.

2) Upon receiving Message 1, S gets k_{AB} from the encryption of $\{B, k_{AB}\}_{K_{AS}}$ using the shared long-term key K_{AS}.

3) In Message 2, S notices B launching a new session between A and B by sending the encryption $\{A, k_{AB}\}_{K_{BS}}$.

4) Upon receiving Message 2, B gets k_{AB} from the encryption of $\{A, k_{AB}\}_{K_{BS}}$ using the shared long-term key K_{BS}.

Successful execution should establish a new session key k_{AB} between A and B.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the confidentiality and freshness assurances of the TVP k_{AB}. From Lemma 4.4, A has the association assurance of the k_{AB} with B from the explicitly mentioned principal name and the trusted freshness k_{AB}. From Lemma 4.4, A also has the association assurance of the k_{AB} with A since only A could have the new session key k_{AB} which is encrypted under the shared long-term key K_{AS}.

2) Upon receiving Message 1, from Lemma 4.2, B has the confidentiality assurance of the TVP k_{AB}.

3) In Message 2, B couldn't get any new assurance about this protocol since there is not a trusted freshness in the encryption $\{A, k_{AB}\}_{K_{BS}}$.

Upon termination of the protocol run, the security analysis result in Table 4.27 shows that: both A and B are not sure whether the opponent principal is present or not, and B is not sure whether k_{AB} is a fresh session key for A and B or not. From the absence of the security properties of an authentication protocol, various attacks could be directly constructed.

Table 4.27 Security analysis of the big mouth frog protocol

	A			B		
	B	S	k_{AB}	A	S	k_{AB}
Message 1			11AB			1?#
Message 2						
End of run			11AB			1?#

Example 4.30 (Attack on the big mouth frog protocol) From the absence of the A's liveness, the attacker may launch an attack without the presence of A, as shown in Fig. 4.32. Suppose the adversary I has recorded the message $\{A, \{B, k'_{AB}\}_{K_{AS}}\}$ in an old protocol run, then I can launch a new protocol

run by impersonating A.

$$\begin{array}{ll} \text{Message 1} & I(A) \rightarrow S : A, \{B, k'_{AB}\}_{K_{AS}} \\ \text{Message 2} & S \rightarrow B : \quad \{A, k'_{AB}\}_{K_{BS}} \end{array}$$

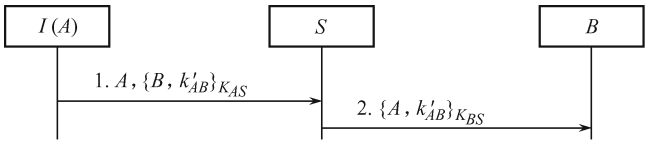


Fig. 4.32 An attack on the big mouth frog protocol.

Notation

$I(A)$ is an adversary I impersonating A.

Premise

k'_{AB} is a compromised session key between A and B. The adversary I has recorded $\{A, \{B, k_{AB}\}_{K_{AS}}\}$ in Message 1 of an old protocol run.

Protocol actions

1) In Message 1, the adversary I replays the recorded message $\{A, \{B, k_{AB}\}_{K_{AS}}\}$ to S to indicate that A wants to establish a subsequent communication key k'_{AB} with B, and k'_{AB} is intended for A and B.

2) Upon receiving Message 1, S gets k'_{AB} from the encryption of $\{B, k'_{AB}\}_{K_{AS}}$ using the shared long-term key K_{AS}.

3) In Message 2, S notices B launching a new session between A and B by sending the encryption $\{A, k'_{AB}\}_{K_{BS}}$.

Upon termination of the attack on the big mouth frog protocol, the adversary I causes B to have false beliefs: B has completed a successful protocol run with A, and is sharing a new session key k'_{AB} with A, but actually the key k'_{AB} is known by I. Furthermore, B concludes that subsequently messages could be encrypted using k'_{AB} and safely transmitted to A (actually known by I), whereas in fact, A knows nothing about the key establishment.

From the absence of the B's liveness, the attacker may also intercept the protocol run, and cause A to have false beliefs: A has completed a successful protocol run with A, and is sharing a new session key k'_{AB} with B, but actually B knows nothing about the key establishment.

Example 4.31 A electronic voting protocol is a voting protocol with deniable authentication for general elections[39, 40]. Fig. 4.33 shows that the voting center S collects each voter A_i's vote V_{A_i} and believes that the vote V_{A_i} is not repudiated via the use of the fresh nonce N_S; Every voter A_i is notified of the voting result RE and the voter A_i believes that the vote result RE is

noot repudiated via the use of the fresh nonce N_{A_i}.

Message 1 $S \rightarrow A_i : N_S$
Message 2 $A_i \rightarrow S : A_i, N_{A_i}, V_{A_i}, H(N_S, S_{A_i}, V_{A_i})$
Message 3 $S \rightarrow A_i : RE, H(N_{A_i}, S_{A_i}, RE)$

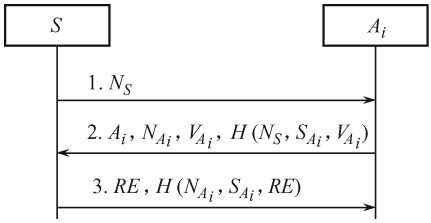


Fig. 4.33 An electronic voting protocol.

Notation

A_i is a voter who casts a vote in this electronic voting protocol, and S is a trusted third party who works as a voting center. S_{A_i} is the shared long-term key between the voter A_i and the voting center S. V_{A_i} is the vote which is sent by every voter A_i at the beginning of voting, and RE is the voting result collected by the voting center S.

Premise

The shared long-term S_{A_i} is initially established by non-cryptographic, and out-of-band techniques. H (nonce, key, data) is a keyed hash function with three input values. N_S is a nonce randomly chosen by S. N_{A_i} is a nonce randomly chosen by A_i.

Protocol actions

1) In Message 1, S launches a new electronic voting process by sending each voter A_i a randomly chosen fresh nonce N_S.

2) Upon receiving Message 1, every voter A_i gets N_S.

3) In Message 2, the voter A_i randomly chooses a nonce N_{A_i} for this voting, sends the identity of A_i, the nonce N_{A_i}, and A_i's vote V_{A_i} to S. A_i wants to show the freshness and the ownership of the vote V_{A_i} to S via a keyed hash function under the shared long-term S_{A_i}.

4) Upon receiving Message 2, S gets the vote V_{A_i} of the voter A_i and checks the validity of V_{A_i} via the ownership of S_{A_i} and the recalculation of $H(N_S, S_{A_i}, V_{A_i})$. Then, S summarizes all the votes of every voter and works out the voting result RE. Every voter A_i could not deny its sending vote V_{A_i} to S for the hash value $H(N_S, S_{A_i}, V_{A_i})$ including the trusted freshness N_S.

5) In Message 3, S announces the voting result RE to every voter A_i and wants to show the freshness and the ownership of the voting result RE to every A_i via a keyed hash function under the shared long-term key S_{A_i}.

6) Upon receiving Message 3, the voter A_i checks the validity of RE via the ownership of S_{A_i} and the recalculation of $H(N_{A_i}, S_{A_i}, RE)$. S could not deny its sending the voting result RE, which is summarized by S, to every voter A_i for the hash value $H(N_{A_i}, S_{A_i}, RE)$ includes the trusted fresh nonce N_{A_i} generated by A_i.

Successful execution should work out a new voting result RE, and neither the voter A_i nor the voting center S could deny the participation of this electronic voting.

Protocol security analysis

1) In Message 1, from Lemma 4.3, S has the freshness assurance of the TVP N_S, that is, N_S is the trusted freshness of S.

2) Upon receiving Message 1, B couldn't get any assurance about this protocol.

3) In Message 2, from Lemma 4.3, A_i has the freshness assurance of the TVP N_{A_i} and the vote V_{A_i}. From Lemma 4.2, A_i knows that the vote V_{A_i} is open.

4) Upon receiving Message 2, from Lemma 4.3, S has the freshness assurance of the vote V_{A_i}. From Lemma 4.1, S believes that it must be A_i who has generated the message $H(N_S, S_{A_i}, V_{A_i})$ using the shared long-term S_{A_i}. From Lemma 4.2, S knows that the vote V_{A_i} is open.

5) In Message 3, from Lemma 4.3, S has the freshness assurance of the voting result RE. From Lemma 4.2, S knows that the vote RE is open. From Lemma 4.4, S has the association assurance of the voting result RE with S.

6) Upon receiving Message 3, from Lemma 4.2, A_i knows that the voting result RE is open. From Lemma 4.3, A_i has the freshness assurance of the voting result RE. From Lemma 4.1 and Lemma 4.4, A_i has the liveness assurance of S and the association assurance of the vote V_{A_i} with S, since only S could generate $H(N_{A_i}, S_{A_i}, RE)$ using the shared long-term S_{A_i}, hence $H(N_{A_i}, S_{A_i}, RE)$ is associated with the protocol run with S, and S is present.

The security analysis result in Table 4.28 based on the trusted freshness shows that the electronic voting protocol has achieved the general elections with deniable authentication objectives as it intends to. The voter A_i could not deny A_i's vote V_{A_i} and the voting center S could not deny S's announcement of the voting result RE. However, the voter A_i can only believe that RE is a recently announced voting result by S, but A_i does not know whether the voting result RE is associated with each V_{A_i} or not.

Table 4.28 Security analysis of the electronic voting protocol

	A_i					S				
	S	N_S	N_{A_i}	V_{A_i}	RE	A_i	N_S	N_{A_i}	V_{A_i}	RE
Message 1							01#			
Message 2			01#	0?#		1		01#	01#	
Message 3	1				01S					01#
End of run	1				01S	1				01#

Example 4.32 A fair non-repudiation protocol is based on a trusted third party[39, 40]. In the protocol as shown in Fig. 4.34, the principal A sends message m to the principal B with the help of a trusted third party S. At the end of the protocol, A could deny A's sending of the message m, B could not deny B's receiving the message m.

$$\begin{aligned}
&\text{Message 1} \quad A \to B : f_1, B, L, \{m\}_{k'}, \{f_1, B, L, \{m\}_{k'}\}_{K_A^{-1}} \\
&\text{Message 2} \quad B \to A : f_2, A, L, \{m\}_{k'}, \{f_2, A, L, \{m\}_{k'}\}_{K_B^{-1}} \\
&\text{Message 3} \quad A \to S : f_3, B, L, k', \{f_3, B, L, k'\}_{K_A^{-1}} \\
&\text{Message 4} \quad S \to A : f_4, A, B, L, k', \{f_4, A, B, L, k'\}_{K_S^{-1}} \\
&\text{Message 5} \quad S \to B : f_4, A, B, L, k', \{f_4, A, B, L, k'\}_{K_S^{-1}}
\end{aligned}$$

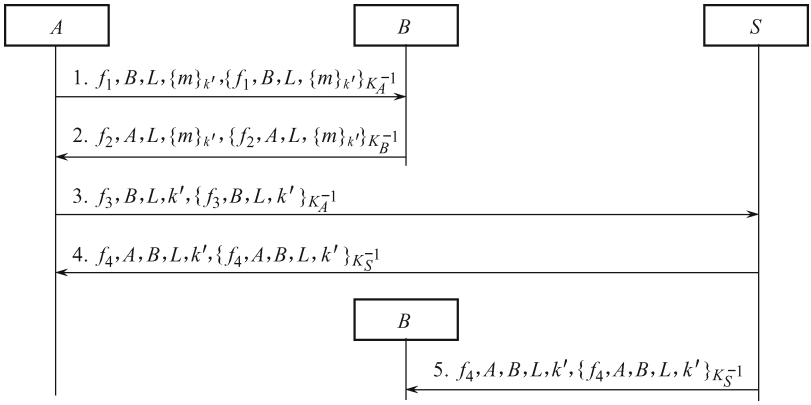


Fig. 4.34 A fair non-repudiation protocol.

Notation

A and B are two protocol principals, and S is a trusted third party. f_i is a message tag to indicate the message step, i.e., data type. L is the life of this protocol run. m is a sensitive data to be sent to B by A. k' is a randomly chosen temporary key for this protocol run by A. K_A and K_A^{-1}, K_B and K_B^{-1}, K_S and K_S^{-1} are the public, and private key pairs of the principals A, B and S respectively.

Premise

The public and private long-term key K_A and K_A^{-1}, K_B and K_B^{-1}, K_S and K_S^{-1} are initially established by non-cryptographic, and out-of-band techniques.

Protocol actions

1) In Message 1, A randomly chooses a temporary key k', and sends the message tag f_1, the opponent partner's identity B, and the life circle L

with the encryption $\{m\}_{k'}$ of the data m to B to launch a new protocol run between A and B. $\{f_1, B, L, \{m\}_{k'}\}_{K_A^{-1}}$ is signed by A's private key K_A^{-1} to indicate that Message 1 is from A.

2) Upon receiving Message 1, B verifies the signature $\{f_1, B, L, \{m\}_{k'}\}_{K_A^{-1}}$ using A's public-key K_A and gets the encryption $\{m\}_{k'}$ of the data m. But B can not confirm whether the received message is new generated by A or not, hence also B can't confirm whether A is present or not.

3) In Message 2, B sends the message tag f_2, the opponent partner's identity A, and the life circle L with the encryption $\{m\}_{k'}$ of the data m to A. $\{f_2, A, L, \{m\}_{k'}\}$ is signed by B's private key K_B^{-1} to indicate that Message 2 is from B.

4) Upon receiving Message 2, A checks the validity of $\{m\}_{k'}$ using B's public-key K_B. If it is right, then it must be B who has gotten $\{m\}_{k'}$ using A's public-key K_A and signed $\{f_2, A, L, \{m\}_{k'}\}$ using B's private key K_B^{-1}.

5) In Message 3, A sends the message tag f_3, the opponent partner's identity B, the life circle L and the temporary key k' to S. $\{f_3, B, L, k'\}$ is signed by A's private key K_A^{-1} to indicate that Message 3 is from A.

6) Upon receiving Message 3, S gets and checks k' from the signature $\{f_3, B, L, k'\}_{K_A^{-1}}$ using A's public-key K_A. Hence, A could not deny A's sending of k'.

7) In Message 4 and Message 5, S sends the message tag f_4, the protocol partners' identities A and B, the life circle L and the randomly chosen temporary key k' to A and B respectively. $\{f_4, A, B, L, k'\}$ is signed by the trusted third party S's private key K_S^{-1} to indicate that Message 4 is from S and the temporary key k' has been checked by S.

8) Upon receiving Message 4, A gets k' from the signature $\{f_4, A, B, L, k'\}_{K_S^{-1}}$ using S's public-key K_S and checks the validity of k'.

9) Upon receiving Message 5, B gets and checks k' from the signature $\{f_4, A, B, L, k'\}_{K_S^{-1}}$ using S's public-key K_S. Hence, B can get the data m from the encryption $\{m\}_{k'}$ via using k'.

Upon termination of the protocol run, from (b) and (f), A could not deny her sending of the message m, and from (d) and (i), B can get the message m. However, B could deny B's receiving of the message m, since no witness shows that B has gotten k', hence the message m.

Protocol security analysis

1) In Message 1, from Lemma 4.2, A has the confidentiality assurance of the temporary key k' and the data m. From Lemma 4.3, A has the freshness assurance of the temporary key k'. That is, k' is the trusted freshness of A.

2) Upon receiving Message 1, from Lemma 4.2 and Lemma 4.3, B has the confidentiality assurance of the temporary key k' and the data m.

3) Upon receiving Message 2, from Lemma 4.1, A has the liveness assurance of B. From Lemma 4.4, A has the association assurance of k' and m with B since Message 2 could not be a replay one, and only B could generate

$\{f_2, A, L, \{m\}_{k'}\}_{K_B^{-1}}$ using B's private key K_B^{-1}. From Lemma 4.4, A has the association assurance of k' and m with A, since the identity of A has been explicitly indicated in $\{f_2, A, L, \{m\}_{k'}\}_{K_B^{-1}}$, so k' and m are also associated with A.

4) In Message 3, from Lemma 4.2, A knows that k' is open.

5) After Message 3 is sent, from Lemma 4.2, B also knows that k' is open according to the protocol.

6) Upon receiving Message 4, from Lemma 4.1, A has the liveness assurance of S since Message 4 could not be a replay one, and only S could generate $\{f_4, A, B, L, k'\}_{K_S^{-1}}$ using its private key K_S^{-1}, hence S is present.

7) Upon receiving Message 5, B couldn't get any assurance about this protocol for B has not gotten any trusted freshness identifier.

The security analysis result in Table 4.29 based on the trusted freshness shows that from the legitimate participant A's point of view, B is present and the data m is open, fresh and associated with both A and B, and from the legitimate participant B's point of view, A is not present and the data m is open, and B could not achieve the association assurance of m with both A and B.

Table 4.29 Security analysis of the fair non-repudiation protocol

	A				B			
	B	S	k'	m	A	S	k'	m
Message 1			11#	11#			1?#	1?#
Message 2	1		11AB	11AB				
Message 3			01AB	01AB			0?#	0?#
Message 4		1						
Message 5								
End of run	1	1	01AB	01AB			0?#	0?#

Example 4.33 (Attack on the fair non-repudiation protocol) From the absence of A's liveness, the attacker may launch an attack without the presence of A, as shown in Fig. 4.35. Suppose the adversary I has recorded the message $\{f_1, B, L, \{m\}_{k''}, \{f_1, B, L, \{m\}_{k''}\}_{K_A^{-1}}\}$ in an old protocol run, then I can launch a new protocol run by impersonating A.

$$\begin{aligned}
&\text{Message 1} \quad I(A) \to B : f_1, B, L, \{m\}_{k''}, \{f_1, B, L, \{m\}_{k''}\}_{K_A^{-1}} \\
&\text{Message 2} \quad B \to I(A) : f_2, A, L, \{m\}_{k''}, \{f_2, A, L, \{m\}_{k''}\}_{K_B^{-1}} \\
&\text{Message 3} \quad I(A) \to S : f_3, B, L, k'', \{f_3, B, L, k''\}_{K_A^{-1}} \\
&\text{Message 4} \quad S \to I(A) : f_4, A, B, L, k'', \{f_4, A, B, L, k''\}_{K_S^{-1}} \\
&\text{Message 5} \quad S \to B : \quad f_4, A, B, L, k'', \{f_4, A, B, L, k''\}_{K_S^{-1}}
\end{aligned}$$

Notation

$I(A)$ is an adversary I impersonating A.

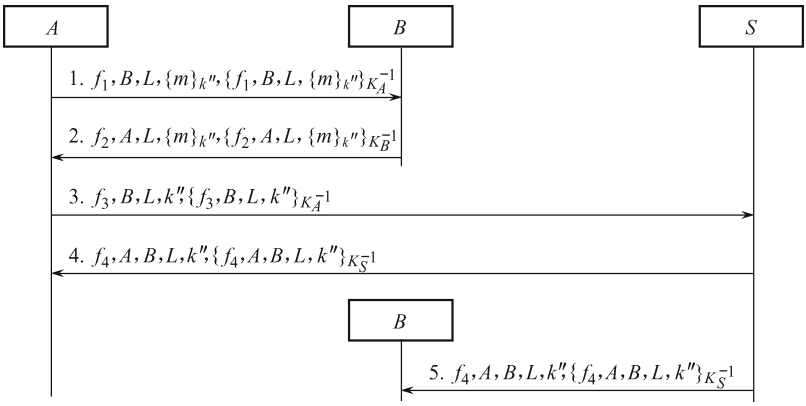


Fig. 4.35 An attack on the fair non-repudiation protocol.

Premise

The adversary I has recorded $\{f_1, B, L, \{m\}_{k''}, \{f_1, B, L, \{m\}_{k''}\}_{K_A^{-1}}\}$ in Message 1 and $\{f_3, B, L, k'', \{f_3, B, L, k''\}_{K_A^{-1}}\}$ in Message 3 of an old protocol run.

Protocol actions

1) In Message 1, the adversary I replays the recorded message $\{f_1, B, L, \{m\}_{k''}, \{f_1, B, L, \{m\}_{k''}\}_{K_A^{-1}}\}$ to B to indicate that A wants to send a sensitive data m to B.

2) Upon receiving Message 1, B verifies the signature $\{f_1, B, L, \{m\}_{k''}\}_{K_A^{-1}}$ and gets the encryption $\{m\}_{k''}$ of the data m.

3) In Message 2, B sends the message $\{f_2, A, L, \{m\}_{k''}\}_{K_B^{-1}}$ to A (actually it is the adversary I).

4) In Message 3, the adversary I replays the recorded message $\{f_3, B, L, k'', \{f_3, B, L, k''\}_{K_A^{-1}}\}$ to S.

5) Upon receiving Message 3, S gets and checks k'' from the signature $\{f_3, B, L, k''\}_{K_A^{-1}}$ using A's public-key K_A.

6) In Message 4 and Message 5, S signs $\{f_4, A, B, L, k''\}$ using S's private key K_S^{-1} and sends it to A and B respectively.

7) Upon receiving Message 4, the adversary I intercepts the message intended for A.

8) Upon receiving Message 5, B gets and checks k'' from the signature $\{f_4, A, B, L, k''\}_{K_S^{-1}}$ using S's public-key K_S.

Upon termination of this attack on the fair non-repudiation protocol, the adversary I causes B to have false beliefs: B has completed a successful protocol run with A, B has received the message m sent from A, and A could not deny the sending of m to B.

In real life, the freshness of non-repudiation record is important. For example, suppose a car agent has ordered 1000 cars from the General Motors

Corporation before, if the adversary can simply replay this order record without freshness guarantee, then the General Motors Corporation may check and accept this order, since the replayed order record may be regarded as non-repudiation assurance.

References

[1] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[2] Feige U, Fiat A, Shamir A (1987) Zero Knowledge Proofs of Identify. In: STOC'87 Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, 25 – 27 May 1987

[3] Miller SP, Neuman BC, Schiller JI, Saltzer JH (1987) Kerberos Authentication and Authorization System. Paper Presented at the Project Athena Technical Plan Section E.2.1. MIT, Boston

[4] CCITT (1987) CCITT Draft Recommendation X.509. The Directory-Authentication Framework (Version 7), New York

[5] Woo TYC, Lam SS (1992) Authentication for Distributed Systems. Computer 25(1): 39 – 52

[6] Kaufman C (1993) Distributed Authentication Security Service, RFC 1507. http://www.ietf.org/rfc/rfc1507.txt. Accessed 7 Sept 2010

[7] Okamoto T (1993) Provably Secure and Practical Identification Schemes and Corresponding Signature Scheme. In: CRYPTO'92 Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara 16 – 20 Aug 1992. Lecture Notes in Computer Science, vol 740, pp 31 – 53, Springer

[8] IBM Zurich Laboratory (1995) Internet Keyed Payments Protocol (IKP). http://www.zurich.ibm.com /Technology/Security/extern/ecommerce/spec. Accessed 30 June 2010

[9] Lowe G (1995) An Attack on the Needham-Schroeder Public-key Authentication Protocol. Information Processing Letters 56(3): 131 – 133

[10] Abadi M, Needham R (1996) Prudent Engineering Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering 22(1): 6 – 15

[11] Freier AO, Karlton P, Kocher PC (1996) The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt. Accessed 18 Nov 1996

[12] Clark J and Jacob J (1997) A Survey of Authentication Protocol Literature: Version 1.0. http://www.win.tue.nl/∼ecss/downloads/clarkjacob.pdf. Accessed Nov 2010

[13] SET. Secure Electronic Transaction. The SET Standard Specification. http://www.setco.org/set-specifications. Accessed May 1997

[14] Harkins D, Carrel D (1998) The Internet Key Exchange Protocol (IKE), RFC 2409. http://www.ietf.org/rfc/rfc2409.txt. Accessed 12 Dec 2010

[15] ANSI/IEEE Std 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Sept 1999

[16] Burrows M, Abadi M, Needham R (1990) A Logic of Authentication. ACM Transactions on Computer Systems 8(1): 18 – 36

[17] Bellare M, Rogaway P (1993) Entity Authentication and Key Distribution. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology

Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 773, pp 232 – 249, Springer

[18] Lowe G (1999) Towards a Completeness Result for Model Checking of Security Protocols. Journal of Computer Security 7(2 – 3): 89 – 146

[19] Canetti R, Krawczy H (2001) Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In: EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045, pp 453 – 474, Springer

[20] Blanchet B (2006) A Computationally Sound Mechanized Prover for Security Protocols. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Berkeley/Oakland, 21 – 24 May 2006

[21] Datta A, Derek A, Mitchell JC, Roy A (2007) Protocol Composition Logic (PCL). Electronic Notes in Theoretical Computer Science 172: 311 – 358.

[22] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[23] Dolev D, Yao AC (1983) On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2): 198 – 208

[24] Bellare M, Rogaway P (1993) Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In: CCS'93 Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, 3 – 5 Nov 1993

[25] Goldwasser S, Micali S (1984) Probabilistic Encryption. Journal of Computer and System Sciences 28(2): 270 – 299

[26] Dong L, Chen K, Zheng Y, Hong X (2008) The Guarantee of Authentication Protocol Security. Journal of Shanghai JiaoTong University 42(4): 518 – 522

[27] Otway D, Rees O (1987) Efficient and Timely Mutual Authentication. Operating Systems Review 21(1): 8 – 10

[28] Diffie W, Hellman ME (1976) New Directions in Cryptography. IEEE Transactions on Information Theory 22(6): 644 – 654.

[29] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[30] Matsumoto T, Takashima Y, Imai H (1986) On Seeking Smart Public-key Distribution Systems. Trans. IECE Japan 69(2): 99 – 106.

[31] Denning DE, Sacco GM (1981) Timestamps in Key Distribution Protocols. Communication of the ACM 24(8): 533 – 536

[32] Woo TYC, Lam SS (1994) A Lesson on Authentication Protocol Design. ACM Operating Systems Review 28(3): 24 – 37

[33] Neuman BC, Stubblebine SG (1993) A Note on the Use of Timestamps as Nonces. Operating Systems Review 27(2): 10 – 14

[34] Bird R, Gopal I, Herzberg A, Janson P, Kutten S, Molva R, Yung M (1995) The KryptoKnight Family of Light-weight Protocols for Authentication and Key Distribution. IEEE/ACM Transactions on Networking 3(1): 31 – 41

[35] Yahalom. http://www.lsv.ens-cachan.fr/spore/yahalom.pdf. Accessed 12 May 2011

[36] Tatebayashi M, Matsuzaki N, Newman D (1989) Key Distribution Protocol for Digital Mobile Communication Systems. In: CRYPTO'93 Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 20 – 24 Aug 1989. Lecture Notes in Computer Science, vol 435, pp 324 – 334, Springer

[37] Lowe G, Roscoe B (1997) Using CSP to Detect Errors in the TMN Protocol. IEEE Transactions on Software Engineering 23(10): 659 – 669

[38] Tanenbaum AS (2001) Computer Networks, 3rd edn. Prentice Hall, New Jersey

[39] Zhou J, Gollmann D (1996) A Fair Non-repudiation Protocol. In: Proceedings of 1996 IEEE Symposium on Security and Privacy, Oakland, 6 – 8 May 1996

[40] Zhou J (1996) Non-repudiation. PhD Dissertation, University of London

5 Security Analysis of Real World Protocols

Abstract Several de facto or industrial standards are widely used in many real world applications are discussed and analyzed via the trusted freshness approach. The typical cryptographic protocols include the Secure Socket Layer Protocol (SSL) and its variant, Transport Layer Security Protocol (TLS), the Internet Key Exchange Protocol (IKE) and the Kerberos Authentication Protocol. From the discussion and the security analysis of these protocols, we will see that it is very challenging to achieve strong security properties of the cryptographic protocols fit for application.

Our study of cryptographic protocols in the preceding chapters has focused on the academic protocols, while in this chapter, we will touch some widely used real world cryptographic protocols and analyze them using the trusted freshness method. This will help the readers to understand the trusted freshness method deeply and evaluate the security strength of a cryptographic protocol they may use in practice.

The Internet is an enormous open network of computers and devices called "nodes". To deal with the complicated network well, the ISO (the International Organization for Standardization) presents the Open System Interconnection Reference Model (OSI Reference Model or OSI Model) which is an abstract description for layered communications and computer network protocol design. In its most basic form, it divides network architecture into seven layers which, from top to bottom, are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layers, and they are also called the seventh layer, the sixth layer, and so on and so forth. A layer is a collection of conceptually similar functions that provide services to the layer above it and receive service from the layer below it.

Figure 5.1 illustrates a simplified ISO Open System Interconnection (OSI) architecture considering placement of key distribution protocols[1–12].

Internet is an open network environment, and each node in the network trusts each other from the Internet's original design intention, hence insecure systems are already in wide use. To keep backward compatibility, secure solutions should be added in with the least interruption to the insecure systems which are already in operation[13]. E.g. the SSL avoids modifying "TCP

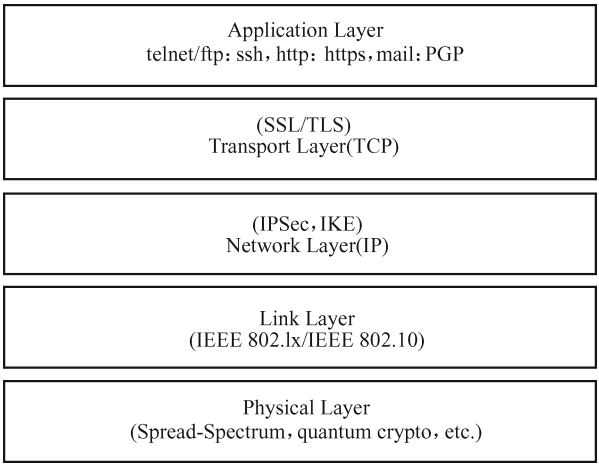


Fig. 5.1 The ISO Open System Interconnection.

stack" and requires minimum changes to the application, mostly used to authenticate servers; IPSec is transparent to the application, but it requires modification of the network stack and establishes a secure channel between nodes.

In this chapter, we shall introduce and discuss several cryptographic protocols which are de facto or industrial standards, and they are already widely used in many real world applications. The real world protocols we shall study include the Secure Socket Layer Protocol (SSL)[2], and its variant, Transport Layer Security Protocol (TLS)[3]; the Internet Key Exchange Protocol (IKE)[4, 5] and the Kerberos Authentication Protocol[6, 7]. From the discussion and the security analysis of these protocols, we will see that it is very challenging to achieve strong security properties of the cryptographic protocols fit for application.

5.1 Secure Socket Layer and Transport Layer Security

Secure Socket Layer (SSL), and its variant, Transport Layer Security (TLS), are the de facto standards used to end-to-end encrypt, and they are mainly for WorldWideWeb (Web for short) security[2, 3]. This includes specifically credit card purchases and bank sites, but it may also be used on any site requesting a password or dealing with personal information. SSL and TLS use public-key encryption.

The most recent draft of the SSL 3.0 specification was published in November 1996 by Netscape and SSL 3.0 was the basis for the TLS 1.0 (RFC 2246) specification published by the Internet Engineering Task force (IETF) in 1999. The IETF made some small changes and clarifications and published RFC4346 in 2006 detailing TLS 1.1. The most recent draft of the TLS 1.2

(RFC 5246) specification was published in August 2008 by IETF.

5.1.1 SSL and TLS overview

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. Except cryptographic security, the goals also include interoperability, extensibility, and relative efficiency. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The SSL Handshake Protocol can be considered as a stateful process running on the client and server machines. A stateful connection is called a "session", and a session can be renegotiated.

The SSL protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., Transmission Control Protocol, that is TCP protocol) is the SSL Record Protocol. The SSL Record Protocol provides secure encapsulation of the communication channel for use by higher layer application protocols. The higher level protocols include Handshake, Change Cipher Spec, and Alert protocols as well as application data. The SSL handshake protocol is a key exchange protocol which initializes and synchronizes cryptographic state at the two endpoints. After the key-exchange protocol completes, sensitive application data can be sent via the SSL record layer. One advantage of SSL is that a higher level protocol can layer on top of the SSL Protocol transparently. The technique used to encrypt and verify the integrity of SSL records is specified by the currently active Cipher Spec. A typical example would be to encrypt data using DES and generate authentication codes using MD5.

SSL/TLS has 4 underlying protocols: Handshake, Record, Change Cipher Spec, and Alert. This is laid out as:

8 bit	8 bit	8 bit	16 bit	16384 bytes
Type	Major version	Minor Version	Record Length	Record Data

In decimal, the types are as follows:
20 Change Cipher Spec
21 Alert
22 Handshake
23 Application (data)

The version would be 3 and then 0 for SSL 3.0. Since TLS is a "minor modification to the SSL 3.0 protocol," TLS 1.0 is defined as SSL major version 3, minor version 1, TLS 1.1 is 3 and then 2, and the upcoming TLS 1.2 will be major version 3, then minor version 3.

The record length is written in terms of bytes and can not exceed 2^{14} (16, 384). Compression allows for the length to be extended by up to 1024 bytes,

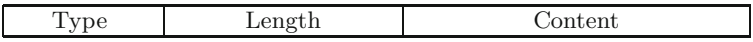
to a new max of 17,408 bytes in the TLS compressed length field.

SSL connections begin with a 4-way handshake. The keys for symmetric encryption and for HMAC are generated uniquely for each session connection and are based on a secret negotiated by the SSL Handshake Protocol.

Alert messages with a level of fatal result in the immediate termination of the connection. In this case, other connections corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections.

5.1.2 The SSL handshake protocol

The SSL Handshake Protocol is one of the defined higher level clients of the SSL Record Protocol. The SSL Handshake Protocol allows the server and client to authenticate each other and to negotiate a encryption algorithm and cryptographic keys for symmetric encryption and for HMAC uniquely for each session connection, and thereby to establish a secure session connection with the SSL Record Protocol to process secure communications with higher level application protocols. The handshake protocol structure is:



The allowed values for type are indicated in Table 5.1.

Table 5.1 Allowed values for type in SSL handshake protocol

Type Value	Type	Remark
0	HelloRequest	
1	ClientHello	
2	ServerHello	
11	Certificate	Optional
12	ServerKeyExchange	Optional
13	CertificateRequest	Optional
14	ServerHelloDone	
15	CertificateVerify	
16	ClientKeyExchange	Optional
20	Finished	

The handshake protocol messages are presented in the order in which they must be sent; sending handshake messages in an unexpected order results in a fatal error.

The data handshake process performs the following steps, as shown in Fig. 5.2.

Message 1 $C \rightarrow S$: Client Hello

Message 2 $S \rightarrow C$: ServerHello
ServerCertificate*
ServerKeyExchange*
CertificateRequest*
ServerHelloDone
Message 3 $C \rightarrow S$: ClientCertificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished
Message 4 $S \rightarrow C$: [ChangeCipherSpec]
Finished

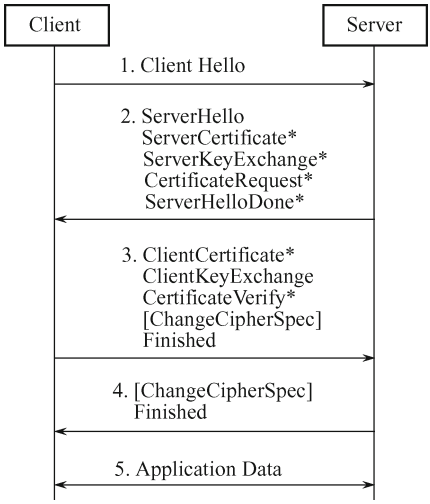


Fig. 5.2 Message flow for a full handshake of SSL.

Notation

C denotes a client (the client-side web browser), S denotes the web server, and * indicates optional or situation-dependent messages that are not always sent.

Premise

ClientCertificate (ServerCertificate) is a certification of C's (or S's) identity and corresponding public-key K_C (or K_S) signed by a trusted certification authority center CA.

Protocol actions

1) In Message 1 (*ClientHello message*:)

The client starts the session connection by sending a message to which the server must respond with a ServerHello message, or else the connection will fail. The client may resume an existing session. The *ClientHello* message will have the following data:

— The protocol version is in the client hello which is for backward compatibility use.
— A 32-bit Unix format timestamp and a 28-byte random number ClientHello.random are generated by the client.
— The session identifier: When the client wishes to start a new session connection, this field should be empty. The client may specify a session identifier of a current or previous session. Doing this allows for multiple secure connections without going through the entire handshake process each time, although both Hello, the Change Cipher Spec, and both Finished messages must still be exchanged and be valid.
— The cipher suite, a list of the cryptographic options supported in the client side machine, sorted with the client's first preference first. Each cipher suite defines the algorithm for key exchange, the bulk encryption algorithm with secret key and length, and the message authentication code (MAC). A wide range of public-key and symmetric cryptographic algorithms, digital signature schemes, MAC schemes and hash functions can be proposed by the client.

A cipher suite identifies a Cipher Spec. These structures are part of the SSL session state. The Cipher Spec includes:

```
enum {stream, block} CipherType;
enum {true, false} IsExportable;
enum {null, rc4, rc2, des, 3des, des40, fortezza}
      BulkCipherAlgorithm;
enum {null, md5, sha} MACAlgorithm;
struct {
      BulkCipherAlgorithm bulk_cipher_algorithm;
      MACAlgorithm mac_algorithm;
      CipherType cipher_type;
      IsExportable is_exportable;
      uint8 hash_size;
      uint8 key_material;
      uint8 IV_size;
     } CipherSpec;
```

Here, unit8 is an unsigned byte.

— The compression method supported in the client side machine.

2) In Message 2 (*ServerHello message*:)

The server responds to Client Hello message with the ServerHello message. The *ServerHello* message will have the following data:

— The version number being used: the lowering of the server's highest supported version and the version in the client hello.

— A 32-bit Unix format timestamp and a 28-byte random number ServerHello.random are generated by the server.
— The session identifier: if the session ID is recognized, then a short handshake is used and the following fields are filled in with the values from the previous connection. Otherwise, the ServerHello generates a new session ID, uses this new value in this field, and caches the session ID in its local memory. The server may return an empty session ID to indicate that the session will not be cached and therefore cannot be resumed.
— The cipher suite chosen by the server, where the server selects a single scheme for each necessary cryptographic operation, informs the client in this field.
— The compression method chosen by the server.

If the server can not find an acceptable cipher suite and compression method, it will respond with a handshake failure alert.

(1) ServerCertificate message: Unless the key exchange method is anonymous, if the server is to be authenticated (which is generally the case), the server will send out a certificate immediately after sending the ServerHello. The certificate is generally an X.509 v3 certificate public-key and unless otherwise specified uses the same key exchange method and signing algorithm previously decided on. An X.509 certificate contains sufficient information about the name and the public-key of the certificate owner and that about the issuing certification authority. Sending a list of certificates permits the client to choose one with the public-key algorithm supported in the client's machine. That is, certificates from all the up line servers are necessary to get to the one that the client trusts must be included. The order of these should be so that each certificate validates the one before it.

(2) ServerKeyExchange message: If the ServerCertificate does not contain enough data for a pre-master secret, then a ServerKeyExchange is sent with either an RSA public, or a Diffie-Hellman public-key (This is the case for DHE_DSS, DHE_RSA, and DH_anon; but not for RSA, DH_DSS, and DH_RSA key exchange methods). ServerKeyExchange contains the server's public-key material matching the certificate list in ServerCertificate. The material for Diffie-Hellman key agreement will be included here which is the tuple (p, g, g^y) where p is a prime modulus, g is a generator modulo p of a large group and y is an integer cached in the server's local memory.

(3) CertificateRequest message: If it is appropriate, the server may request a certificate from the client with a CertificateRequest. This would immediately follow the ServerCertificate, or present the ServerKeyExchange. The CertificateRequest would specify the types of certificates the server will accept and the Certificate Authorities the server trusts.

(4) ServerHelloDone message: The ServerHelloDone indicates to the client that server is done sending data and the client should now verify the certificates and whatnot it has received.

3) In Message 3 (*ClientCertificate message*:)

After receiving the ServerHelloDone the client would respond with a message identical in format to the ServerCertificate if a CertificateRequest was received before.

(1) ClientKeyExchange message: If RSA is used, the Client Key Exchange message includes an encrypted pre-master secret which consists of a 48-bit number that is encrypted with the server's public-key.

If Diffie-Hellman is used, but not Fixed Diffie-Hellman, then the public-key parameters are sent here.

(2) CertificateVerify message: If the client sent a certificate, then it would send a CertificateVerify message at this point, in most cases. This would include a signature in the same format as defined for the ServerKeyMessage as well as an MD5 sum of all of the previous messages and a SHA hash of all of the previous messages.

(3) ChangeCipherSpec message: The Client sends the ChangeCipherSpec message indicating that all future traffic will be computed with the master secret. The random numbers and the pre-master secret are used by both systems in a pseudorandom function to calculate the master secret.

The change cipher spec protocol is a single byte that will always have a value of 1. It is encrypted and compressed under the current cipher (the pre-master secret) and with compression method.

(4) Finished message: Up to now, the client and server have negotiated the shared secret information known only to themselves. This value is a 48-byte quantity called the master secret.

master_secret =

MD5(pre_master_secret + SHA('A' + pre_master_secret
+ ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('BB' + pre_master_secret
+ ClientHello.random + ServerHello.random)) +
MD5(pre_master_secret + SHA('CCC' + pre_master_secret
+ ClientHello.random + ServerHello.random)).

The client now sends the Finished message. This consists of the master secret, the finished label, an MD5 of all previous messages and an SHA of all previous messages. All of this is encrypted with the master secret. If the server can read all of this, then the server knows that the key generation was successful. The Finished message is the first protected with the just-negotiated algorithms, keys, and secrets.

4) In Message 4 (*ChangeCipherSpec & Finished message*:)

The server responds with its own ChangeCipherSpec and Finished messages which verify to the client that the key generation was successful.

If any warning or fatal errors occur, an alert is sent. Alerts consist of a byte that defines whether it's a warning (1) or a fatal (2) alert, and a byte

that indicates the specific alert. The possible values for alerts are indicated in Table 5.2.

Table 5.2 Alerts in SSL

Fatal alerts	Not fatal alerts
unexpected_message (10)	close_notify (0)
bad_record_mac (20)	no_certificate_RESERVED (41) – this is SSL 3.0 only
decryption_failed (21)	bad_certificate (42)
record_overflow (22)	unsupported_certificate (43)
decompression_failure (30)	certificate_revoked (44)
handshake_failure (40)	certificate_expired (45)
illegal_parameter (47)	certificate_unknown (46)
unknown_ca (48)	decrypt_error (51)
access_denied (49)	no_renegotiation (100)
decode_error (50)	
export_restriction_RESERVED (60)	
protocol_version (70)	
insufficient_security (71)	
internal_error (80)	
user_canceled (90)	

Application data: The master secret is used to generate keys and secrets for encryption and MAC computations. To generate the key material, compute key_block until enough output has been generated, where

key_block =

MD5(master_secret + SHA('A' + master_secret
+ ServerHello.random + ClientHello.random)) +
MD5(master_secret + SHA('BB' + master_secret
+ ServerHello.random + ClientHello.random)) +
MD5(master_secret + SHA('CCC' + master_secret
+ ServerHello.random + ClientHello.random)) + [...].

Then the key_block is partitioned as follows.

client_write_MAC_secret[CipherSpec.hash_size]
server_write_MAC_secret[CipherSpec.hash_size]
client_write_key[CipherSpec.key_material]
server_write_key[CipherSpec.key_material]
client_write_IV[CipherSpec.IV_size] /* non-export ciphers */
server_write_IV[CipherSpec.IV_size] /* non-export ciphers */

Any extra key_block material is discarded.

Now that the keys and secrets are computed, data may be sent encapsulated inside record protocol. This data will be encrypted and compressed in the agreed upon methods and can be reliably read by the other end but not

likely anyone in-between.

5.1.3 Security analysis of SSL based on trusted freshness

SSL is helpful for enhancing the security of communications, however it is not as secure as it intends to. In this subsection, we give the security analysis of SSL based on trusted freshness, and some attacks are given from the absence of the protocol security properties. Moreover, attacks related to TLS renegotiation implementation are also briefly introduced.

5.1.3.1 Security analysis of SSL negotiation based on trusted freshness

Example 5.1 (A full SSL handshake protocol with both side certificates) When a new session begins, the CipherSpec encryption, hash, and compression algorithms are initialized to null. Figure 5.3 illustrates the message exchanges in SSL handshake protocol related to authentication and key establishment with the certification verifying both server side and client side.

Message 1 $C \rightarrow S : Ver_C, T_C, N_C, \ NULL$

Message 2 $S \rightarrow C : Ver_S, T_S, N_S, SID_S, Cert_S$

Message 3 $C \rightarrow S : \{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}}$

Message 4 $S \rightarrow C : \{S, N_S, N_C\}_{k_{CS}}$

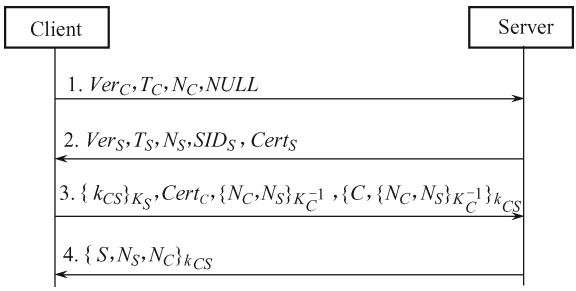


Fig. 5.3 Message in a full handshake of SSL with certifications of both sides.

Notation

C denotes a client (the client-side web browser), S denotes the web server. Ver_C is the protocol version in the client hello, and Ver_S is the version number being used (the lowering of the server's highest supported version and the version in the client hello). T_C and T_S are timestamps generated by C and S referring to an absolute time, where clocks are not required to be set correctly by the basic SSL Protocol, but higher level or application protocols may define additional requirements. N_C and N_S are nonces randomly chosen by C and S respectively. k_{CS} is a new session key between C and S to

be established in this authentication protocol (Note: we do not distinguish pre_master_secret, master_secret and other keys and secrets for encryption and MAC computations since they do not effect the security analysis of SSL). SID_S is a new session ID generated and cached by the Server. $Cert_C$ ($Cert_S$) is a certificate of C (or S) and corresponding public-key K_C (or K_S) signed by a trusted certificate authority CA. K_S and K_S^{-1} are public-key and private key of S, while K_C and K_C^{-1} are public-key and private key of C.

Premise

Both C and S know the public-key of the trusted certificate authority CA to get K_C and K_S. Each principal knows the key pair of himself, that is, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S.

Protocol actions

1) In Message 1, the client C starts the negotiation by sending the client SSL version Ver_C, the timestamp T_C, a randomly chosen new nonce N_C by C for this protocol run, a $NULL$ session identifier and the intended cryptographic algorithm for key exchange, the bulk encryption algorithm with secret key and length, and MAC (the chosen cipher suite does not effect the security analysis of SSL, hence omitted.).

2) Upon receiving Message 1, since the session ID is $NULL$, a full handshake is launched.

3) In Message 2, S generates a new session ID SID_S, and sends it to C with the version number Ver_S being used, the timestamp T_S, the randomly chosen new nonce N_S by S, and the certificate of S containing the name and the public-key.

4) Upon receiving Message 2, C randomly chooses a new session key k_{CS} for this protocol run, then encrypts it under K_S and sends it to C; C encrypts the identity of himself, the randomly chosen nonce N_C and N_S, and sends $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$ to S.

5) Upon receiving Message 3, S gets the new session key k_{CS} using S's private key K_S^{-1}, then decrypts $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$ using k_{CS} also K_C and verifies the correction of N_C and N_S.

6) In Message 4, S encrypts the identity of S, the randomly chosen nonce N_C and N_S, and sends $\{S, N_S, N_C\}_{k_{CS}}$ to C.

7) Upon receiving Message 4, C decrypts $\{S, N_S, N_C\}_{k_{CS}}$ using k_{CS} and verifies the correction of N_C and N_S.

Successful execution should convince C and S that k_{CS} is a new session key between C and S. Actually, this protocol has not achieved the key exchange and authentication security objects as it intends to.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_C, and C also believes that N_C is open.

2) Upon receiving Message 1, from Lemma 4.2, S believes that N_C is open.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, S has the freshness assurance of the randomly chosen nonce N_S, and S also believes that N_S is open.

4) Upon receiving Message 2, from Lemma 4.2, C believes that N_S is open.

5) In Message 3, from Lemma 4.2 and Lemma 4.3, C has the confidentiality and the freshness assurances of the new session key k_{CS}.

6) Upon receiving Message 3, from Lemma 4.3, S has the freshness assurance of the new session key k_{CS}. From Lemma 4.4, S has the association assurance of k_{CS} with C, since only C could sign $\{N_C, N_S\}_{K_C^{-1}}$ using its private key, and the identity of C is explicitly indicated in Message 3. Upon receiving $\{N_C, N_S\}_{K_C^{-1}}$, from Lemma 4.1, S has the liveness assurance of C based on the trusted freshness N_S.

7) Upon receiving Message 4, from Lemma 4.1, C has the liveness assurance of S. From Lemma 4.3, C has the freshness assurance of N_S. From Lemma 4.4, S has the association assurances of N_C, N_S and k_{CS} with S, since only S could get k_{CS} from the encryption $\{k_{CS}\}_{K_S}$ using its private key, and the identity of S is explicitly indicated in Message 4.

Upon termination of the protocol run, as indicated in Table 5.3, C believes that S is present, and the new session key k_{CS} is confidential, fresh, and associated with S, while S believes that C is present, and the new session key k_{CS} is confidential, fresh, and associated with C.

Table 5.3 Security analysis of the full handshake of SSL protocol

	C				S			
	S	N_C	N_S	k_{CS}	C	N_C	N_S	k_{CS}
Message 1		01#				0?#		
Message 2			0?#				01#	
Message 3				11#	1		01C	11C
Message 4	1	01S	01S	11S				
End of run	1			11S	1			11C

Example 5.2 (Attack-1 on a full SSL handshake protocol with both side certificates) From the absence of the association of k_{CS} with S in the point of view of S, we can construct an attack as shown in Fig. 5.4.

$$
\begin{aligned}
&\text{Message 1 } C \to I: && Ver_C, T_C, N_C, NULL\\
&\text{Message 1}' \; I(C) \to S: && Ver_C, T_C, N_C, NULL\\
&\text{Message 2}' \; S \to I(C): && Ver_S, T_S, N_S, SID_S, Cert_S\\
&\text{Message 2 } I \to C: && Ver_S, T_S, N_S, SID_S, Cert_I\\
&\text{Message 3 } C \to I: && \{k_{CS}\}_{K_I}, Cert_C, \{N_C, N_S\}_{K_C^{-1}},\\
& && \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}}
\end{aligned}
$$

$$\begin{aligned}
&\text{Message } 3' \quad I(C) \to S : \{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \\
&\qquad\qquad\qquad\qquad \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}} \\
&\text{Message } 4 \quad I \to C : \quad \{I, N_S, N_C\}_{k_{CS}} \\
&\text{Message } 4' \quad S \to I(C) : \{S, N_S, N_C\}_{k_{CS}}
\end{aligned}$$

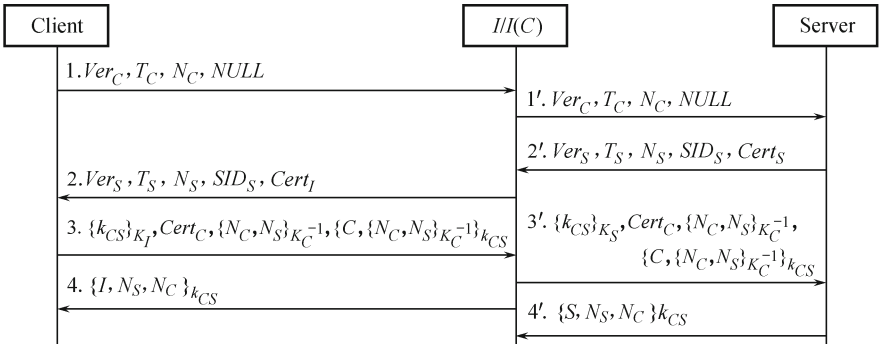


Fig. 5.4 An attack on a full handshake of SSL with certifications of both sides.

Notation

I denotes the adversary, and $I(C)$ is an adversary I impersonating C. $Cert_I$ is a certificate of I and corresponding public-key K_I signed by CA. K_I and K_I^{-1} are public-key and private key of I. Other notations are the same as the original SSL handshake protocol.

Premise

Each principal knows the public-key of the trusted certification authority center CA to get K_C, K_S and K_I. Each principal knows the key pair of himself, that is, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S, K_I and K_I^{-1} for I.

Protocol actions

1) In Message 1, the client C starts a protocol run with I. In Message $1'$, the adversary $I(C)$ replays the Message 1 to S to start a fake protocol run between C and S by impersonating C.

2) Upon receiving Message $1'$, S makes response to $I(C)$ with Message $2'$ including the certificate of S. I substitutes $Cert_S$ with $Cert_I$ in Message $2'$, then forwards Message 2 $\{\text{Ver}_S, T_S, N_S, SID_S, Cert_I\}$ to C.

3) Upon receiving Message 2, C randomly chooses a new session key k_{CS} for this protocol run between C and I. In Message 3, C generates $\{N_C, N_S\}_{K_C^{-1}}$ using C's private key K_C^{-1} to show that it is really C who has sent this message $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$.

4) Upon receiving Message 3, I gets the new session key k_{CS} using I's private key K_I^{-1}. Then, I encrypts k_{CS} with S's public-key K_S, generates Message $3'$ $\{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}$, and $\{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}$, then

forwards it to S.

5) Upon receiving Message $3'$, S gets k_{CS} and verifies $\{N_C, N_S\}_{K_C^{-1}}$, then S believes that it must be C who is sharing the new session key k_{CS} with S. At the same time, I will complete his protocol run with C normally.

Upon termination of the attack on the full SSL handshake protocol with both side certificates, the adversary I causes S to have false beliefs: S has completed a successful protocol run with C, and is sharing a new session key k_{CS} with C, whereas in fact, C knows nothing about the key establishment with S, and actually C shares the key k_{CS} with I. Furthermore, S concludes that subsequently messages could be encrypted using k_{CS} and safely transmitted to C (actually known by I).

Example 5.3 (Attack-2 on a full SSL handshake protocol with both side certificates) From the absence of the association of k_{CS} with C in the point of view of C, we can construct an attack as shown in Fig. 5.5.

$$
\begin{array}{lll}
\text{Message } 1 & C \to I(S): & Ver_C, T_C, N_C, NULL \\
\text{Message } 1' & I(C) \to S: & Ver_C, T_C, N_C, NULL \\
\text{Message } 2' & S \to I(C): & Ver_S, T_S, N_S, SID_S, Cert_S \\
\text{Message } 2 & I(S) \to C: & Ver_S, T_S, N_S, SID_S, Cert_I \\
\text{Message } 3 & C \to I(S): & \{k_{CS}\}_{K_I}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \\
 & & \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}} \\
\text{Message } 3' & I(C) \to S: & \{k_{CS}\}_{K_S}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \\
 & & \left\{C, \{N_C, N_S\}_{K_C^{-1}}\right\}_{k_{CS}} \\
\text{Message } 4 & I(S) \to C: & \{S, N_S, N_C\}_{k_{CS}} \\
\text{Message } 4' & S \to I(C): & \{S, N_S, N_C\}_{k_{CS}}
\end{array}
$$

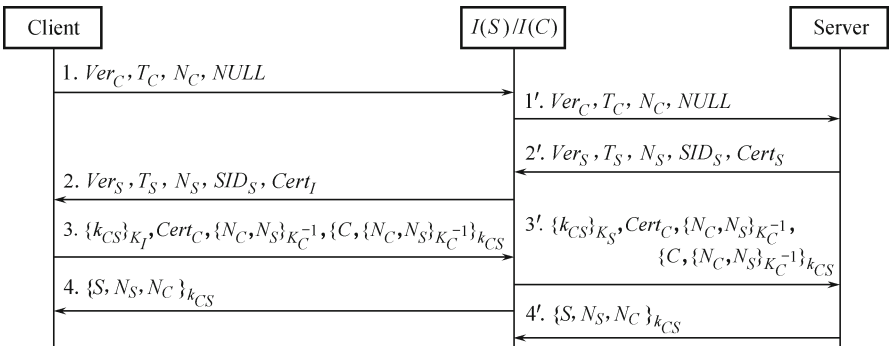


Fig. 5.5 Another attack on a full handshake of SSL with certifications of both sides.

Notation

I denotes the adversary, and $I(C)/I(S)$ denotes the adversary I impersonating C/S respectively. $Cert_I$ is a certificate of I and K_I signed by CA. K_I and K_I^{-1} are public-key and private key of I. Other notations are the same as the original SSL handshake protocol.

Premise

Each principal knows the public-key of the trusted certificate authority CA to get K_C, K_S and K_I. Each principal knows the key pair of himself, that is, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S, K_I and K_I^{-1} for I.

Protocol actions

1) In Message 1, the client C starts a protocol run with S. I intercepts Message 1, then forwards it to S as Message $1'$.

2) Upon receiving Message $1'$, S makes response to $I(C)$ with Message $2'$ including the certificate of S. I substitutes $Cert_S$ with $Cert_I$ in Message $2'$, then forwards Message 2 $\{Ver_S, T_S, N_S, SID_S, Cert_I\}$ to C.

3) Upon receiving Message 2, C randomly chooses a new session key k_{CS} for this protocol run between C and S (Actually, it is the adversary I impersonating S). In Message 3, C encrypts k_{CS} with S's public-key K_S (Actually, the public-key is deduced from $Cert_I$ in Message $2'$, hence it is the adversary I's public-key K_I), and generates Message 3 $\{\{k_{CS}\}_{K_I}, Cert_C, \{N_C, N_S\}_{K_C^{-1}}, \{C, \{N_C, N_S\}_{K_C^{-1}}\}_{k_{CS}}\}$.

4) Upon receiving Message 3, I gets the new session key k_{CS} using I's private key K_I^{-1}. I encrypts k_{CS} with S's public-key K_S in Message $3'$, then forwards Message $3'$ to S.

5) In Message 4, I will complete his protocol run with C normally by impersonating S.

6) Upon receiving Message $4'$, S will complete his protocol run with C normally (Actually, it is the adversary I impersonating C).

Upon termination of the attack on the full SSL handshake protocol with both side certificates, the adversary I causes both sides to have false beliefs: each side has completed a successful protocol run with its opponent, and is sharing a new session key k_{CS} with the opponent, whereas in fact, the shared key k_{CS} between C and S is also known by I.

In an execution of the SSL Handshake Protocol, the client may be chosen to be anonymous and so is not authenticated to the server. As a result, the server unilaterally authenticates itself to the client. The output from the execution is a unilaterally authenticated channel from the server to the client. This is a typical example of using the SSL Protocol in a Web-based electronic commerce application, for example, buying a book from an online bookseller. The output channel assures the client that only the authenticated server will receive its instructions on book purchase which may include confidential information such as its user's bankcard details, the book title, and the delivery

address[13].

5.1.3.2 Attacks related to renegotiation

This protocol can be executed with all the optional messages and the ClientKeyExchange message omitted. This is the case when the client wants to resume an existing session. Session resumption can save the server time and CPU by obviating the need to do a full basic SSL negotiation. The ostensible reason for renegotiation is to allow either end to decide that it would like to refresh its cryptographic keys, increase the level of authentication, increase the strength of the cipher suite in use, and so forth. If the session ID in the ClientHello message is non-empty, the server will look in its session cache for a match. If a match is found and the server is willing to establish the new connection using the specified session state, the server will respond with the same session ID as supplied by the client. This indicates a resumed session and dictates that the parties must proceed directly to the finished messages, omitting all the optional messages and the ClientKeyExchange message. The renegotiation SSL handshake protocol structure is illustrated in Fig. 5.6.

$$
\begin{aligned}
&\text{Message 1} \quad C \rightarrow S: \ Ver_C, T_C, N_C, SID_{Existing} \\
&\text{Message 2} \quad S \rightarrow C: \ Ver_S, T_S, N_S, SID_{Existing} \\
&\text{Message 3} \quad C \rightarrow S: \{k_{CS}\}_{K_S}, \{C, N_C, N_S\}_{k_{CS}} \\
&\text{Message 4} \quad S \rightarrow C: \{S, N_S, N_C\}_{k_{CS}}
\end{aligned}
$$

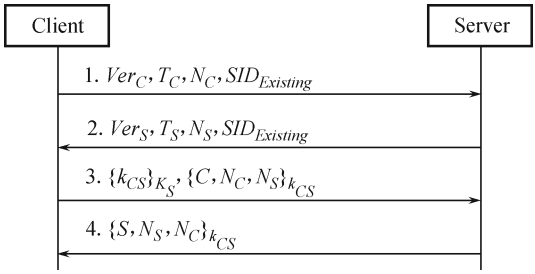


Fig. 5.6 Message exchanges in a renegotiation handshake of SSL.

Notation

$SID_{Existing}$ is the ID of an existing session stored in the server's session cache. Other notations are the same as the original SSL handshake protocol.

Premise

$SID_{Existing}$ is stored in the server's session cache and the server is willing to establish the new connection using the specified session state.

The premises are the same as the original SSL handshake protocol.

Protocol actions

1) In Message 1, if the client C wants to resume an existing session, then the client sends a ClientHello message using the Session ID of the session to be resumed.

2) Upon receiving Message 1, if the Session ID of the ClientHello Message is non-empty, the server will check its session cache for a match. If a match is found, and the server is willing to reestablish the connection under the specified session state, it will send a ServerHello Message with the same Session ID value. Otherwise this field will contain a different value identifying the new session.

3) In Message 3, C sends a randomly chosen new session key k_{CS} under K_S for this resumption, and also an encryption $\{C, N_C, N_S\}_{k_{CS}}$ of previously sent handshake messages under k_{CS}.

4) Upon receiving Message 3, S gets the new session key k_{CS} using S's private key K_S^{-1}, then decrypts $\{C, N_C, N_S\}_{k_{CS}}$ using k_{CS} and verifies the correction of N_C and N_S.

5) In Message 4, S encrypts the identity of himself, the randomly chosen nonce N_C and N_S, and sends $\{S, N_S, N_C\}_{k_{CS}}$ to C.

6) Upon receiving Message 4, C decrypts $\{S, N_S, N_C\}_{k_{CS}}$ using k_{CS} and verifies the correction of N_C and N_S.

Successful execution should convince C and S that k_{CS} is a new session key for this resumption. Actually, the security of the renegotiation process is even worse than the original full SSL handshake protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_C, and C also believes that N_C is open.

2) Upon receiving Message 1, from Lemma 4.2, S believes that N_C is open.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, S has the freshness assurance of the randomly chosen nonce N_S, and S also believes that N_S is open.

4) Upon receiving Message 2, from Lemma 4.2, C believes that N_S is open.

5) In Message 3, from Lemma 4.2 and Lemma 4.3, C has the confidentiality and the freshness assurances of the new session key k_{CS}.

6) Upon receiving Message 3, from Lemma 4.2 and Lemma 4.3, S has the confidentiality and the freshness assurances of the new session key k_{CS}.

7) Upon receiving Message 4, from Lemma 4.1, C has the liveness assurance of S. From Lemma 4.3, C has the freshness assurance of k_{CS}. From Lemma 4.4, S has the association assurances of N_C, N_S and k_{CS} with S, since only S could get k_{CS} from the encryption $\{k_{CS}\}_{K_S}$ using its private key, and the identity of S is explicitly indicated in Message 4.

Upon termination of the protocol run, we get analyzing result from Table 5.4, it means C believes that S is present, and the new session key k_{CS} is confidential, fresh, and associated with S, while S only believes that the new session key k_{CS} is confidential, fresh, but S does not know whether k_{CS} is associated with C and/or S or not.

Table 5.4 Security analysis of the full handshake of SSL protocol

	C				S			
	S	N_C	N_S	k_{CS}	C	N_C	N_S	k_{CS}
Message 1		01#				0?#		
Message 2			0?#				01#	
Message 3				11#				11#
Message 4	1	01S	01S	11S				
End of run	1			11S				11#

Example 5.4 (Attack-1 on SSL renegotiation) From the absence of the liveness of C in the point of view of S, we can construct an attack as shown in Fig. 5.7.

$$\begin{aligned}
&\text{Message 1} \quad I(C) \rightarrow S : Ver_C, T_C, N_C, SID_{Existing}\\
&\text{Message 2} \quad S \rightarrow I(C) : Ver_S, T_S, N_S, SID_{Existing}\\
&\text{Message 3} \quad I(C) \rightarrow S : \{k_{CS}\}_{K_S}, \{C, N_C, N_S\}_{k_{CS}}\\
&\text{Message 4} \quad S \rightarrow I(C) : \{S, N_S, N_C\}_{k_{CS}}
\end{aligned}$$

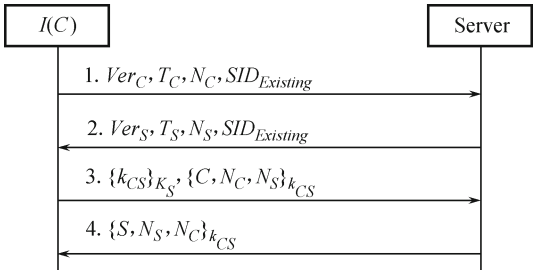


Fig. 5.7 An attack on the SSL renegotiation to cheat S.

Notation

$I(C)$ is an adversary I impersonating C. Other notations are the same as the original renegotiation SSL handshake protocol.

Premise

Premises are the same as the original renegotiation SSL handshake protocol.

Protocol actions

1) In Message 1, the adversary I starts SSL renegotiation to resume an existing session by impersonating C.

2) Upon receiving Message 1, S checks its session cache for a match and makes response to $I(C)$ with Message 2.

3) Upon receiving Message 2, $I(C)$ randomly chooses a new session key k_{CS} for this resumption between C (actually the adversary) and S, then encrypts k_{CS} under K_S and sends the encryption to S. $I(C)$ also sends the encryption of the previously sent handshake messages under k_{CS} to S.

4) Upon receiving Message 3, S gets the new session key k_{CS} using his private key K_S^{-1} and verifies $\{C, N_C, N_S\}_{k_{CS}}$. Then S believes that it must be C who is sharing the new session key k_{CS} with S.

Upon termination of the attack on the SSL renegotiation, the adversary I causes S to have false beliefs: S has completed a successful protocol run with C, and is sharing a new session key k_{CS} with C, whereas in fact, C knows nothing about the key establishment with S, and actually S shares the key k_{CS} with I. Furthermore, S concludes that subsequently messages could be encrypted using k_{CS} and safely transmitted to C (actually I).

The above attack can be launched directly by an adversary I impersonating a legitimate client C even without the liveness of C. This case is perhaps more attractive to the attacker because this characteristic permits the adversary to apply an attack on an intended victim at any time, and no particular client-side or server-side configuration is required for this attack to succeed.

5.1.3.3 Attacks related to TLS renegotiation implementation[14]

TLS (including RFC 5246 and previous ones, SSL v3 and previous ones) is subject to a number of serious man-in-the-middle (MITM) attacks related to renegotiation in real world. The TLS standard permits either end to request renegotiation of the TLS session at any time. The MITM attacks related to renegotiation are expected to generalize well to not only HTTPS but also other protocols layered on TLS. There are three general attacks related to renegotiation against HTTPS, each with slightly different characteristics, all of which yield the same result: the attacker is able to execute an HTTP transaction of his choice, authenticated by a legitimate user (the victim of the MITM attack).

Example 5.5 (Client certificate authentication renegotiation attack) The server cannot insist that the client provide a valid client certificate until it has received the certificate request from the client and filtered it through its authentication rules. For requests that are found to require client certificate authentication, the HTTPS server must then renegotiate the TLS channel to obtain and validate the certificate from the client. Unfortunately, because HTTP lacks a specific response code to instruct the client to resubmit the request within the newly authenticated channel, the server must apply the authentication retroactively to the original request. This "authentication gap" is the central weakness exploited by these attacks. Most existing installations which currently rely on client certificates for authentication appear to be vulnerable to this client certificate authentication renegotiation attack.

Example 5.6 (Differing server cryptographic requirements renegotiation attack) HTTPS servers that host resources with varying cipher suite requirements (this is often the case since web servers often host "secure" or "anonymous" content with varying certificate authentication requirements) may be vulnerable to another renegotiation attack. Because of the variations in the level of cipher suite strength, the web server has to be willing to negotiate TLS at the most basic encryption level supported on the server. Only after having seen the URL requested by the client can the server accurately determine which cipher suites will be acceptable. If the current cipher suite is not one of the required cipher suites, the server must request a renegotiation and agree on new parameters. The act of soliciting client renegotiation triggers the same weakness as in the case of client certificates: the server is forced to replay the buffered request, which in this case includes the chosen plaintext of the attacker. This has the effect of authorizing the transaction requested by the MITM.

Example 5.7 (Client-initiated renegotiation attack) TLS equally allows the client side of the connection to initiate a renegotiation. The MITM splices an initial request with an un-terminated HTTP "ignore" header onto the beginning of the client's intended request, again steals whatever authentication or authorization information provided. Note that this does not require pipelining or HTTP keep-alive. In all other respects, the server sees the same sort of request buffer as above. Most or all server applications built on TLS implementations which honor client-initiated renegotiation are vulnerable.

5.2 Internet Protocol Security

The Internet is an enormous open network of computers and devices, which are called "network nodes", each with unique Internet Protocol (IP) address. IPSec (Internet Protocol Security) is a suite of protocol designed by IETF[15, 16] to provide security for IPv4 and IPv6. The security services include confidentiality, data authentication, data integrity, and key management. IPSec provides securing communications over the Internet in key layer – the network layer of TCP/IP, hence the protection which covers the addressing information as well as the content can be very effective. In general, security at the IP layer can provide a wide protection on all applications at higher layers. Due to scalability and practical implementation considerations automatic key management seems a natural choice in significantly large virtual private networks (VPNs), IPSec has become standard by default of the most of the IP VPN technology in the world.

5.2.1 IPSec overview

IPSec has two modes: Transport mode (between two hosts) and Tunnel mode (between hosts/firewalls) and three sub protocols: Authentication Header (AH, RFC 2402[17]), Encapsulating Security Payload (ESP, RFC 2406[18]) and Internet Key Exchange Protocol (IKE, RFC 2409[19]). AH assures integrity protection, ESP provides encryption services and optional integrity protection while IKE allows communicating entities to derive session keys for secure communication via a series of messages exchange. IKE is the current IETF standard of authenticated key exchange protocol for IPSec, hence it is the concern of this book and it will be discussed in detail in Subsection 5.2.2.

5.2.1.1 Authentication Header (AH)

The Internet Protocol (IP) has evolved from version 4 (IPv4) to version 6 (IPv6). The data structure for IPv6 is a multiple of 32-bit data blocks called datagrams. In IPv6 with IPSec protection, an IP packet has an additional field called "Authentication Header" (AH)[17, 20]. Authentication protection (in fact, data integrity with origin identification) is a mandatory service for IPSec. The position for the AH in an IP packet (see Fig. 5.8, Fig. 5.9) is between "IP header" and the "TCP field". AH can have a variant length but must be a multiple of 32-bit datagrams which are organized into several subfields which contain data for providing cryptographic protection on the IP packet.

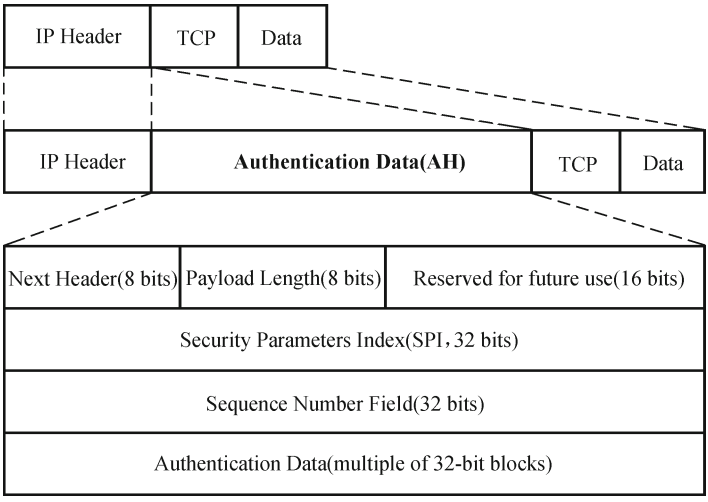


Fig. 5.8 Structure of an AH and its Position in an IP Packet in transport mode.

The subfield named "Security Parameters Index" (SPI) in an AH is an arbitrary 32-bit value which specifies (uniquely identifies) the cryptographic algorithms used for the authentication service for this IP packet. The sub-

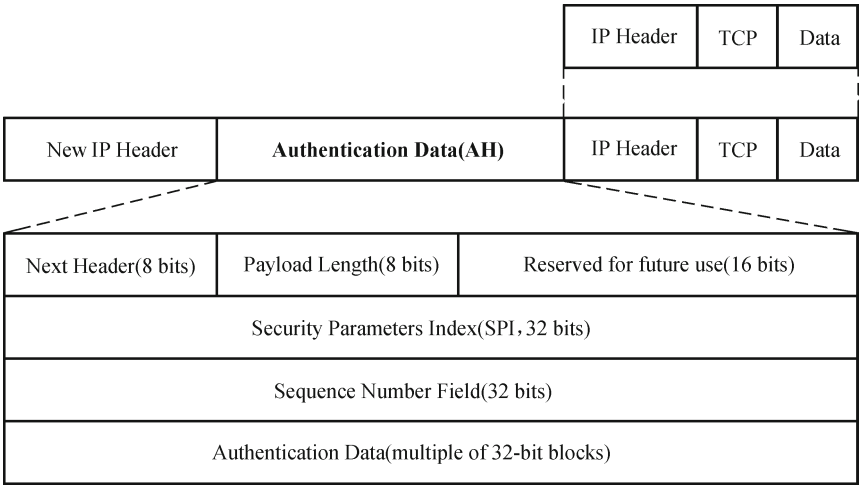


Fig. 5.9 The Structure of an AH and its Position in an IP Packet in tunnel mode.

field named "Sequence Number" can be used against replay of IP packets. The other subfield named "Authentication Data" (also called Integrity Check Value, ICV) in an AH contains the authentication data generated by the message sender for the message receiver to conduct data integrity verification. The receiver of the IP packet can use the algorithm uniquely identified in SPI and a secret key to regenerate "Authentication Data" and to compare with that received. End nodes have already established a shared secret session key manually or by IKE.

5.2.1.2 Encapsulating Security Payload (ESP)

Confidentiality (encryption) protection is an optional service for IPSec. To achieve this, a multiple of 32-bit datagrams named "Encapsulating Security Payload" (ESP)[18] is specified and allocated in an IP packet. The position for the ESP in an IP packet (see Fig. 5.10, Fig. 5.11) is between "IP header" (note that an ESP can follow an AH too) and the "TCP field". The format of an ESP is shown in Fig. 5.12.

The subfield named "Security Parameters Index" (SPI) in an ESP is an arbitrary 32-bit value which specifies (uniquely identifies) the encryption algorithm used for the confidentiality service for this IP packet.

The second subfield "Sequence Number" can be used against replay of IP packets.

The third subfield "Payload Data" has a variable length which is the ciphertext of the confidential data. Since an IP (v6) packet must have a length as a multiple of 32 bits, the plaintext "Payload Data" of variable length must be padded, and the paddings are given in "Padding". The Padding bytes are initialized with a series of (unsigned, 1-byte) integer values. The first padding

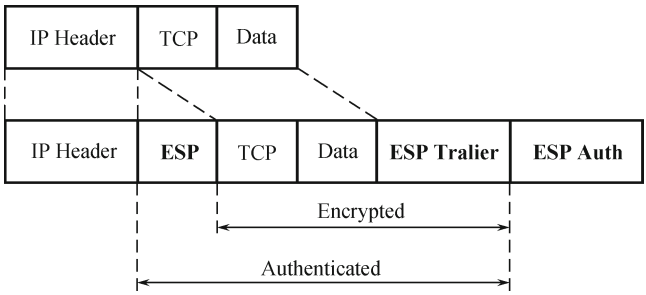


Fig. 5.10 An ESP and its Position in an IP Packet in transport mode.

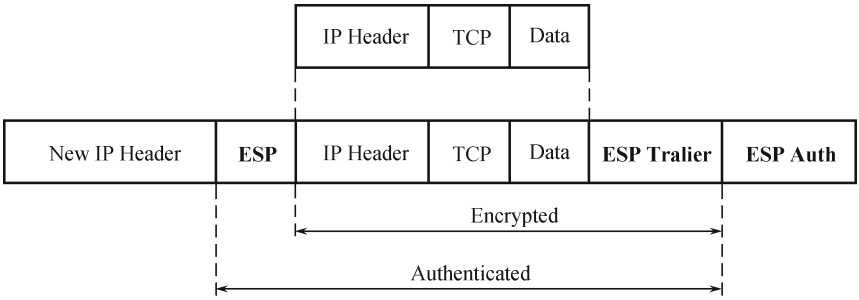


Fig. 5.11 An ESP and its Position in an IP Packet in tunnel mode.

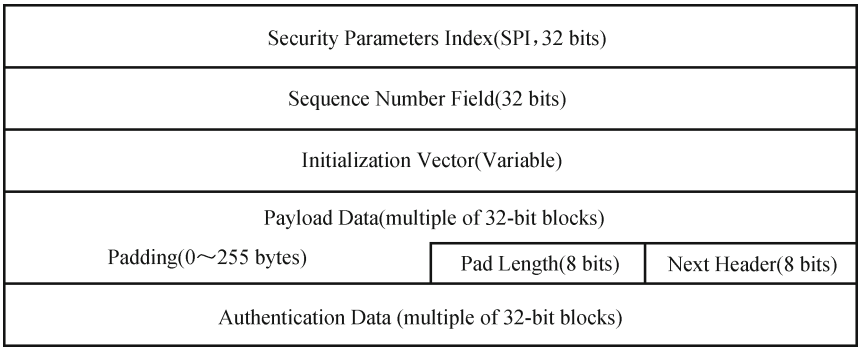


Fig. 5.12 The Structure of an Encapsulating Security Payload.

byte appended to the plaintext is numbered 1, with subsequent padding bytes making up a monotonically increasing sequence: $'01'||'02'||\cdots||'xy'$, where $'xy'$ is the hexadecimal value so that $'01' < 'xy' < 'FF'$. Therefore, the maximum number of the padding bytes is $'FF' = 255_{(10)}$. The length of the padding bytes is stated in "Pad Length".

The fourth subfield "Authentication Data" has the same meaning as that in an AH. However, "Authentication Data" in an ESP and that in an AH are

different. In an ESP, "Authentication Data" is for providing a data integrity protection on the ciphertext in the ESP packet, while in an AH, "Authentication Data" is for providing a data integrity protection on an IP packet.

5.2.2 Internet Key Exchange

Internet Key Exchange (IKE) is a set of protocols and mechanisms designed to perform two functions, creation of a protected environment (which includes authenticated peers that are unknown to each other in advance) and to establish and manage Security Associations (SA) between the authenticated peers, called the initiator and responder (or the receiver)[4, 19]. Both of the parties need to provide a digital signature in the key exchange protocol that the other party will verify. A successful verification means that the other party is authenticated. In order to be able to verify the signature also the public-key (certificate) needs to be trusted, and verified. SA defines how the traffic between the two hosts is to be protected.

Notation 5.1 (Security Associations (SA)) Each secure connection is called a security association (SA). A SA is simply a contract between two entities to provide a minimum set of services. It can be bi-directional or unidirectional. In case of unidirectional SA, which is often the case, we shall need two unidirectional SAs to complete one communication. With the view point of a programmer an SA can be considered as a data structure containing the information on Security Policy Index (SPI), its state (alive or expired), authentication algorithm, sequence number and SA life time. Considering globally, an SA is a set of proposals. A proposal can be thought of as a set of protocols and a protocol is, in turn, a set of transforms. A transform is a set of algorithms.

IKE is heart of the IPSec because it not only controls the services to be offered to secure the traffic but also manages the whole range of different transform options available at different levels and at different granularity. IKE architecture is based on three other protocols, namely, the Internet Security Association and Key Management Protocol (ISAKMP) [RFC 2408][21], the Oakley Key Determination Protocol (OAKLEY) [RFC 2412][22] and the Versatile Secure Key Exchange Mechanism for Internet (SKEME)[23].

ISAKMP provides a common framework for two communication parties to establish SA and cryptographic keys in an Internet environment. ISAKMP defines the procedures for authenticating a communicating peer, for creation and management of Security Associations, for key generation techniques, and for threat mitigation (e.g., denial of service and replay attacks). However ISAKMP does not define any specific key exchange technique so that it can support many different key exchange techniques.

OAKLEY describes a protocol by which two authenticated parties can agree on secure and secret keying material. The OAKLEY protocol supports Perfect Forward Secrecy, compatibility with the ISAKMP protocol for managing Security Associations, user-defined abstract group structures for use with the Diffie-Hellman algorithm, key updates, and incorporation of keys distributed via out-of-band mechanisms. The basic mechanism is the Diffie-Hellman key exchange algorithm.

SKEME describes an authenticated key exchange technique which supports deniability of connections between communication partners and quick key refreshment.

As a hybrid protocol of these work, IKE can be thought of as a suite of two-party protocols, featuring authenticated session key exchange, most of which in the suite use the Diffie-Hellman key exchange mechanism. IKE has many options for the two participants to negotiate and agree upon in an on-line fashion.

5.2.2.1 Two phases of IKE

IKE operates in two phases, namely, Phase 1 and Phase 2.

1. IKE Phase 1

The purpose of Phase 1 is to authenticate the communicating peers and generate the shared secret from which other keys will be computed. For IKE Phase 1, IKE has several modes, Main Mode, Aggressive Mode, Base Mode, New Group Mode, etc.

Phase 1 assumes that each of the two parties involved in a key exchange can verify the cryptographic capability of the other party, where capability might be enabled by a pre-shared secret key for a symmetric cryptosystem, or by a private key matching a reliable copy of a public-key for a public-key cryptosystem. Phase 1 attempts to achieve mutual authentication based on showing that cryptographic capability and establishes a shared session key from which other keys will be computed as an output from the IKE phases of exchanges.

2. IKE Phase 2

Phase 2 intends to create an IPSec security association and to generate new keys quickly, which relies on the shared session key agreed in Phase 1. All messages in this phase are made secure due to the algorithms and keys negotiated in Phase 1. This Create-Child-SA request may be launched by any party once Phase 1 is completed.

A multiple number of Phase 2 exchanges may take place between the same pair of entities involved in Phase 1. The reason for having a multiple number of Phase 2 exchanges is that they allow the users to set up multiple connections with different security properties, such as “integrity-only,” “confidentiality-only”, “encryption with a short key” or “encryption with a strong key”.

Notation 5.2 (IKE key material) SKEYID is a string derived from secret material known only to the active parties in the exchanges. The value SKEYID is computed separately for each authentication method and SKEYID is also a key seed of other keys.

For signature public-keys: $SKEYID = prf(nonces, g^{xy} \bmod p)$;

For encryption public-keys: $SKEYID = prf(hash(nonces), cookies)$;

For pre-shared secret key: $SKEYID = prf(pre\text{-}shared\ secret\ key,\ nonces)$.

The result of either Main Mode or Aggressive Mode is three groups of authenticated keying material:

For secret to generate other keys: $SKEYID\_d = prf(SKEYID, g^{xy}|cookies|0)$;

For integrity key: $SKEYID\_a = prf(SKEYID, SKEYID\_d|g^{xy}|cookies|1)$;

For encryption key: $SKEYID\_e = prf(SKEYID, SKEYID\_a|g^{xy}|cookies|2)$.

Here, $prf(key, msg)$ is the keyed pseudo-random function, while nonces and cookies are from the IKE exchanges between the initiator and the responder, and g^{xy} is the Diffie-Hellman shared secret.

5.2.2.2 IKE Modes

For IKE Phase 1, IKE has several modes, Main Mode, Aggressive Mode, Base Mode, New Group Mode, etc., which define how the actual key exchange procedure is to be done. Main Mode and Aggressive Mode are two of the most common modes. Main Mode (MM) has six message exchanges, and it should be run first in Phase 1, that is, two parties cannot run an aggressive mode without running a main mode first. Aggressive Mode (AM) has only three messages, and it is optional in Phase 1, that is, it can be omitted. There are four types of keys: pre-shared secret key, public encryption key (fields are separately encrypted using the public-key), optimized public encryption key (used to encrypt a random symmetric key, and then data is encrypted using the symmetric key) and public signature key (used only for signature purpose). For each key type there are two types of Phase 1 exchanges: a "main mode" and an "aggressive mode", hence there are 8 variants of IKE Phase 1.

For IKE Phase 2, IKE supports only one mode: the Quick Mode. The Quick Mode takes 3 packets to complete.

1. IKE Phase-1 Mode

The IKE Phase-1 is used to perform the mutual authentication, to exchange proposals, specific information and certificates. The result of the Phase-1 can be called in many ways: *Phase-1 SA*, *ISAKMP SA*, *IKE SA*, etc. They all mean the same thing. The Phase-1 SA is also used to protect the actual Phase-2 negotiation and other informational notifications that may be sent in IKE.

1) Phase-1 Main Mode The IKE negotiation always starts by executing the Main Mode in Phase-1 of the protocol. The Phase-1 Main Mode is

performed as shown in Fig. 5.13.

1. *Initiator* → *Responder* : *HDR, SA*
2. *Responder* → *Initiator* : *HDR, SA*
3. *Initiator* → *Responder* : *KE, NONCE, CR*
4. *Responder* → *Initiator* : *KE, NONCE, CR*
5. *Initiator* → *Responder* : *ID, *CERT, SIGR*
6. *Responder* → *Initiator* : *ID, *CERT, SIG*

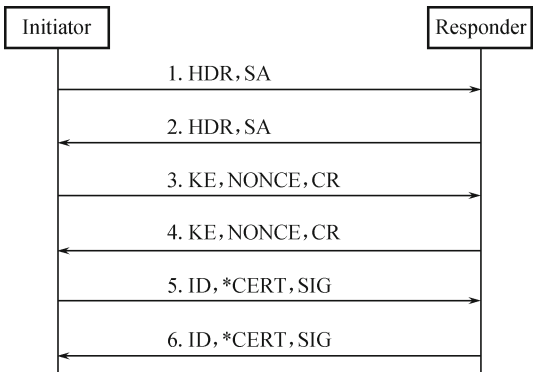


Fig. 5.13 Message flow for IKE Phase-1 Main Mode.

Notation

HDR is an ISAKMP header whose exchange type defines the IKE mode. SA is used to list the security properties supported by the initiator and the responder respectively. Key Exchange (KE) payload is the Diffie-Hellman public value, and is used to exchange the Diffie-Hellman public-keys. NONCE payload by the initiator or the responder respectively is used (with other information) in the IKE to compute the secret data for the Phase-1 SA. CR payload is certification request, and it includes the name of the CA (certification authority). ID payload is used to tell the other party who the sender or the responder is. The CERT payload includes the sender's or the responder's end entity certificate, and is also possible to send Certificate Revocation List (CRL). The signature payload (SIG) is the digital signature that the other party must verify. * indicates optional or situation-dependent messages that are not always sent.

Premise

CERT is a certification of the initiator or the responder and corresponding public-key signed by a trusted certificate authority *CA*.

Protocol actions

The first two messages negotiate policy; the next two exchange Diffie-

Hellman public values and ancillary data (e.g., nonces) necessary for the exchange, and the last two messages authenticate the Diffie-Hellman Exchange.

In Message 1: The initiator sends an HDR including an initiator cookie. The HDR is an ISAKMP header whose exchange type defines the IKE mode. An ISAKMP Header fields includes Initiator Cookie, Responder Cookie, Message ID etc. During Phase 1 negotiation, the initiator and responder cookies determine the ISAKMP SA. Message ID is a unique message identifier randomly generated by the initiator in Phase 2, which is used to identify protocol state during Phase 2 negotiation. This Message ID and the initiator's SPI(s) to be associated with each protocol in the proposal are sent to the responder. The SPI(s) will be used by the security protocols once the Phase 2 negotiation is completed. During Phase 1 negotiation, Message ID must be set to 0.

The SA payload is mandatory and it is used to list the security properties the initiator supports. It includes the ciphers, hash algorithms, key lengths, life time, and other information. It is possible to send only one SA payload in Phase-1.

In Message 2: The responder sends back an HDR including a responder cookie to the initiator. The responder must also include SA payload in its reply. The SA payload the responder sent includes the security properties it selected from the initiator's security property list (the SA payload).

Note that Message 1 and Message 2 are not encrypted, since there are no key to encrypt them with.

In Message 3: The IKE protocol is based on the Diffie-Hellman key exchange algorithm, which was the first ever invented algorithm that uses public-key cryptography (in 1976). Message 3 is used to exchange the Diffie-Hellman public-keys inside a Key Exchange (KE) payload. The Diffie-Hellman public-keys are created automatically every time the Phase-1 negotiation is performed, and they are destroyed automatically after the Phase-1 SA is destroyed.

There is also a NONCE payload that is generated by the sender and sent in Message 3 which is used (with other information) in the IKE to compute the secret data for the Phase-1 SA. The NONCE payload includes random data from random number generator.

The CR payload is a Certificate Request payload and is used to request for certificates by a specific CA. The CR payload includes the name of the CA for which it would like to receive the remote's end entity certificate (peer certificate). If empty CR payload is received, it means that it requests any certificate from any CA.

In Message 4: The responder also sends its Diffie-Hellman public-key, NONCE and CR payload in Message 4 to the initiator.

The CR payload is usually sent in the third and fourth packets, but it can be sent also in the first and second packets. Message 3 and Message 4 are not encrypted, since there is no key to encrypt them with.

In Message 5: The ID payload is used to tell the other party who the

sender is, and it also can be used to make policy decisions and to find the certificate of the remote end. The ID may be IP address, Fully Qualified Domain Name (FQDN), email address, or something similar.

The initiator may send zero (0) or more certificate payloads (CERT), with each including one certificate (or CRL). The CERT payload is optional payload, but usually if it is not sent, the result of the IKE is "Authentication Failed" error. The CERT payload includes the sender's end entity certificate, but it is also possible to send CRL inside a CERT payload. The CERT payload is optional because it is possible that the remote end has cached the public-key locally, and does not need to receive the CERT payload in the negotiation. Usually implementations do not cache it locally and in this case failing to send CERT payload also causes failure of the IKE negotiation.

The signature payload (SIG) is the digital signature that the other party must verify. The SIG payload includes the digital signature computed with the private key of the corresponding public-key (usually sent inside the CERT payload), and provides the authentication to the other party. When both of the parties successfully verify each other's SIG payloads, they are then mutually authenticated. They use the public-key found in the certificate (usually received in CERT payload, or some other means (cached locally, fetched from Lightweight Directory Access Protocol (LDAP), etc.)) to verify the signature. Before verifying the signature they also verify the certificate of the remote end. They check whether they trust the issuer (Certificate Authority, CA) of the certificate, and they also check whether the certificate is valid (not revoked, etc.) or not.

In Message 6: The responder also sends its ID, CERT and SIG payload in Message 6 to the initiator.

Note that Message 5 and Message 6 are fully encrypted, since the key was computed after Message 3 and Message 4 (where the Phase-1 SA is created and Diffie-Hellman public values is computed). When communication is protected, all payloads following the ISAKMP header MUST be encrypted.

After these packets are sent and the digital signatures are successfully verified the result of this Phase-1 negotiation is the Phase-1 SA, which can be used to protect other packets sent in the IKE, such as the packets of the Phase-2 negotiation. This also completes the Phase-1 negotiation successfully.

2) The Phase-1 Aggressive Mode The Phase-1 Aggressive Mode is performed as shown in Fig. 5.14.

1. Initiator → Responder: *VID, SA, KE, NONCE, ID*
2. Responder → Initiator: *VID, SA, KE, NONCE, ID, *CERT, SIG*
3. Initiator → Responder: **CERT, SIG*

Notation

VID means Vendor ID. Other notations are the same as in the IKE Phase-1 Aggressive Mode

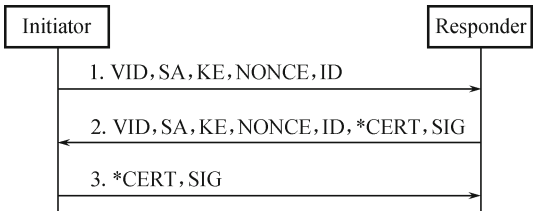


Fig. 5.14 Message flow for IKE Phase-1 Aggressive Mode.

Premise

CERT is a certificate of the initiator or the responder and its corresponding public-key signed by a trusted certificate authority *CA*.

Protocol actions

1) In Message 1: The initiator sends the crypto proposal supported, exchanges Diffie-Hellman public values and ancillary data necessary for the exchange, and identities.

2) In Message 2: The responder selects the crypto supported from the initiator's security property list (the SA payload), exchanges Diffie-Hellman public values and ancillary data necessary for the exchange, and identities. In addition Message 2 authenticates the responder.

3) In Message 3: Message 3 authenticates the initiator and provides a proof of participation in the exchange.

2. The IKE Phase-2 Mode

After the Phase-1 is successfully completed the Phase-2 negotiation can proceed. The purpose of the Phase-2 exchange is to provide and to refresh the key material that is used to create the Security Associations (SAs) to protect the actual IP traffic with IPSEC. The Phase-2 exchange is protected by encrypting the Phase-2 packets with the key material derived from the Phase-1. The Phase-2 also provides proposal list which defines the actual ciphers, HMACs, hash algorithms and other security properties that are used in the protection of the IP traffic. The proposal proposed in the Phase-1 is merely for protection of traffic under the Phase-1 SA (like the packets of the Phase-2), and not for the actual IP traffic. The Phase-2, also called the Quick Mode, is for the protection of the actual IP traffic. Since the ISAKMP SA is bi-directional, either communication party may initiate Quick Mode. The Phase-2 Quick Mode is performed as shown in Fig. 5.15.

Notation

HASH payload is the keying material exchanged. The SA payload is the Phase-2 proposal list which indicates the security properties. The NONCE payload includes always random data. The KE payload is from the ephemeral Diffie-Hellman exchange of Phase-1 Main Mode, and is confidential in Quick Mode. The ID payload is the participant's ID, usually IP address. * indicates

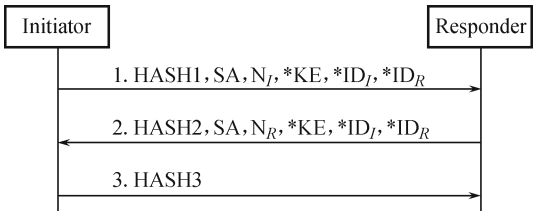


Fig. 5.15 Message flow for IKE Phase-2 Quick Mode.

optional or situation-dependent messages that are not always sent.

Premise

Diffie-Hellman public values have already exchanged in Phase 1.

Protocol actions

1) In Message 1: The initiator sends SA, HASH, NONCE and ID payloads to the responder to generate a new session key for IP traffic.

The SA payload is the Phase-2 proposal list which includes the ciphers, HMACs, hash algorithms, life times, key lengths, the IPSEC encapsulation mode (ESP, AH etc.) and other security properties. Note that it is possible to send more than one SA payloads in Phase-2, although usually only one is sent.

The HASH and NONCE payloads (marked here as N_I) are the keying material which are exchanged, and then are used to create the new key pair. The NONCE payload includes always random data.

The ID payloads, marked as ID_I and ID_R, for initiator's ID and responder's ID, respectively, are optional in Phase-2. Usually IKE implementations do send the ID payloads in Phase-2 since they can be easily used to make local policy decisions. However, as noted, they are not mandatory and can be omitted. The ID_I is the initiator's ID, usually IP address or similar, and the ID_R is the responder's ID, usually IP address, IP range or IP subnet. Both of the initiator and responder usually use the ID payloads to search the local policy for matching connection. The ID payloads in the Phase-2 are also called "proxy IDs", "pseudo IDs" or similar, since they do not necessarily represent the actual negotiator (for example when Security Gateway (SGW) negotiates on behalf of some client).

2) In Message 2: The responder selects the crypto from the Phase-2 proposal list, and sends HASH, NONCE (marked here as N_R) and ID payloads.

3) In Message 3: The initiator sends a HASH payload.

After the Phase 2 has been completed by sending the last packet, the result of the Phase-2 is two Security Associations (SAs). One is for inbound traffic, the other is for outbound traffic. This also completes the IKE key exchange for basic key exchange.

Note that all messages in Phase 2 are encrypted using $SKEYID\_e$, and integrity protected using $SKEYID\_a$.

5.2.3 Security analysis of IKE based on trusted freshness

The examples in this subsection assume that digital signatures are used in authentication and key establishment of IKE.

5.2.3.1 Security analysis of main mode

Example 5.8 (Public Signature Keys, IKE Phase-1 Main Mode) Figure 5.16 illustrates the message exchanges related to authentication and key establishment in public signature key-based IKE Phase-1 Main Mode, and some minute details are omitted.

Message 1 $A \rightarrow B:\ HDR_A, SA_A$

Message 2 $B \rightarrow A:\ HDR_B, SA_B$

Message 3 $A \rightarrow B:\ HDR_A, g^x, N_A$

Message 4 $B \rightarrow A:\ HDR_B, g^y, N_B$

Message 5 $A \rightarrow B:\ HDR_A, \left\{ ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}} \right\}_{SKEYID}$

Message 6 $B \rightarrow A:\ HDR_B, \left\{ ID_B, Cert_B, \{HASH_B\}_{K_B^{-1}} \right\}_{SKEYID}$

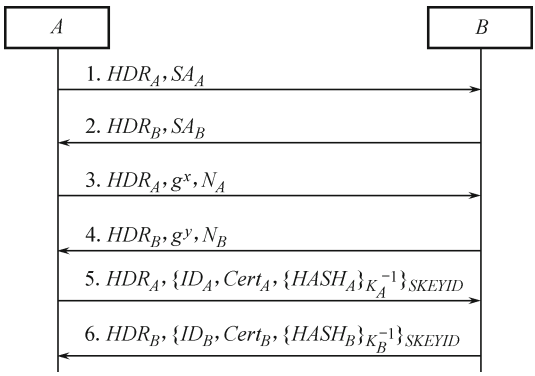


Fig. 5.16 Message exchanges in signature-based IKE phase-1 main mode.

Notation

A denotes the initiator, B denotes the responder.

HDR_A and HDR_B, ISAKMP headers of A and B, respectively are for keeping the session state information for these two entities.

SA_A, SA_B are the Security Associations (SA) of A and B, respectively. A and B use SA_A, SA_B to negotiate parameters to be used in the current run of the protocol: encryption algorithms, signature algorithms, pseudo-random functions for hashing messages to be signed, etc. A may propose a set of proposals, whereas B must reply with only one choice.

x and y are the private values randomly chosen by the initiator A and the responder B respectively. g^x, g^y are the Diffie-Hellman public values of A and B respectively. The g^{xy} can be computed via $(g^y)^x$ or $(g^x)^y$ by A and B, respectively.

N_A and N_B are nonces randomly chosen by A and B, respectively.

ID_A and ID_B are endpoint identities of A and B, respectively.

$Cert_A$ and $Cert_B$ are certifications of A and B issued by a trusted certificate authority CA, respectively.

$HASH_A$ and $HASH_B$ are hash values computed by A and B, respectively. The entire ID payload (including ID type, port, and protocol but excluding the generic header HDR) is hashed into both $HASH_A$ and $HASH_B$.

$$HASH_A = prf_1\,(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$$
$$HASH_B = prf_1\,(SKEYID|g^y|g^x|C_B|C_A|SA_B|ID_B)$$

where $SKEYID$ is the new session key between A and B, which can be computed as $SKEYID = prf_2(N_A|N_B|g^{xy})$. C_A and C_B are the initiator's and the responder's cookie respectively. prf_1 and prf_2 are pseudo-random functions agreed in SAs. For A and B to authenticate each other, the mutually obtainable hash values $HASH_A$ and $HASH_B$ will be signed by A and B respectively via the negotiated digital signature algorithm.

Note that we do not distinguish $SKEYID$ from $SKEYID\_d$, $SKEYID\_a$, $SKEYID\_e$, etc., since all these keys can be derived from $SKEYID$ and some open key materials.

Premise

Both A and B trust the issuer CA of the certificates $Cert_A$ and $Cert_B$, and they could verify that the certificate is valid (not revoked, etc.). That is, they know the public-key of CA to get K_A of A and K_B of B. Each principal knows the key pair of himself, that is, K_A and K_A^{-1} for A, K_B and K_B^{-1} for B.

Protocol actions

1) In Message 1, the initiator A starts the negotiation by sending an HDR_A including a null Message ID, an initiator cookie C_A, and an SA_A including encryption algorithms, signature algorithms, pseudo-random functions for hashing messages to be signed, key lengths, life time, and other information.

2) Upon receiving Message 1, since the Message ID is NULL, then a Phase-1 Main Mode is applied.

3) In Message 2, the responder B sends back an HDR_B including a responder cookie C_B and also the initiator cookie C_A to A. The responder also includes an SA_B in its reply to indicate the security properties B selects.

4) In Message 3, A randomly chooses a Diffie-Hellman private value x and a nonce N_A for this run. Then A computes the Diffie-Hellman public-key g^x and sends it to the responder B with N_A.

5) In Message 4, B also randomly chooses a Diffie-Hellman private value y and a nonce N_B for this run. Then B computes the Diffie-Hellman public value g^y and sends it to the initiator A with N_B.

Up to now, they have exchanged the Diffie-Hellman public values and ancillary data (e.g., nonces) necessary, and both A and B could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$.

6) In Message 5, A computes the hash of the entire ID payload $HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$, signs $HASH_A$ using A's private key K_A^{-1} to show his identity of A, and then encrypts all payloads following the ISAKMP header $\{ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}}\}$ using the negotiated new session key $SKEYID$.

7) Upon receiving Message 5, B decrypts $\{ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}}\}_{SKEYID}$ to get $\{ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}}\}$ using $SKEYID$, verifies $Cert_A$ to get K_A, and then checks the correction of the hash value:

$$\left\{\{HASH_A\}_{K_A^{-1}}\right\}_{K_A} = HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A).$$

8) In Message 6, B does similar things as A has done in Message 5. Upon receiving Message 6, A does similar things as B has done upon receiving Message 5.

Successful execution should authenticate the identities of the communication parties A and B, and establish a new session key $SKEYID$ between A and B. Actually, Phase-1 Main Mode l suffers from a flaw which has been proved by Lowe, Meadows and Mao respectively[5, 24, 25].

Protocol security analysis

1) In Message 1 and Message 2, neither A nor B could draw any useful assurance from it since there is not any trusted freshness identifier from the point of view of the two parties.

2) In Message 3, from Lemma 4.2 and Lemma 4.3, A has the freshness assurance of the randomly chosen Diffie-Hellman private value x and the nonce N_A, and A also believes that x is confidential and N_A is open.

3) Upon receiving Message 3, from Lemma 4.2, B believes that N_A is open and x is confidential.

4) Similarly, in Message 4, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the randomly chosen Diffie-Hellman private value y, and the nonce N_B is fresh and open. Further, B could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$, and from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of $SKEYID$.

5) Upon receiving Message 4, from Lemma 4.2, A believes that N_B is open and y is confidential. Further, A could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$, and from Lemma 4.2 and Lemma 4.3, A has the confidentiality and the freshness assurances of $SKEYID$.

6) In Message 5, A could not get any new assurance.

7) Upon receiving Message 5, from Lemma 4.1, B has the liveness assurance of A based on the trusted freshness N_B and $SKEYID = prf_2(N_A|N_B|g^{xy})$, since it must be A who has signed the fresh hash value $HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$ using A's private key. From Lemma 4.4, B has the association assurance of $SKEYID$ with A, since only A could sign $HASH_A$ using A's private key, and the identity of A is explicitly indicated in Message 5.

8) Similarly, in Message 6, B could not get any new assurance. Upon receiving Message 6, A has the liveness assurance of B and the association assurance of $SKEYID$ with B.

Upon termination of the protocol run, the analyzing result from Table 5.5shows that A believes that B is present, and the new session key $SKEYID$ is confidential, fresh, and associated with B, while B believes that A is present, and the new session key $SKEYID$ is confidential, fresh, and associated with A.

Table 5.5 Security analysis of the IKE phase-1 main mode

	A						B					
	B	N_A	N_B	x	y	$SKEYID$	A	N_A	N_B	x	y	$SKEYID$
Message 1												
Message 2												
Message 3		01#		11#				0?#		1?#		
Message 4			0?#		1?#	11#			01#		11#	11#
Message 5							1					11A
Message 6	1					11B						
End of run	1					11B	1					11A

Example 5.9 (Attack-1 on Signature-based, IKE Phase-1 Main Mode) From the absence of the association of $SKEYID$ with B in the point of view of B, there exists an attack[5, 24, 25], as shown in Fig. 5.17.

$$
\begin{aligned}
&\text{Message } 1 \quad A \to I: && HDR_A, SA_A\\
&\text{Message } 1' \quad I(A) \to B: && HDR_A, SA_A\\
&\text{Message } 2' \quad B \to I(A): && HDR_B, SA_B\\
&\text{Message } 2 \quad I \to A: && HDR_B, SA_B\\
&\text{Message } 3 \quad A \to I: && HDR_A, g^x, N_A\\
&\text{Message } 3' \quad I(A) \to B: && HDR_A, g^x, N_A\\
&\text{Message } 4' \quad B \to I(A): && HDR_B, g^y, N_B\\
&\text{Message } 4 \quad I \to A: && HDR_B, g^y, N_B\\
&\text{Message } 5 \quad A \to I: && HDR_A, \left\{ ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}} \right\}_{SKEYID}\\
&\text{Message } 5' \quad I(A) \to B: && HDR_A, \left\{ ID_A, Cert_A, \{HASH_A\}_{K_A^{-1}} \right\}_{SKEYID}
\end{aligned}
$$

Message 6′ $B \rightarrow I(A): \ HDR_B, \left\{ ID_B, Cert_B, \{HASH_B\}_{K_B^{-1}} \right\}_{SKEYID}$

Message 6 $I \rightarrow A:$ Dropped

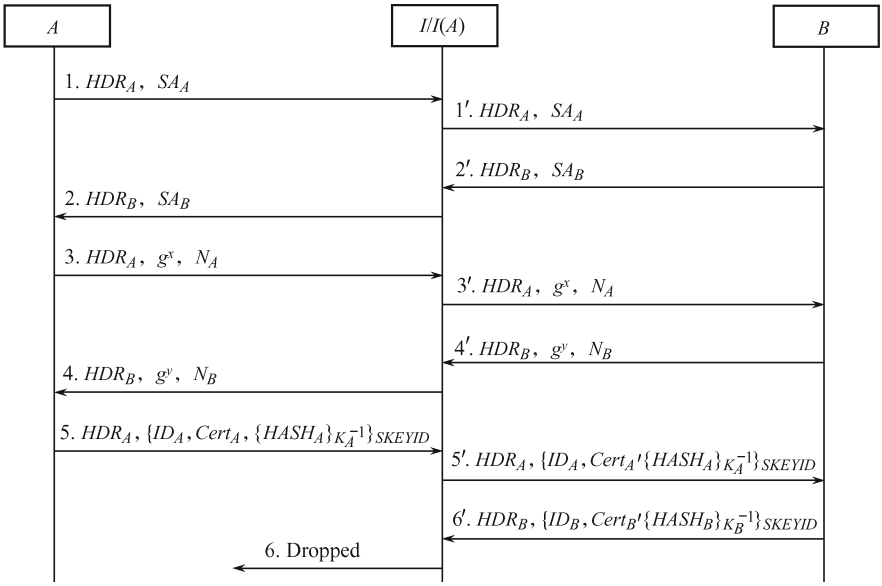


Fig. 5.17 An attack on the signature-based IKE phase-1 main mode.

Notation

I denotes the adversary, and $I(A)$ is an adversary I impersonating the initiator A. Other notations are the same as the original IKE phase-1 main mode.

Premise

The premises are the same as the original IKE phase-1 main mode.

Protocol actions

1) In Message 1, the initiator A starts a protocol run with I. In Message $1'$, the adversary $I(A)$ replays the Message 1 to B to start a fake protocol run between A and B by impersonating B. B responds to Message $2'$.

2) Upon receiving Message $2'$ and so on, the adversary I just replays all the follwing messages to the initiator A or the responder B respectively until receiving Message $6'$.

3) Upon receiving Message $6'$, the adversary I just drops this message.

Upon termination of the attack on the IKE Phase-1 Main Mode with both side certificates, the adversary I causes B to have false beliefs: B has completed a successful protocol run with A, and is sharing a new session key *SKEYID* with A, whereas in fact, A knows nothing about the key establish-

ment with B, while A thinks that A has been talking with I in an incomplete run. Furthermore, B will never be notified of any abnormality and B will keep the session state information for these two entities A (Actually the adversary I) and B. This is effective for the adversary I to make a denial of service attacks[24].

Note that the adversary I could not launch an attack similar to Example 5.3, since the $HASH_A$ includes the confidential and fresh seed of *SKEYID*.

5.2.3.2 Security analysis of aggressive mode

Example 5.10 (Public signature keys, IKE phase-1 aggressive mode) Aggressive Mode is a cut-down simplification from IKE Phase-1 Main Mode, and Fig. 5.18 shows the message exchanges with some minute details omitted. The first two messages negotiate policy, exchange Diffie-Hellman public values and ancillary data necessary for the exchange, and identities. In addition, the second message authenticates the responder. The third message authenticates the initiator and provides a proof of participation in the exchange.

Aggressive Mode can be used to reduce round trips even further.

Message 1 $A \rightarrow B:\ HDR_A, SA_A, g^x, N_A, ID_A$

Message 2 $B \rightarrow A:\ HDR_B, SA_B, g^y, N_B, ID_B, [Cert_B], \{HASH_B\}_{K_B^{-1}}$

Message 3 $A \rightarrow B:\ [Cert_A], \{HASH_A\}_{K_A^{-1}}$

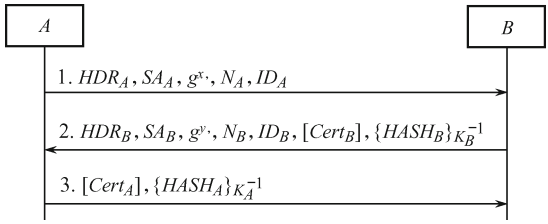


Fig. 5.18 Message exchanges in signature-based IKE phase-1 aggressive mode.

[x] indicates that x is optional, other notations, premise and protocol actions in this mode are the same as those in Main Mode (Example 5.8), hence omitted.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the freshness assurance of the private value x and the nonce N_A, and A also believes that x is confidential and N_A is open.

2) Upon receiving Message 1, from Lemma 4.2, B believes that N_A is open and x is confidential.

3) In Message 2, from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of the private value y, and the nonce N_B

is fresh and open. Further, B could compute the new session key $SKEYID = prf_2(N_A|N_B|g^{xy})$, and from Lemma 4.2 and Lemma 4.3, B has the confidentiality and the freshness assurances of $SKEYID$.

4) Upon receiving Message 2, from Lemma 4.1, A has the liveness assurance of B based on the trusted freshness N_A and $SKEYID = prf_2(N_A|N_B|g^{xy})$, since it must be B who has signed the fresh hash value $HASH_B = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_B|ID_B)$ using B's private key. From Lemma 4.4, A has the association assurance of $SKEYID$ with B, since only B could sign $HASH_B$ using B's private key, and the identity of B is explicitly indicated in Message 2.

5) Similarly, upon receiving Message 3, from Lemma 4.2 and Lemma 4.3, B has the liveness assurance of A based on the trusted freshness N_B and $SKEYID = prf_2(N_A|N_B|g^{xy})$, since it must be A who has signed the fresh hash value $HASH_A = prf_1(SKEYID|g^x|g^y|C_A|C_B|SA_A|ID_A)$ using A's private key. From Lemma 4.4, B has the association assurance of $SKEYID$ with A, since only A could sign $HASH_A$ using A's private key, and the identity of A is explicitly indicated in Message 3.

Upon termination of the protocol run, the analyzing result is indicated in Table 5.6, it means A believes that B is present, and the new session key $SKEYID$ is confidential, fresh, and associated with B, while B believes that A is present, and the new session key $SKEYID$ is confidential, fresh, and associated with A.

Table 5.6 Security analysis of the IKE phase-1 aggressive mode

	A						B					
	B	N_A	N_B	x	y	$SKEYID$	A	N_A	N_B	x	y	$SKEYID$
Message 1		01#		11#				0?#		1?#		
Message 2	1		0?#		1?#	11B			01#		11#	11#
Message 3							1					11A
End of run	1					11B	1					11A

Note that $HASH_A$ and $HASH_B$ take the new session key $SKEYID$ as its seed input to pseudo-random function, hence the signature of these two hash values could not be faked, and the signatures are exclusively verifiable by the principals who hold the agreed session key. However, the attack (Example 5.9 Attack-1) on IKE main mode is still effect on the IKE aggressive mode.

5.2.3.3 Security analysis of quick mode

Example 5.11 (Public signature keys, IKE phase-2 quick mode) Each instance of a Quick Mode uses a unique initialization vector (e.g., Message ID), hence it is possible to have multiple simultaneous Quick Modes, based on a single ISAKMP SA, in progress at any one time. Quick Mode is essentially an SA negotiation and exchanges of nonces that provides replay protection. The nonces are used to generate fresh key material and to prevent replay attacks from generating bogus security associations. Base Quick Mode (without the

KE payload) refreshes the keying material derived from the exponentiation in Phase-1. Figure 5.19 illustrates the Quick Mode exchanges related to security.

Message 1 $A \rightarrow B:\ HDR_A, \{HASH_1, SA_A, N_A, [g^x], [ID_A, ID_B]\}_{SKEYID}$

Message 2 $B \rightarrow A:\ HDR_B, \{HASH_2, SA_B, N_B, [g^y], [ID_A, ID_B]\}_{SKEYID}$

Message 3 $A \rightarrow B:\ HDR_A, \{HASH_3\}_{SKEYID}$

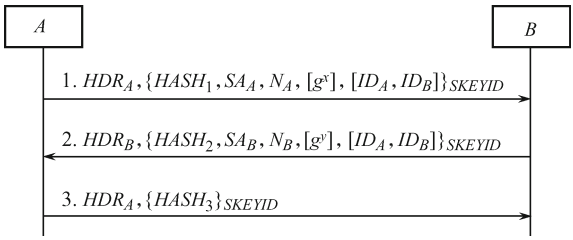


Fig. 5.19 Message exchanges in signature-based IKE phase-2 quick mode.

Notation

[x] indicates that x is optional, other notations in Quick Mode are the same as those in Main Mode except the following:

k is a new session key generated from this Quick Mode run.

$$k = prf(SKEYID\_d, g_{qm}^{xy} | protocol | SPI | N_A | N_B)$$

where g_{qm}^{xy} is the shared secret from the ephemeral Diffie-Hellman exchange of this Quick Mode. "*protocol*" and "*SPI*" are from the ISAKMP proposal payload that contains the negotiated transform.

$HASH_1$ and $HASH_3$ are hash values computed by A, while $HASH_2$ by B:

$$HASH_1 = prf(SKEYID\_a, ID_{AB} | SA_A | N_A [| g^x][| ID_A | ID_B])$$
$$HASH_2 = prf(SKEYID\_a, ID_{AB} | N_A | SA_B | N_B [| g^y][| ID_A | ID_B])$$
$$HASH_3 = prf(SKEYID\_a, 0 | ID_{AB} | N_A | N_B)$$

ID_{AB} is a message ID to indicate a Quick Mode in progress for a particular ISAKMP SA, and this particular SA is identified by the cookies in the ISAKMP header.

Premise

g^{xy} is from the ephemeral Diffie-Hellman exchange of Phase-1 Main Mode, and is confidential in Quick Mode. Suppose the key $SKEYID\_a$ from Main Mode is confidential and only known by A and B. Other premises are the same as those in Main Mode.

Protocol actions

Omitted for interest of concision.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, A has the freshness assurance of the nonce N_A, and the confidentiality and freshness assurances of the value x. From Lemma 4.4, A has the association assurance of N_A and x with A and B, since the key $SKEYID\_a$ from Main Mode is confidential and only known by A and B.

2) Upon receiving Message 1, B could not draw any useful assurance from Message 1 since there is not any trusted freshness identifier from the point of view of B.

3) Similarly, in Message 2, from Lemma 4.2 and Lemma 4.3, B has the freshness assurance of the nonce N_B, and the confidentiality and freshness assurances of both y and k. From Lemma 4.4, B has the association assurance of N_B, y and k with A and B.

4) Upon receiving Message 2, from Lemma 4.2, Lemma 4.3, and Lemma 4.4, A believes that k is confidential, fresh and associated with A and B based on the trusted freshness N_A. From Lemma 4.1, A has the liveness assurance of B based on the trusted freshness N_A, since it must be B who has generated the fresh hash value $HASH_2 = prf(SKEYID\_a, ID_{AB}|N_A|SA_B|N_B[|g^y][|ID_A|ID_B])$ using the shared key $SKEYID\_a$.

5) Similarly, upon receiving Message 3, from Lemma 4.1, B has the liveness assurance of A from the hash value $HASH_3 = prf(SKEYID\_a, 0|ID_{AB}|N_A|N_B)$ including the trusted freshness N_B.

Table 5.7 shows the analyzing result of the IKE phase-2 quick mode. Upon termination of the protocol run, with the premise that the g^{xy} from Main Mode is confidential in Quick Mode, the protocol achieves that A believes that B is present, and the new session key k is confidential, fresh, and associated with both A and B, while B believes that A is present, and the new session key k is confidential, fresh, and associated with A. Hence, IKE Phase-2 Quick Mode has achieved the key exchange and authentication security objects as it intends to.

Table 5.7 Security analysis of the IKE phase-2 quick mode

	A						B					
	B	N_A	N_B	x	y	k	A	N_A	N_B	x	y	k
Message 1		$01AB$		$11AB$				$0?\#$		$1?\#$		
Message 2	1		$0?\#$		$1?\#$	$11AB$			$01AB$		$11AB$	$11AB$
Message 3							1					
End of run	1					$11AB$	1					$11AB$

However, we should note that if g^{xy} from Main Mode was not confidential in Quick Mode, that is, if g^{xy} was known by the adversary, then the adversary could initiate a Quick Mode instance at any one time by impersonating A or

B, and shared a new session key with the victim.

5.3 Kerberos — the network authentication protocol

In open network computing environments, a user (an employee, a subscriber or a customer) may be provided with various kinds of information resources and services: remote hosts, file servers, printers, and many other networked services. When a user requests use of a network service, the service provider wants an assurance that the user is who he says he is in a physically insecure network. However, it would be unrealistic and uneconomic to require a user to maintain several different cryptographic assurances, no matter whether in terms of memorizing various passwords, or in terms of holding a number of smartcards. Furthermore, unencrypted passwords sent over the network may suffer from the "sniff" attack. Hence, Kerberos[6, 7] is introduced by Massachusetts Institute of Technology (MIT) as a solution to these network security problems: a trusted third party mediates between a user and a resource server by issuing a shared session key between the two entities. The Kerberos protocol allows a legitimate user to log onto his terminal once a day (typically) and then transparently access all the networked resources he needs for the rest of that day. Each time the legitimate user wants to access an information resource, to retrieve a file from a remote server for example, Kerberos will securely handle the required authentication behind the scene without any user intervention.

5.3.1 Kerberos overview

Kerberos is developed as part of the MIT Athena project. The Kerberos protocol is designed to provide strong authentication so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and a server have proved their identities via Kerberos, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business. Furthermore, Kerberos is being used as a building block for higher-level protocols. Kerberized applications are those that use the Kerberos authentication protocol to provide authentication, and to provide encryption and signing for subsequent messages. Kerberos relies on conventional encryption rather than public-key encryption, that is, it uses private-key cryptography. PKINIT[26], which adds public-key authentication and a fair amount of complexity to the basic protocol, is an extension of Kerberos 5.

In basic Kerberos, a session generally starts with a user logging onto a system. This triggers the creation of a client process that will transparently

handle all his authentication requests. The client process — usually acting for a human user – interacts with three other types of principals when using Kerberos 5. The initial authentication between the client and the Kerberos administrative principal is traditionally based on a shared key derived from a password chosen by the user. The client's goal is to be able to authenticate himself/herself to various application servers (e.g., email, file, and print servers). This is done by obtaining a "Ticket-granting ticket" (TGT) from a "Kerberos Authentication Server" (AS) and then presenting this TGT to a "Ticket-Granting Server" (TGS) in order to obtain a "Service ticket" (ST). ST is the credential that the client uses to authenticate himself/herself to the application server. The AS and the TGS together are known as the "Key Distribution Center" (KDC)

A TGT might be valid for a day, and may be used to obtain several STs for many different application servers from the TGS, while a single ST is valid for a few minutes (although it may be used repeatedly) and is used for a single application server. That is, Authentication Service Exchange occurs once for every logon session, the user doesn't need to login every time he starts an application that uses Kerberos. *Delegation* refers to the facility for a service to impersonate an authenticated client in order to relieve the user of the additional burden of authenticating to multiple services. To the latter services, it will look as if they are communicating directly with the user, whereas in reality another service will sit between them and the user.

Kerberos can provide authentication, authorization and accounting security properties. Authentication is the confirmation that a user who is requesting services is a valid user of the network services requested. Authorization is the granting of specific type of service to a user based on their authentication, and what services they are requesting and what the current system state is. Accounting is the tracking of the consumption of network resources by users.

Kerberos is more secure than LAN Manager(LM) authentication and NTLM authentication, since user's passwords are never sent across the network encrypted or in plain text; session keys are only passed across the network in encrypted form; client and server systems are mutually authenticated, and Kerberos limits the duration of their users' authentication.

There are two major versions of Kerberos in common use. Version 4 is the most widely used version and it uses DES encryption algorithm, which has been shown to be vulnerable to brute-force-attacks with little computing power. Version 5 is a draft Internet Standard (RFC 1510[6]) which has corrected some of the security deficiencies of Version 4. For example, Kerberos Version 5 has indicated the notion realm of the user, added a random nonce to assure the response fresh, improved the ticket lifetime to enhance the security, etc.

Here are the related ports, protocols and functions of Kerberos:

Port	Protocol	Function
88	UDP TCP	Kerberos V5
750	UDP TCP	Kerberos V4 Authentication
751	UDP TCP	Kerberos V4 Authentication
752	UDP	Kerberos password server
753	UDP	Kerberos user registration server
754	TCP	Kerberos slave propagation
1109	TCP	POP with Kerberos
2053	TCP	Kerberos demultiplexer
2105	TCP	Kerberos encrypted rlogin

In Windows 2000, 56bit DES and 128bit RC4 are the most commonly used ones in Kerberos; in Windows Server 2003, RC4-HMAC, DES-CBC-CRC and DES-CBC-MD5 are commonly used; in Windows XP, RC4 is commonly used while others are allowed, and DES is notably used; in Windows Vista, 256bit AES, 3DES, SHA2 are commonly used.

5.3.1.1 Terms

Term 5.1 Kerberos uses "*realm*" to group user accounts. A Kerberos realm means a single Kerberos administrative domain, and it includes at lease a Kerberos server, a number of Clients and several Application servers.

Kerberos supports inter-realm authentication, but the Kerberos server in each realm should shared a secret key with those servers in other realms, and the Kerberos server in one realm should trust the one in other realm to authenticate its users. Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child.

A Kerberos server, which is a trusted third party, or, in Kerberos terminology, the Key Distribution Center (KDC). The KDC itself is made of two subservices: the Authentication Service and the Ticket Granting Service. In Windows 2000 and Windows Server 2003, both services run on the KDC server. While in other Kerberos implementations these two subservices can run on different machines such as an Authentication Server (AS) and a Ticket Granting Server (TGS). This is for the scenario where application servers belong to different TGS's in different domains.

A number of clients, all should have registered with the Kerberos server KDC to use the information resources and services.

Several application servers, which are target servers that provide information resources and services, and they all have shared long-term keys with the Kerberos server KDC.

Notation 5.3 Key Distribution Center (KDC) is composed of an Authentication Server (AS) and a Ticket Granting Server (TGS). It has a database that houses all principals, including severs and clients, and their keys for a given realm. For example, the Kerberos KDC runs on every Windows 2000 domain controller and Windows 2003 server.

Notation 5.4 Authentication Server (AS) authenticates a client logon and issues a Ticket Granting Ticket (TGT) for future authentication.

Notation 5.5 Ticket Granting Server (TGS) is responsible for accepting and verifying TGT from the AS, and grants application service tickets to clients holding this TGT. The existence of TGS allows the clients only to have to authenticate themselves once to the AS to get TGT, which can then be presented to the TGS.

Notation 5.6 Application Server (S) is responsible for accepting and verifying service tickets from the TGS, and grants information resources and services to a network client.

Notation 5.7 Client (C)Client is a Client (a user process) which makes use of a network service on behalf of a user. The user credential is given to the client C as C prompts the user to key-in his password. Note that in some cases a server may itself be a client of some other servers. Kerberos assumes that the workstations or machines are secure, i.e., there is no way for an attacker to intercept communication between a user and a client.

Notation 5.8 Ticket or Kerberos ticket is encrypted protocol messages used to confirm identities of the end participants and to establish a new session key that both parties will shared for secure communication. Kerberos uses two types of tickets in its process of authentication: TGTs and Service Tickets.

Notation 5.9 Authenticator consists of timestamps that are encrypted with the secret session key shared between the client and the AS, or between the client and the application server. Note that the timestamp cannot exceed the expiration time. The authenticator has a very short life time to prevent replay attacks, and the authenticator can only be used once. One authenticator is typically built in per session of use of a service.

Notation 5.10 Ticket Granting Ticket (TGT) is issued by the Authentication Server (SA) that contains the client's Privilege Attribute Certificate (PAC).

Notation 5.11 Privilege Attribute Certificate (PAC) is strictly used in Windows 2000 Kerberos authentication, which contains information such as the user's Security ID (SID), group membership SIDs, and users' rights on the domain.

Notation 5.12 Service Ticket is issued by the Ticket Granting Server, which provides authentication for a specific application server or resource.

Notation 5.13 Session key is a derived value used strictly for the immediate session between a client and a resource server.

5.3.1.2 Kerberos exchanges

The Kerberos protocol consists of several sub-protocols (or phases, or exchanges): the authentication service exchange, ticket granting service exchange, client/server exchange. The Kerberos authentication process is illustrated in Fig. 5.20.

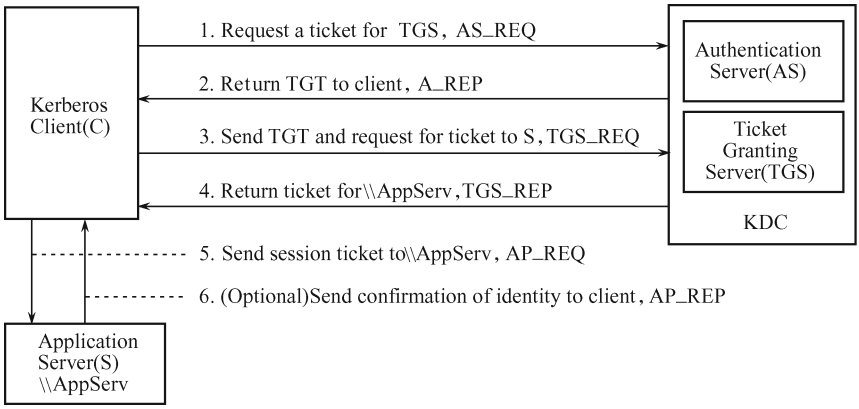


Fig. 5.20 The Kerberos protocol message exchanges.

Phase 1 Authentication service exchange:

— Message 1: The client sends a request AS_REQ to the authentication server (AS) requesting a Ticket Granting Ticket (TGT) to the Ticket Granting Server (TGS).

— Message 2: AS looks up the client and server principals named in the AS_REQ in its database, extracting their respective keys, then generates a "random" session key ($k_{c,tgs}$) for use between the client and the TGS. AS creates and sends the client a TGT which includes the client part and the TGS part. The part of TGT for the client including $k_{c,tgs}$ is encrypted under the client's secret key, and the part of TGT for TGS including $k_{c,tgs}$ is encrypted under the long-term secret key between the AS and the TGS.

Phase 2 Ticket granting service exchange:

— Message 3: The client decrypts the encrypted part using its secret key, verifies client's sending nonce (to detect replays) and recovers the session key $k_{c,tgs}$, then uses $k_{c,tgs}$ to create an authenticator containing the user's name, IP address and a timestamp. The client sends this authenticator, along with the TGT, to the TGS, requesting access to the application server S.

— Message 4: The TGS decrypts the TGT, then uses $k_{c,tgs}$ inside the TGT to verify the user's name, IP address and the timestamp in the authenticator. If everything matches, then the TGS generates a "random" new session key ($k_{c,s}$) for the client and the application server. The Kerberos

database is queried to retrieve the record for the requested server, and the TGS creates and sends the client a new ticket. This ticket is also made of the client part and the S part. The part for the client encrypts the $k_{c,s}$ using $k_{c,tgs}$, and the part for S encrypts the $k_{c,s}$ with the network address, the client's name, the server's name, the time of initial authentication and an expiration time using the long-term secret key $K_{s,tgs}$ between the S and the TGS.

Phase 3 Client/server exchange:

— Message 5: The client decrypts the encrypted part using $k_{c,tgs}$, verifies client's sending nonce and recovers the session key $k_{c,s}$, then uses $k_{c,s}$ to create a new authenticator containing the user's name, IP address and a timestamp. The client sends the received encrypted session ticket and the encrypted new authenticator to the application server S.

— Message 6: The application server decrypts and checks the ticket, authenticator, client address and timestamp. For applications that require two-way authentication, the target server returns a message consisting of the timestamp plus 1, encrypted with $k_{c,s}$.

Up to now, the application server **S** and the client **C** have authenticated the opponent party is who he claims to be, and the two now shared an encryption key $k_{c,s}$ for future secure communications. The user ID and password are secure since they are never sent over the network.

5.3.2 Basic Kerberos network authentication service

Kerberos 5, the most recent version, is introduced in the early 1990s[6, 27], and it is available for all major operating systems: Microsoft Windows 2000, Microsoft Windows server 2003, and many UNIX and UNIX-like operating systems, including FreeBSD, Apple's Mac OS X, Red Hat Enterprise Linux 4, Sun's Solaris, IBM's AIX, HP's OpenVMSLinux[28]. Especially, Microsoft bases its Kerberos implementation on the standard defined in RFC 1510. Fig. 5.21 shows the Message exchanges of Kerberos V:

Message 1 $C \rightarrow AS:$ $\text{Options}||\text{ID}_c||\text{Realm}_c||\text{ID}_{tgs}||\text{Times}||\text{Nonce}_1$

Message 2 $AS \rightarrow C:$ $\text{ID}_c||\text{Realm}_c||\text{Ticket}_{tgs}||\{\text{ID}_{tgs}||\text{Realm}_{tgs}||k_{c,tgs}||\text{Times}||\text{Nonce}_1\}_{K_c}$

Message 3 $C \rightarrow TGS:$ $\text{Options}||\text{ID}_s||\text{Realm}_s||\text{Times}||\text{Nonce}_2||\text{Ticket}_{tgs}||\text{Authenticator}_{c_1}$

Message 4 $TGS \rightarrow C:$ $\text{Realm}_c||\text{ID}_c||\text{Ticket}_s||\{k_{c,s}||\text{Times}||\text{Nonce}_2||\text{ID}_s||\text{Realm}_s\}_{k_{c,tgs}}$

Message 5 $C \rightarrow S:$ $\text{Options}||\text{Ticket}_s||\text{Authenticator}_{c_2}$

Message 6 $S \rightarrow C:$ $\{C_{time}||[\text{Subkey}]||[\text{Seq}]\}_{k_{c,s}}$

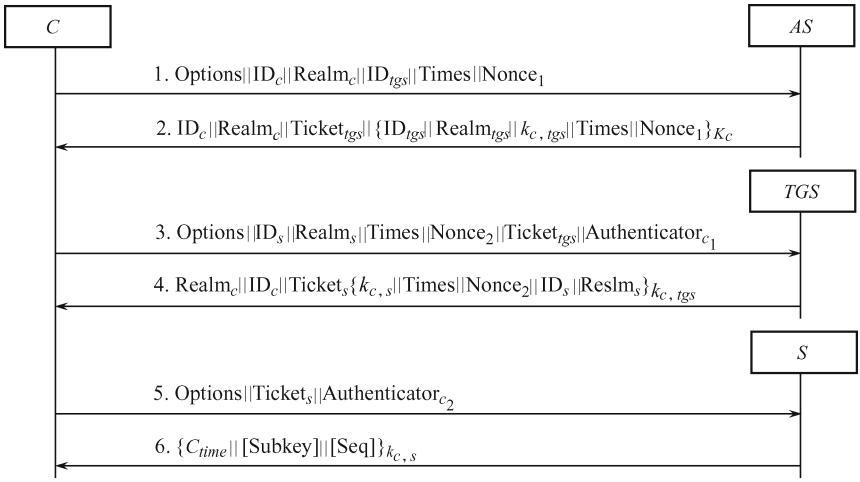


Fig. 5.21 Message exchanges of domain authentication based on Kerberos.

where

$\text{Ticket}_{tgs} = \{\text{ID}_{tgs}||\text{Realm}_{tgs}||\text{Flags}||k_{c,tgs}||\text{Realm}_c||\text{ID}_c||\text{Times}\}_{K_{as,tgs}}$,

$\text{Authenticator}_{c_1} = \{\text{ID}_c||\text{Realm}_c||\text{C}_{time}\}_{k_{c,tgs}}$,

$\text{Ticket}_s = \{\text{ID}_s||\text{Realm}_s||\text{Flags}||k_{c,s}||\text{Realm}_c||\text{ID}_c||\text{Times}\}_{K_{s,tgs}}$,

$\text{Authenticator}_{c_2} = \{\text{ID}_c||\text{Realm}_c||\text{C}_{time}||[\text{Subkey}]||[\text{Seq}]\}_{k_{c,s}}$

The fields in the above messages are:

Options: the client may specify a number of options in the initial request. Among these options are whether preauthentication is to be performed; whether the requested ticket is to be renewable, proxiable, or forwardable; whether it should be postdated or allow postdating of derivative tickets, etc.

ID_c, ID_{tgs} or ID_s: it is the identity of the client user, the ticket granting server or the application server. It is a uniquely named client or server instance that participates in a network communication.

Realm_c, Realm_{tgs} or Realm_s: it indicates the realm of the client user, the *TGS* server or the application server respectively.

Time consists of three parts:

from: the desired start time for the ticket;

till: the requested expiration time;

rtime: requested renew-till time.

Nonce_1 or Nonce_2: it is a random value generated by the client to assure the response of freshness. If the same number is included in the encrypted response from the *KDC*, it provides evidence that the response is fresh and has not been replayed by an attacker.

Ticket_{tgs}: it is a ticket granting ticket to obtain service-granting ticket, which is received from *AS*.

K_c: it is a long-term key between the client and the authentication server, which is traditionally derived from a password chosen by the user.

$k_{c,tgs}$: a temporary key for secure communication between the client and the ticket granting server.

Authenticator$_{c_1}$: it is the authenticator that contains plaintext encrypted under $k_{c,tgs}$, hence it proves that the client knows the temporary key $k_{c,tgs}$.

$k_{c,s}$: a temporary session key for secure communication between the client and the application server.

Authenticator$_{c_2}$: it is the authenticator that contains plaintext encrypted under $k_{c,s}$, hence it proves that the client knows the session key $k_{c,s}$.

[...]: it indicates an optional field.

Subkey: it contains the client's choice for an encryption key which is to be used to protect this specific application session. If this field is left out the session key from the ticket will be used.

Seq: it is a sequence number used to detect replays. The initial sequence number should be random and uniformly distributed across the full space of possible sequence numbers, so that it cannot be guessed by an attacker.

C$_{time}$: it contains the current time on the client's host.

5.3.3 Security analysis of Kerberos based on trusted freshness

Example 5.12 Figure 5.22 illustrates the security analysis of Kerberos V5 based on trusted freshness. For ease of exposition of the mutual authentication and key establishment idea in the Kerberos Authentication Protocol, only some mandatory protocol messages will be presented and some minute details are omitted to avoid obscuration.

Message 1	$C \rightarrow AS:$	C, TGS, T, N_1
Message 2	$AS \rightarrow C:$	$C, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_{K_c}$
Message 3	$C \rightarrow TGS:$	$S, T, N_2, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}},$ $\{C, Client\_time\}_{k_{c,tgs}}$
Message 4	$TGS \rightarrow C:$	$C, \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$
Message 5	$C \rightarrow S:$	$\{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{C, Client\_time\}_{k_{c,s}}$
Message 6	$S \rightarrow C:$	$\{Client\_time\}_{k_{c,s}}$

Notation

C denotes the client, AS denotes the authentication server, TGS denotes the ticket granting server and S denotes the application server.

T and *Client_time* are the timestamps chosen by the client for the ticket.

N_1 and N_2 are nonces randomly chosen by the client for AS_REQ and TGS_REQ respectively.

K_c is a long-term key between the client C and the AS, which is usually

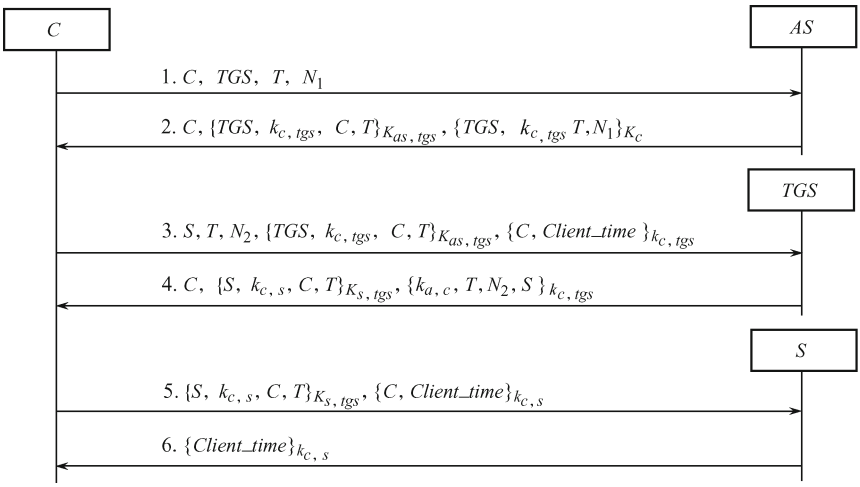


Fig. 5.22 Messages of the Kerberos protocol.

derived from the user's password.

$K_{as,tgs}$ and $K_{s,tgs}$ are the long-term keys between AS and TGS, between S and TGS respectively.

$k_{c,tgs}$ is a temporary key randomly chosen by the authentication server AS for the temporary session between C and TGS.

$k_{c,s}$ is a new session key randomly chosen by the ticket granting server TGS for this protocol run between C and the application server S.

Premise

The authentication server AS and the client C know their shared key K_c and also K_c^{-1}. $K_{as,tgs}$ is confidential and only known by the authentication server AS and the ticket granting server TGS; $K_{s,tgs}$ is confidential and only known by the authentication server S and the ticket granting server TGS. N_1 and N_2 are randomly chosen nonces. $k_{c,tgs}$ and $k_{c,s}$ are randomly chosen temporary keys for this protocol run.

Protocol actions

1) In Message 1, the client C starts the protocol run by sending the identities of the client and the ticket granting server (from whom C desires a TGT), a timestamp T and a randomly chosen new nonce N_1 by C.

2) In Message 2, the authentication server AS randomly chooses a temporary key $k_{c,tgs}$ for the subsequent communication between C and TGS, then generates a granting ticket $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ to TGS using the long-term key $K_{as,tgs}$ between AS and TGS to show that it is AS who has sent the ticket. AS also sends C the temporary key $k_{c,tgs}$ using the long-term key K_c to keep $k_{c,tgs}$ secret. K_c is usually derived from the user's password. This is the only time that this long-term key is used in a standard Kerberos

run because later exchanges use freshly generated keys.

3) Upon receiving Message 2, the client C may undertake the Ticket-Granting exchange. It decrypts $\{TGS, k_{c,tgs}, T, N_1\}_{K_c}$ using the key K_c, verifies the correction of N_1, and gets the temporary key $k_{c,tgs}$.

4) In Message 3, the client C generates the authenticator $\{C, Client\_time\}_{k_{c,tgs}}$ to prove that the client knows the key $k_{c,tgs}$. The encrypted timestamp prevents an eavesdropper from recording both the ticket and the authenticator to replay them later. C also forwards the ticket $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ received in Message 2. Encrypting the authenticator in the session key $k_{c,tgs}$ proves that it is generated by a party possessing the session key. Since no one except C and the server TGS knows the session key (it is never sent over the network in the clear), this guarantees the identity of the client C.

5) Upon receiving Message 3, the ticket granting server TGS decrypts $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ using the long-term key $K_{as,tgs}$, verifies the correction of the timestamp T, and gets the temporary key $k_{c,tgs}$. TGS also verifies the authenticator $\{C, Client\_time\}_{k_{c,tgs}}$.

6) In Message 4, TGS randomly chooses a new session key $k_{c,s}$ for the application between S and C, then generates a service ticket $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ to S using the long-term key $K_{s,tgs}$ to show that it is TGS who has sent the ticket. TGS also sends C the session key $k_{c,s}$ using the negotiated temporary key $k_{c,tgs}$ to keep $k_{c,s}$ secret.

7) Upon receiving Message 4, C decrypts $\{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$ using $k_{c,tgs}$, verifies the correction of the nonce N_2, and gets the temporary key $k_{c,s}$.

8) In Message 5, the client C generates the authenticator $\{C, Client\_time\}_{k_{c,s}}$ to prove that the client knows the key $k_{c,s}$. C also forwards the service ticket $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ received in Message 4 to the application server S.

9) Upon receiving Message 5, the application server S decrypts $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ using the long-term key $K_{s,tgs}$, verifies the correction of the timestamp T, and gets the new session key $k_{c,s}$. S also verifies the authenticator $\{C, Client\_time\}_{k_{c,s}}$.

10) In Message 6, S sends the encryption $\{Client\_time\}_{k_{c,s}}$ to show the ownership of $k_{c,s}$ by the identity S.

Successful execution should achieve mutual authentication and convince both C and S that $k_{c,s}$ is a secure new session key between C and S.

Protocol security analysis

The security properties related to mutual authentication and key establishment idea will be indicated in detail, and other security properties, such as the security of $k_{c,tgs}$ from the point of view of S, and the security of $k_{c,s}$ from the point of view of TGS, will be omitted.

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_1, and C also believes that N_1 is open. AS could not draw any useful assurance from Message 1 since there is

not any trusted freshness identifier from the point of view of AS.

2) Upon receiving Message 2, from Lemma 4.2, C has the confidential assurance of $k_{c,tgs}$. From Lemma 4.3, C has the freshness assurance of the temporary key $k_{c,tgs}$ since $k_{c,tgs}$ is sent to C together with C's trusted freshness N_1. From Lemma 4.4, C has the association assurance of N_1 and $k_{c,tgs}$ with C, since only C could get $k_{c,tgs}$ from the encryption $\{TGS, k_{c,tgs}, T, N_1\}_{K_c}$ using the long-term key K_c between C and AS. From Lemma 4.4, C also has the association assurance of N_1 and $k_{c,tgs}$ with TGS since the identity of TGS is explicitly indicated in $\{TGS, k_{c,tgs}, T, N_1\}_{K_c}$ of Message 2.

3) In Message 3, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_2, and C also believes that N_2 is open.

4) Upon receiving Message 3, from Lemma 4.2 and Lemma 4.3, TGS has the confidential and freshness assurances of the temporary key $k_{c,tgs}$ based on the timestamp T. From Lemma 4.4, TGS has the association assurance of $k_{c,tgs}$ with both TGS and C since the identities of both TGS and C are explicitly indicated in Message 3 and the encryption $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ could only be generated by the authentication server AS using the shared long-term key $K_{as,tgs}$. From Lemma 4.1, TGS has the liveness assurance of C based on the timestamp $Client\_time$, since it must be C who has just generated the authenticator $\{C, Client\_time\}_{k_{c,tgs}}$ using the shared temporary key $k_{c,tgs}$.

5) Upon receiving Message 4, from Lemma 4.2, C has the confidential assurance of $k_{c,s}$. From Lemma 4.3, C has the freshness assurance of the temporary key $k_{c,s}$ since $k_{c,s}$ is sent to C together with C's trusted freshness N_2. From Lemma 4.4, C has the association assurance of $k_{c,s}$ with C and S since the identity of S is explicitly indicated in Message 4 and the encryption $\{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$ is generated under the shared temporary key $k_{c,tgs}$ between C and TGS. From Lemma 4.1, C has the liveness assurance of TGS based on the trusted freshness N_2, since it must be TGS who has just generated the encryption $\{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$ using $k_{c,tgs}$.

6) Upon receiving Message 5, from Lemma 4.2 and Lemma 4.3, S has the confidential and freshness assurances of the temporary key $k_{c,s}$ based on the timestamp T. From Lemma 4.4, S has the association assurance of $k_{c,s}$ with both C and S since the identities of both C and S are explicitly indicated in Message 5 and the encryption $\{S, k_{c,s}, C, T\}_{K_{s,tgs}}$ could only be generated by TGS using the shared long-term key $K_{s,tgs}$. From Lemma 4.1, S has the liveness assurance of C based on the timestamp $Client\_time$, since it must be C who has just generated the authenticator $\{C, Client\_time\}_{k_{c,s}}$ using the shared session key $k_{c,s}$.

7) Upon receiving Message 6, from Lemma 4.1, C has the liveness assurance of S based on the timestamp $Client\_time$, since it must be S who has just generated the authenticator $\{Client\_time\}_{k_{c,s}}$ using the shared session key $k_{c,s}$.

Table 5.8 indicates the analyzing result of Kerberos protocol. Upon termi-

nation of the protocol run, S believes that C is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both C and S, while C believes that S is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both C and S. That is, the analyzed Kerberos authentication protocol has achieved the security objects of mutual authentication and secure key establishment.

Table 5.8 Security analysis of the Kerberos protocol

	C							TGS		S	
	AS	TGS	S	N_1	$k_{c,tgs}$	N_2	$k_{c,s}$	C	$k_{c,tgs}$	C	$k_{c,s}$
Message 1				$01\#$							
Message 2	1			$01C\,TGS$	$11C\,TGS$						
Message 3						$01\#$		1	$11C\,TGS$		
Message 4		1				$01CS$	$11CS$				
Message 5										1	$11CS$
Message 6			1								
End of run			1				$11CS$			1	$11CS$

5.3.4 Public-key Kerberos

PKINIT introduces a new trust model in which the KDC is not the first entity to identify the users (as is the case for classical Kerberos). Before KDC authentication, users are identified by the certification authority CA in order to obtain a certificate. In this new model the users and the KDC obviously both need to trust the same CA.

Public-Key Kerberos PKINIT[26], which is included in Windows 2000 and Windows Server 2003, is an extension to Kerberos 5 that uses public-key cryptography for initial authentication. That is, PKINIT modifies the authentication service exchange but not other parts of the basic Kerberos 5 protocol to avoid shared secrets between a client and an authentication server. PKINIT enables the smart card logon process to a Windows 2000 or later domain. PKINIT allows a client's master key to be replaced with its public-key credentials in the Kerberos Authentication[27].

In traditional Kerberos 5 protocol, the long-term shared key in the authentication service exchange is typically derived from a password, which limits the strength of the authentication to the user's ability to choose and remember good passwords, while PKINIT uses public-key cryptography and thus avoids this problem. Furthermore, if a public-key infrastructure (PKI) is already in place, PKINIT allows network administrators to use it rather than to expend additional effort to manage users' long-term keys needed for traditional Kerberos. However, this protocol extension adds complexity to Kerberos as it retains symmetric encryption in the later exchanges but relies on asymmetric encryption, digital signatures, and corresponding certificates

in the first exchange. PKINIT is intended to add flexibility, security and administrative convenience by introducing public-key cryptography.

In PKINIT, the client C and the authentication server AS each possesses an independent public/secret key pair, K_C and K_C^{-1} for C, K_S and K_S^{-1} for S, respectively. Certificate sets $Cert_C$ and $Cert_S$ issued by a PKI independent of Kerberos are used to testify the binding between each principal and his purported public-key. C and AS need only maintain the public-keys of a few known certification authorities CA within the PKI. Hence, AS need not maintain keys individually shared with each client, and dictionary attacks are defeated as user-chosen passwords are replaced with automatically generated asymmetric keys. Since very few users would be able to remember a random public/private key pair, PKINIT authentication is typically used with smartcard, where the keys and certificate chains are stored in a smartcard that the user swipes in a reader at login time or in the user's hard drive.

PKINIT is supported by Kerberized versions of Microsoft Windows, including Windows 2000 Professional and Server, Windows XP, and Windows Server 2003[29]; it has also been included in Heimdal since 2002[30]. PKINIT is not yet supported in the MIT reference implementation.

The manner in which PKINIT works depends on both the protocol version and the mode invoked. "PKINIT-n" is used to refer to the protocol as specified in the nth draft revision and "PKINIT" for the generic protocol[31]. PKINIT can operate in two modes: Diffie-Hellman (DH) mode and public-key encryption mode.

5.3.4.1 PKINIT public-key encryption mode

In public-key encryption mode, the key pairs are used for both signature and encryption. The latter is designed to (indirectly) protect the confidentiality of AK, while the former ensures its integrity.

Phase 1 Authentication service exchange

The abstract structure of the authentication service exchange in public-key encryption PKINIT – 26 is given:

$$\begin{aligned}
&\text{Message 1} \quad C \rightarrow AS: \; Cert_C, \{t_c, n_2\}_{K_c^{-1}}, C, TGS, T, N_1 \\
&\text{Message 2} \quad AS \rightarrow C: \; \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C, \\
&\qquad\qquad \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k
\end{aligned}$$

The last part of Message 1, "C, TGS, T, N_1", is exactly as in basic Kerberos 5, containing the client's name, the name of the TGS from which the client wants to get a TGT, a timestamp and a nonce. $Cert_C, \{t_c, n_2\}_{K_c^{-1}}$ is added by PKINIT and contains the client's certificates $Cert_C$ and client's signature $\{t_c, n_2\}_{K_c^{-1}}$ over a timestamp t_c and another nonce n_2. The nonces n_2 and timestamp t_c are generated by C specifically for this request.

Message 2 is more complex than in basic Kerberos. The last part of Message 2 "$C, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$, $\{TGS, k_{c,tgs}, T, N_1\}_k$" is very similar to AS's reply in basic Kerberos; the difference is that the symmetric key k

which is used to protect $k_{c,tgs}$ is now freshly generated by AS and not a long-term shared key that is usually derived from the client's password. The ticket-granting ticket TGT $\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}$ is encrypted with a long-term key $K_{as,tgs}$ shared between AS and the ticket granting server TGS; the TGT contains TGS's name, $k_{c,tgs}$, C's name, and AS's local time T. The message part encrypted under the freshly generated symmetric key k includes TGS's name, $k_{c,tgs}$, AS's local time T, and the nonce N_1 from the request. To ensure the ability to learn k of C, PKINIT adds the message part $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ in Message 2. This encryption is encrypted under K_c, and it contains AS's certificate $Cert_{AS}$ and the signature $\{k, n_2\}_{K_{as}^{-1}}$ by AS, hence only C can get the freshly generated key k.

PKINIT leaves the subsequent exchanges of Kerberos unchanged.

Example 5.13 Here is the security analysis of the Kerberos public-key encryption PKINIT based on trusted freshness. The whole message exchanges of PKINIT mode are illustrated in Fig. 5.23.

Message 1 $C \rightarrow AS: \; Cert_C, \{t_c, n_2\}_{K_c^{-1}}, C, TGS, T, N_1$

Message 2 $AS \rightarrow C: \; \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C,$
$\{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k$

Message 3 $C \rightarrow TGS: S, T, N_2, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}},$
$\{C, Client\_time\}_{k_{c,tgs}}$

Message 4 $TGS \rightarrow C: C, \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$

Message 5 $C \rightarrow S: \; \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{C, Client\_time\}_{k_{c,s}}$

Message 6 $S \rightarrow C: \; \{Client\_time\}_{k_{c,s}}$

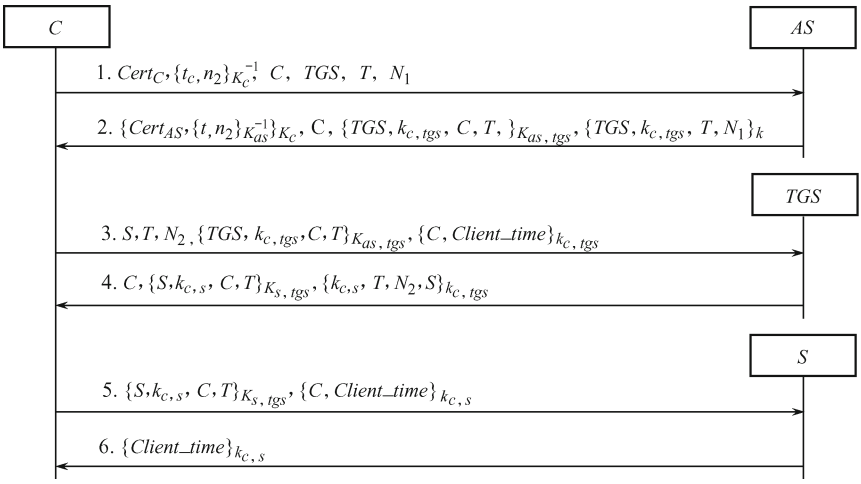


Fig. 5.23 Kerberos message exchanges in Public-key encryption PKINIT mode.

Notation

K_c and K_c^{-1} are the public-key and the private key for C, and K_{as} and K_{as}^{-1} are for AS.

$Cert_C$ and $Cert_{AS}$ denote the client's certificate and the authentication server's certificate respectively.

t_c is a timestamp from the client C and n_2 is a nonce randomly chosen by C.

k is a symmetric key randomly chosen by AS to protect the temporary key $k_{c,tgs}$.

Other notations are the same as in the basic Kerberos protocol, hence omitted.

Premise

Each principal knows the public-key of the trusted certificate authority CA to get K_c or K_{as} from $Cert_C$ or $Cert_{AS}$. Each principal knows the key pair of himself, that is, K_c and K_c^{-1} for C, K_{as} and K_{as}^{-1} for AS.

Protocol actions

1) In Message 1, the client C randomly chooses the nonces n_2 and N_1, and signs n_2 and timestamp t_c using C's private key K_c^{-1}, then C starts the protocol run by sending the certificate of C, the signature $\{t_c, n_2\}_{K_c^{-1}}$, the identities of C and TGS, a timestamp T and N_1.

2) Upon receiving Message 1, AS gets the public-key of C from $Cert_C$, decrypts $\{t_c, n_2\}_{K_c^{-1}}$ using C's public-key K_c, verifies the correction of timestamp t_c, and gets the nonce n_2. The KDC will then query the Active Directory for a mapping between the certificate $Cert_C$ and a Windows account. If it finds a mapping, it will issue a TGT to the corresponding account.

3) In Message 2, the authentication server AS randomly chooses a symmetric key k and a temporary key $k_{c,tgs}$ for the subsequent communication between TGS and C, then AS signs k and n_2 using AS's private key K_{as}^{-1}. AS makes response to C with $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ to keep the freshly generated key k confidential by encrypting under C's public-key K_c. The last part of Message 2 is very similar to AS's reply in basic Kerberos, hence omitted.

Other messages are the same as the basic Kerberos protocol.

Protocol security analysis

1) In Message 1, from Lemma 4.2 and Lemma 4.3, C has the freshness assurance of the randomly chosen nonce N_1 and n_2, and C also believes that N_1 and n_2 are open. TGS could not draw any useful assurance from Message 1.

2) Upon receiving Message 2, from Lemma 4.2, C has the confidential assurance of k and $k_{c,tgs}$ since only C could decrypt $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ using C's private key K_c^{-1} to get k and then $k_{c,tgs}$. From Lemma 4.3, C has the freshness assurance of the temporary key k since k is sent to C together

with C's trusted freshness n_2. Similarly, C has the freshness assurance of the temporary key $k_{c,tgs}$ since $k_{c,tgs}$ is sent to C together with C's trusted freshness N_1. From Lemma 4.4, C has the association assurance of k and n_2 with AS, since AS has signed k and n_2 using AS's private key K_{as}^{-1}. However, from the point of view of C, k and n_2 are not associated with TGS or C since if an adversary is a legal user, then the adversary could impersonate C and generate the encryption $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ using the public K_c. Note that $k_{c,tgs}$ is not associated with TGS since k is not associated with C, although the identity of TGS is explicitly indicated in Message 2.

3) In Message 3, C has the same security beliefs as those in the basic Kerberos.

4) Upon receiving Message 3, TGS has the same security beliefs as those in the basic Kerberos.

5) Upon receiving Message 4, from Lemma 4.2, C has the confidential assurance of the temporary key $k_{c,s}$ since it is encrypted under the temporary key $k_{c,tgs}$. From Lemma 4.3, C has the freshness assurance of $k_{c,s}$ since $k_{c,s}$ is sent to C together with C's trusted freshness N_2.

6) Upon receiving Message 5, S has the same security beliefs as those in the basic Kerberos.

7) Upon receiving Message 6, C could not authenticate the liveness of S since C is not sure whether $k_{c,s}$ is between C and S, hence C is not sure whether the fresh message $\{Client\_time\}_{k_{c,s}}$ is from S or not.

Table 5.9 indicates the analyzing result of Public-key Kerberos protocol. Upon termination of the protocol run, S believes that C is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both S and C, while C only believes that $k_{c,s}$ is confidential and fresh, but C is not sure whether $k_{c,s}$ is between C and S or not.

Table 5.9 Security analysis of the PKINIT Public-key Kerberos protocol

	C									TGS		S	
	AS	TGS	S	k	n_2	N_1	$k_{c,tgs}$	N_2	$k_{c,s}$	C	$k_{c,tgs}$	C	$k_{c,s}$
Message 1					01#	01#							
Message 2	1			11AS	01AS	01#	11#						
Message 3								01#		1	11C TGS		
Message 4									11#				
Message 5												1	11CS
Message 6													
End of run									11#			1	11CS

Example 5.14 From the absence of the association of $k_{c,s}$ with C and S in the point of view of C, there exists an attack[31] as illustrated in Fig. 5.24.

Message 1 $\quad C \rightarrow I(AS): \; Cert_C, \{t_c, n_2\}_{K_c^{-1}}, C, TGS, T, N_1$

Message 1′ $\quad I \rightarrow AS: \; Cert_I, \{t_c, n_2\}_{K_i^{-1}}, I, TGS, T, N_1$

Message 2′	$AS \rightarrow I:$	$\left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_i}, I,$ $\{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k$
Message 2	$I(AS) \rightarrow C:$	$\left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C,$ $\{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k$
Message 3	$C \rightarrow I(TGS):$	$S, T, N_2, \{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}},$ $\{C, Client\_time\}_{k_{c,tgs}}$
Message 3′	$I \rightarrow TGS:$	$S, T, N_2, \{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}},$ $\{I, Client\_time\}_{k_{c,tgs}}$
Message 4′	$TGS \rightarrow I:$	$I, \{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$
Message 4	$I(TGS) \rightarrow C:$	$C, \{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}}$
Message 5	$C \rightarrow I(S):$	$\{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{C, Client\_time\}_{k_{c,s}}$
Message 5′	$I \rightarrow S:$	$\{S, k_{c,s}, I, T\}_{K_{s,tgs}}, \{I, Client\_time\}_{k_{c,s}}$
Message 6	$S \rightarrow I:$	$\{Client\_time\}_{k_{c,s}}$
Message 6′	$I(S) \rightarrow C:$	$\{Client\_time\}_{k_{c,s}}$

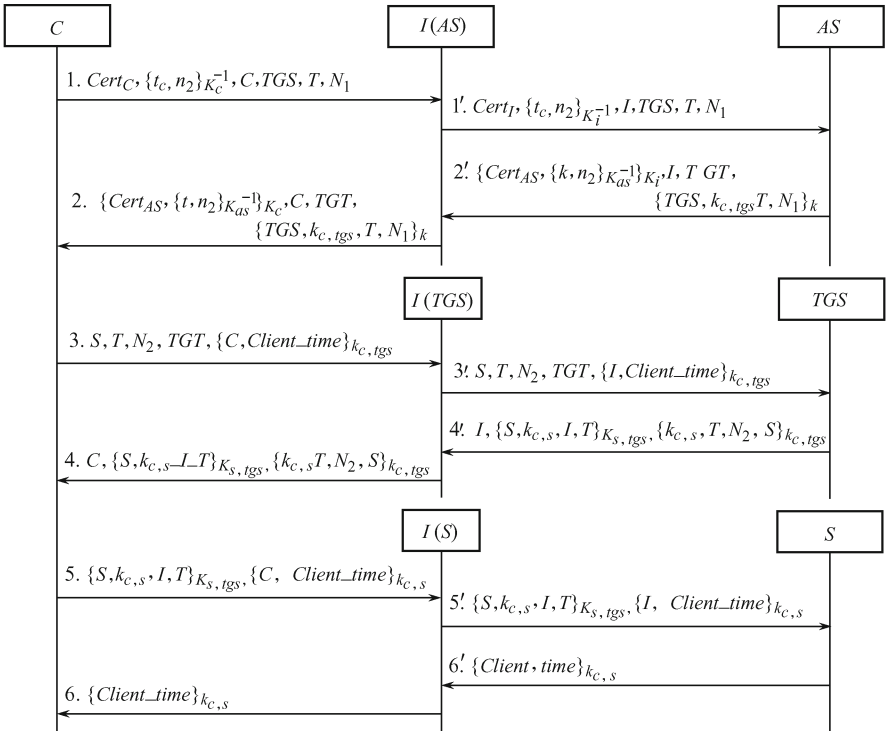


Fig. 5.24 An attack on the Kerberos PKINIT Public-key Encryption mode.

Notation

I denotes the adversary, K_i and K_i^{-1} are the public-key and the private key for I, and $Cert_I$ denotes the adversary's certificate.

$I(TGS)$ or $I(S)$ is the adversary I impersonating TGS or S.

n_2 is a nonce randomly chosen by the client for AS_REQ in PKINIT public-key encryption mode.

$TGT = \{TGS, k_{c,tgs}, I, T\}_{K_{as,tgs}}$ is a ticket granting ticket.

Other notations are the same as the original Keberos PKINIT protocol.

Premise

The adversary is a legal user, Other premises are the same as the original Keberos protocol.

Protocol actions

1) In Message 1, the client C starts a new protocol run. I intercepts Message 1, replaces $Cert_C$ with $Cert_I$, gets $\{t_c, n_2\}$ using C's public-key and encrypts it using I's private key, then sends Message $1'$ to AS.

2) Upon receiving Message $1'$, AS responds to I just as AS does in basic Keberos.

3) Upon receiving Message $2'$, I gets $\{k, n_2\}_{K_{as}^{-1}}$ and k using I's private key and AS's public-key, hence I can get $k_{c,tgs}$ using k.

4) In Message 2, I constructs $\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\}_{K_c}$ from $\{k, n_2\}_{K_{as}^{-1}}$ using C's public-key, replaces the identity of I with C.

5) In Message 3, C responds as usual. I intercepts Message 3, replaces the identity of C with I in $\{C, Client\_time\}$, and then sends $\{I, Client\_time\}_{k_{c,tgs}}$ to TGS.

6) Upon receiving Message $3'$, TGS responds to I just as TGS does in basic Keberos.

7) Upon receiving Message $4'$, I gets $k_{c,s}$ using $k_{c,tgs}$.

8) In Message 4, I replaces the identity of I with C in Message $4'$, and then forwards Message 4.

9) Up to now, both C and I know the session key $k_{c,s}$, hence they can complete the subsequence protocol run as usual.

Upon termination of the attack on the Kerberos PKINIT public-key encryption, the adversary I causes C to have false beliefs: C has completed a successful protocol run with S, and is sharing a new session key $k_{c,s}$ with S, whereas in fact, S knows nothing about the key establishment with C, and thinks that it has been talking with the adversary I, and sharing $k_{c,s}$ with I. From now on, C will send subsequent sensitive data encrypted under $k_{c,s}$ which is also known by I.

5.3.4.2 PKINIT Diffie-Hellman mode

In Diffie-Hellman (DH) mode, the key pairs (K_c and K_c^{-1} for C, K_{as} and K_{as}^{-1} for AS) are used to provide digital signature support for an authenticated

Diffie-Hellman key agreement which is used to protect the temporary key $k_{c,tgs}$ between the client and the TGS. A variant of this mode allows the reuse of previously generated shared secrets $k_{c,tgs}$.

The following abstract description leaves out a number of fields which are of no significance with respect to our analysis. We invite the interested reader to consult the specifications[26]. The simplified authentication service exchange in Diffie-Hellman PKINIT-26 is:

$$\begin{array}{lll}
\text{Message 1} & C \rightarrow AS: & \left\{Cert_C, \{t_c, n_2\}_{K_c^{-1}}\right\}_{K_{as}}, C, TGS, T, N_1 \\
\text{Message 2} & AS \rightarrow C: & \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C, \\
 & & \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k
\end{array}$$

Example 5.15 The whole message exchanges of PKINIT Diffie-Hellman mode are illustrated in Fig. 5.25.

$$\begin{array}{lll}
\text{Message 1} & C \rightarrow AS: & \left\{Cert_C, \{t_c, n_2\}_{K_c^{-1}}\right\}_{K_{as}}, C, TGS, T, N_1 \\
\text{Message 2} & AS \rightarrow C: & \left\{Cert_{AS}, \{k, n_2\}_{K_{as}^{-1}}\right\}_{K_c}, C, \\
 & & \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \{TGS, k_{c,tgs}, T, N_1\}_k \\
\text{Message 3} & C \rightarrow TGS: & S, T, N_2, \{TGS, k_{c,tgs}, C, T\}_{K_{as,tgs}}, \\
 & & \{C, Client_time\}_{k_{c,tgs}} \\
\text{Message 4} & TGS \rightarrow C: & C, \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{k_{c,s}, T, N_2, S\}_{k_{c,tgs}} \\
\text{Message 5} & C \rightarrow S: & \{S, k_{c,s}, C, T\}_{K_{s,tgs}}, \{C, Client_time\}_{k_{c,s}} \\
\text{Message 6} & S \rightarrow C: & \{Client_time\}_{k_{c,s}}
\end{array}$$

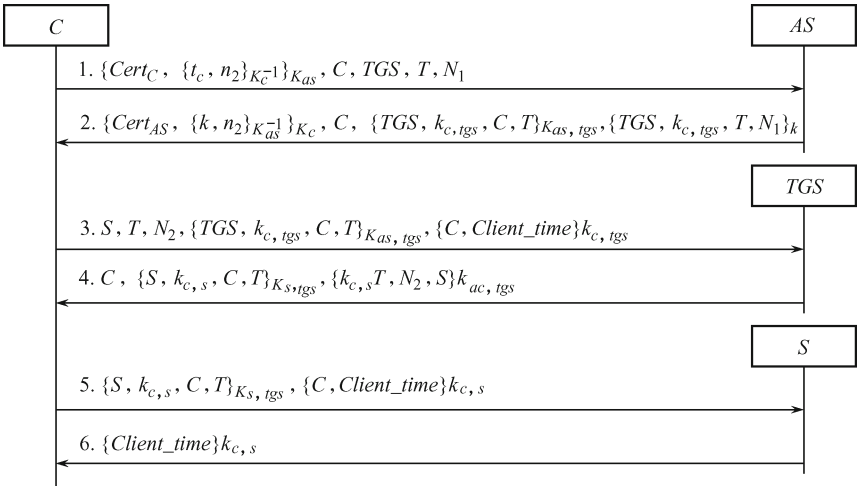


Fig. 5.25 Kerberos message exchanges in PKINIT Diffie-Hellman mode.

Table 5.10 indicates the security analysis result of PKINIT Diffie-Hellman

mode. The security analysis details are left to the interested reader.

Table 5.10 Security analysis of the Kerberos PKINIT Diffie-Hellman protocol

	C									TGS		S	
	AS	TGS	S	k	n_2	N_1	$k_{c,tgs}$	N_2	$k_{c,s}$	C	$k_{c,tgs}$	C	$k_{c,s}$
Message 1					01#	01#							
Message 2	1			11 C AS	01 C AS	01 C TGS	11 C TGS						
Message 3								01#		1	11 C TGS		
Message 4		1							11 C S				
Message 5												1	11 C S
Message 6			1										
End of run			1						11 C S			1	11 C S

Upon termination of the protocol run, S believes that C is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both S and C; at the same time, C believes that S is present, and the new session key $k_{c,s}$ is confidential, fresh, and associated with both S and C.

References

[1] Tanenbaum AS (2001) Computer Networks, 3rd edn. Prentice Hall, New Jersey.

[2] Freier AO, Karlton P, Kocher PC (1996) The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/draft302.txt. Accessed 29 Apr 2007

[3] Dierks T, Allen C (1999) the TLS Protocol Version 1.0, RFC 2246. http://tools.ietf.org/html/rfc2246. Accessed 21 May 2011

[4] Kaufman C (2005) Internet Key Exchange (IKEv2) Protocol, RFC 4306. http://tools.ietf.org/html/rfc4306. Accessed Dec 2005

[5] Meadows C (1999) Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In: Proceedings of 1999 IEEE Symposium on Security and Privacy, Oakland, 9 – 12 May 1999

[6] Neuman C (1993) The Kerberos Network Authentication Service (V5), RFC 1510. http://www.ietf.org/rfc/rfc1510.txt. Accessed 5 May 2011

[7] Neuman BC, Ts'o T (1994) Kerberos: an Authentication Service for Computer Networks. IEEE Communications Magazine 32(9): 33 – 38

[8] Ylonen T (1995) The SSH (secure shell) Remote Login Protocol, Internet-Draft. http://www.free.lp.se/fish/rfc.txt. Accessed 15 Nov 1995

[9] Ylonen T (2002) SSH Authentication Protocol, RFC4252. http://www.ietf.org/rfc/rfc4252.txt. Accessed 5 May 2011

[10] Ylonen T (2002) SSH Connection Protocol, RFC4254. http://www.ietf.org/rfc/rfc4254.txt. Accessed 5 May 2011

[11] Ylonen T (2002) SSH Protocol Architecture, RFC4251. http://www.ietf.org/rfc/rfc4251.txt. Accessed 5 May 2011

[12] Ylonen T (2002) SSH Transport Layer Protocol, RFC4253. http://www.ietf.org/rfc/rfc4253.txt. Accessed 5 May 2011

[13] Stallings W (2006) Cryptography and Network Security: Principles and Practice, 4th edn. Prentice Hall, New Jersey

[14] Ray M, Dispensa S (2009) Renegotiating TLS. http://www.phonefactor.com/sslgapdocs/Renegotia- ting_TLS.pdf. Accessed 5 May 2011

[15] Thayer R, Doraswamy N, Glenn R (1998) IP Security Document Roadmap, RFC2411. http://tools.ietf.org/html/rfc2411. Accessed Nov 1998

[16] Hoffman P (2005) Cryptographic Suites for IPsec, RFC4308. http://tools.ietf.org/html/rfc4308. Accessed 5 May 2011

[17] Kent S, Atkinson R (1998) IP Authentication Header, RFC2402. http://tools.ietf.org/html/rfc2402. Accessed 5 May 2011

[18] Kent S, Atkinson R (1998) IP Encapsulating Security Payload (ESP), RFC2406. http://tools.ietf.org/html/rfc2406. Accessed Nov 1998

[19] Harkins D, Carrel D (1998) The Internet Key Exchange Protocol (IKE), RFC 2409. http://www.ietf.org/rfc/rfc2409.txt. Accessed Dec 2005

[20] Kent S (2005) IP Authentication Header, RFC4302. http://tools.ietf.org/html/rfc4302. Accessed Dec 2005

[21] Maughan D, Schertler M, Schneider M, Turner J (1998) Internet Security Association and Key Management Protocol (ISAKMP). IETF RFC 2408. http://www.ietf.org/rfc/rfc2408. Accessed November 1998

[22] Orman H (1998) The OAKLEY Key Determination Protocol. IETF RFC 2412. http://www.ietf.org/rfc/rfc2412. Accessed November 1998

[23] Krawczyk H (1996) SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In: Proceedings of Symposium on Network and Distributed System Security (SNDSS '96), San Diego, 22 – 23 Feb 1996

[24] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[25] Lowe G (1996) Some new Attacks Upon Security Protocols. In: Proceedings of the 9th IEEE Computer Security Foundations Workshop, Kenmare, 10 – 12 Mar 1996

[26] Zhu L, Tung B (2006) Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), RFC4556. http://www.ietf.org/rfc/rfc4556.txt. Accessed 10 June 2010

[27] Neuman C, Yu T, Hartman S, Raeburn K (2005) The Kerberos Network Authentication Service (V5). http://www.ietf.org/rfc/rfc4120

[28] Wikipedia. Kerberos (protocol). http://en.wikipedia.org/wiki/Kerberos_(protocol). Accessed 5 Dec 2011

[29] Microsoft Security Bulletin MS05-042. http://www.microsoft.com/technet/security/bulletin/ms05-042.mspx. Accessed 9 Aug 2010

[30] Strasser M, Steffen A (2002) Kerberos PKINIT Implementation for Unix Clients. Technical Report, Zurich University of Applied Sciences Winterthur, Nov 2010

[31] Cervesato I, Jaggard AD, Scedrov A, Tsay JK, Walstad C (2008) Breaking and Fixing Public-key Kerberos. Journal Information and Computation 206(2 – 4): 402 – 424

6 Guarantee of Cryptographic Protocol Security

Abstract Some important provable security notions like indistinguishability, match conversation, authentication, etc. are briefly reviewed. The security definitions of UA-Secure, MA-Secure, UK-Secure and MK-Secure are specified based on the trusted freshness principle, and these formalization specifications are proved to be adequate for the intended security goals.

As we have witnessed in cryptographic literature, cryptographic protocols are notoriously error-prone. These protocols can be flawed in very subtle ways. It is widely agreed by researchers with different backgrounds that formal methods should be taken into the security analysis of cryptographic protocols.

The methodology provable security is introduced where the security could be proved under "standard" and well-believed complexity theoretic assumptions (e.g., the assumed intractability of factoring). The provable security method often entails providing (i) a definition of the security goal, (ii) a protocol, and (iii) a proof that the protocol meets its goal, assuming some standard complexity-theoretic assumption holds true.It is the opinion of many researchers that provable security should be in hand for all of the "basic" cryptographic primitives[1, 2].

In provable security field, some novel definitional ideas are achieved: Goldwasser, Micali et al. have suggested probabilistic encryption[3] and digital signatures[4], Blum – Micali and Yao suggested pseudorandom number generation[5, 6], Bellare, Rogaway et al. suggested authentication[2, 7].

The security goals discussed in this book involve unilateral entity authentication secure, mutual entity authentication secure, unilateral authenticated key secure and mutual authenticated key secure. The question of whether the security properties of a cryptographic protocol are adequate for a security goal or not will be answered in this chapter. Particularly we try to raise the security specification guarantees of unilateral entity authentication secure, mutual entity authentication secure, unilateral authenticated key secure and mutual authenticated key secure, similar to those primitives such as encryptions, pseudorandom generators, or digital signatures.

In this chapter, some important provable security notions like indistin-

guishability, authentication, etc., will be briefly reviewed first. Then, we try to formalize the security goals – UA-Secure, MA-Secure, UK-Secure and MK-Secure – of cryptographic protocols based on the trusted freshness principle, and prove that the formalization specifications are adequate for the intended security goals. In deed, the latter security goal MK-Secure, which is well known to applied cryptographers, is very useful to build secure distributed system.

6.1 Security definition of authentication

Bellare and Rogaway are the first researchers who propose a computational model for the security of authentication and authenticated key establishment protocols[1]. The idea of the definition of a mutual authentication in this model is simple but strong: any adversary effectively behaves as a trusted wire, if not a broken one. This simple idea is formalized via a notion of matching conversations. The idea of the definition of an authenticated key exchange is to keep the session key remaining protected. The adversary's inability is formalized to gain any helpful information about the session key along the lines of formalizations of security for probabilistic encryption[1, 7].

Four protocols are specifically discussed in the Bellare and Rogaway's model. Protocol MAP1 is a mutual authentication protocol for an arbitrary set of parties. Protocol MAP2 is an extension of MAP1, allowing arbitrary text strings to be authenticated along with its flows. Protocol AKEP1 is a simple authenticated key exchange which uses MAP2 to do the key distribution. Protocol AKEP2 is a particularly efficient authenticated key exchange which introduces the idea of "implicitly" distributing a key; its flows are identical to MAP1, but it accomplishes a key distribution all the same. The primitive required for all of these protocols is a pseudorandom function.

In practice, the pseudo-random function can be practically realized by a message authentication code in cipher-block-chaining mode of operation (CBC-MAC) or by a keyed cryptographic hash function (HMAC). The proof of the security of authentication and authenticated key establishment protocols in the Bellare and Rogaway's model leads from an alleged successful attack on a protocol to the collapse of pseudo-randomness, i.e., the output of a pseudo-random function can be distinguished from that of a truly random function by a polynomial-time distinguisher, in the proof of the adversary; in other words, the existence of pseudo-random functions is denied. This implies that the result of the reduction should be either false or a major breakthrough in the foundations for modern cryptography. As the former is more likely the case, the reduction derives a contradiction as desired[8].

6.1.1 Formal modeling of protocols

The protocols considered in Bellare and Rogaway's model are two party ones, formally specified by an efficiently computable function, a polynomial-time function Π on the following input values:

1^k: the security parameter k, $k \in \mathbb{N}$ where $\mathbb{N}$ is a set of natural numbers.

i: the identity of the sender ranging over I, $i \in I \subseteq \{0,1\}^k$ where I is a set of principals who can participate in the protocol and share a secret long-term key, and $\{0,1\}^k$ denotes the set of finite binary strings of length at most k.

j: the identity of the (intended) communication partner of the sender, the receiver ranging over I. Elements of I will sometimes be denoted as A and B (or Alice and Bob), rather than i and j. Note that the adversary is not a partner in the Bellare-Rogaway model and $A = B$ (or $i = j$) is quite possible.

K: the long-term symmetric key shared between the sender i and the receiver j.

conv: the conversation so far, $conv \in \{0,1\}^*$ where $\{0,1\}^*$ denotes the set of finite binary strings. *conv* grows with the protocol run; new string is concatenated to it.

r: the random coin inputs of the sender like a nonce generated by the sender.

The function of $\Pi(1^k, i, j, K, conv, r)$ implies that K, r is of size k, and $i, j, conv$ is of size polynomial in k. The value of $\Pi(1^k, i, j, K, conv, r) = (m, \delta, \alpha)$ specifies:

m: the next message to send out, $m \in \{0,1\}^* \cup \{$"no message output"$\}$.

α: the privte output, $\alpha \in \{0,1\}^* \cup \{$"no private output"$\}$.

δ: the decision for the sender, $\delta \in \{$Accept, Reject, Undetermined$\}$. An acceptance decision usually does not occur until the end of the protocol, although a rejection decision may occur at any time. For mutual authentication protocol, only acceptance or rejection decision is concerned with. For key exchange protocols, the private output, such as an agreed session key, is concerned with. Once a decision other than "undetermined" is reached, the private output will no longer change.

Oracle $\Pi^s_{i,j}$ models partner i attempting to authenticate partner j in a "session" s for $i, j \in I$ and $s \in \mathbb{N}$.

6.1.2 Formal modeling of communications

In Bellare-Rogaway model, all communications among interacting parties are assumed to be under the adversary I's control. Particularly, the adversary can read the messages produced by the parties, provide messages of his own to them, modify messages before they reach their destination, and delay mes-

sages or replay them. Most importantly, the adversary can start up entirely new "instances" of any of the parties, modeling the ability of communicating agents to simultaneously engage in many sessions at once. Formally the adversary I is a probabilistic machine equipped with an infinite collection of oracles $\Pi^s_{i,j}$, I can conduct as many sessions as I pleases among the honest partners, and I can persuade a partner to start a protocol run as if it is run with another honest partner. Each honest party will be modeled by an infinite collection of oracles which the adversary may run. These oracles only interact with the adversary, they never directly interact with one another[7].

Query

A query is of the form (i, j, s, x) which means that the adversary I is sending message x to i, and the adversary I claims that it is from j in session s. A query from I will be answered by an oracle $\Pi^s_{i,j}$. In response to an oracle call, I learns not only the outgoing message but also whether or not the oracle has accepted, but I couldn't learn the oracle's private output. In a particular execution of a protocol, the adversary's i-th query to an oracle is said to occur at time $\tau = \tau_i \in \mathbb{R}$ where $\mathbb{R}$ is a set of reals.

The Benign Adversary

An adversary is called benign if it is deterministic and it restricts its action to choosing a pair of oracles $\Pi^s_{i,j}$ and $\Pi^s_{j,i}$ and then faithfully conveying each flow from one oracle to the other, with $\Pi^s_{i,j}$ beginning first. In other words, the first query I makes is $(i, j, \tau_1, \texttt{""})$, generating response m_1; the second query I makes is (j, i, τ_2, m_1), generating response m'_1, and so forth. While the choice of i, j, τ_1, τ_2 is up to the adversary, this choice is the same in all executions with security parameter k. Therefore, a benign adversary behaves just like a wire between i and j. If the adversary wished to have the targeted partners to output the acceptance decision, the adversary's behavior should be restricted to that of a benign adversary.

Time

Conforming notions of time include "abstract time", where $\tau_i = i$, and "Turing machine time", where $\tau_i =$ the i-th step in I's computation, when parties are realized by interacting Turing machines. Another conforming notion of time (but a harder one to formalize) is "real time", where τ_i is the exact time when the i-th query is made, when parties are realized by interacting computers. For the i-th query and the j-th query, if $i < j$, we demand that $\tau_i < \tau_j$.

6.1.3 Formal modeling of entity authentication

A central notion in formalizing entity authentication goals is that of a matching conversation.

Bellare and Rogaway defined authenticity security as the matching conversation by an experiment involving the running of the adversary I with security parameter k. When I terminates, each oracle $\Pi_{i,j}^{s}$ has had a certain conversation $conv_{i,j}^{s}$ with I, and it has reached a certain decision $\delta \in \{\text{Accept, Reject, Undetermined}\}$.

Conversation

A conversation of oracle $\Pi_{i,j}^{s}$ is a sequence of timely ordered messages that a partner sent out (respectively, received), and as consequent responses, received (respectively, sent). Let $\tau_1 < \tau_2 < \cdots \tau_n$ be a time sequence, for any oracle $\Pi_{i,j}^{s}$, the conversation (for this execution) can be denoted by the following sequence:

$$conv = (\tau_1, m_1, m_1'), (\tau_2, m_2, m_2'), \cdots, (\tau_n, m_n, m_n').$$

This sequence encodes that at time τ_1, the participant was asked m_1 and responded with m_1', and then, at some later time $\tau_2 > \tau_1$, oracle $\Pi_{i,j}^{s}$ was asked m_2, and answered m_2'; and so forth, until, finally, at time τ_n it was asked m_n, and answered m_n'. Adversary I terminates without asking oracle $\Pi_{i,j}^{s}$ any more questions.

Suppose oracle $\Pi_{i,j}^{s}$ has conversation prefixed by (τ_1, m_1, m_1'). Then if $m_1 =$"", we call oracle $\Pi_{i,j}^{s}$ an *initiator oracle*; if m_1 is any other string, we call $\Pi_{i,j}^{s}$ a *responder oracle*. If $m_n =$"no message output", $\Pi_{i,j}^{s}$ ends the conversation. At the end of a protocol run, each participant makes a decision about the authentication of the intended partner: accept, reject, or undetermined.

Matching conversation

Give a protocol Π between partners i and j. Run Π in the presence of a benign adversary I and consider two oracles $\Pi_{i,j}^{s}$ and $\Pi_{j,i}^{s'}$ that engage in conversations *conv* and *conv*$'$ in sessions s and s', respectively.

1) We say that *conv*$'$ is a matching conversation to *conv* if there exist time sequences $\tau_0 < \tau_1 < \tau_2 < \cdots < \tau_n$ and $m_1, m_1', m_2, m_2', m_3, \cdots, m_{t-1}', m_t, m_t'$ so that *conv* is prefixed by

$$conv = (\tau_0, \text{""}, m_1), (\tau_2, m_1', m_2), (\tau_4, m_2', m_3), \cdots, \left(\tau_{2t-2}, m_{t-1}', m_t\right)$$

and *conv*$'$ is prefixed by

$$conv' = (\tau_1, m_1, m_1'), (\tau_3, m_2, m_2'), (\tau_5, m_3, m_3'), \cdots, \left(\tau_{2t-3}, m_{t-1}, m_{t-1}'\right).$$

2) We say that *conv* is a matching conversation to *conv*$'$ if there exists time sequence $\tau_0 < \tau_1 < \tau_2 < \cdots \tau_n$ and $m_1, m_1', m_2, m_2', m_3, \cdots, m_{t-1}', m_t, m_t'$ so that *conv*$'$ is prefixed by

$$conv' = (\tau_1, m_1, m_1'), (\tau_3, m_2, m_2'), (\tau_5, m_3, m_3'), \cdots, (\tau_{2t-1}, m_t, *)$$

and *conv* is prefixed by

$$conv = (\tau_0, \text{""}, m_1), (\tau_2, m_1', m_2), (\tau_4, m_2', m_3), \cdots, (\tau_{2t-2}, m_{t-1}', m_t).$$

Explanation. Case (1) defines when the conversation of a responder oracle matches the conversation of an initiator oracle. Case (2) defines when the conversation of an initiator oracle matches the conversation of a responder oracle.

Consider an execution in which $\Pi_{i,j}^{s}$ is an initiator oracle and $\Pi_{j,i}^{s'}$ is a responder oracle. If every message that $\Pi_{i,j}^{s}$ sends out, except possibly the last, is subsequently delivered to $\Pi_{j,i}^{s'}$, with the response to this message being returned to $\Pi_{i,j}^{s}$ as its own next message, then we say that the conversation of $\Pi_{j,i}^{s'}$ matches that of $\Pi_{i,j}^{s}$. Similarly, if every message that $\Pi_{j,i}^{s'}$ receives was previously generated by $\Pi_{i,j}^{s}$, and each message that $\Pi_{j,i}^{s'}$ sends out is subsequently delivered to $\Pi_{i,j}^{s}$, with the response that this message generates being returned to $\Pi_{j,i}^{s'}$ as its own next message, then it is said that the conversation of $\Pi_{i,j}^{s}$ matches the one of $\Pi_{j,i}^{s'}$.

It is said that oracle $\Pi_{j,i}^{s}$ has a matching conversation with oracle $\Pi_{i,j}^{s}$ if $\Pi_{j,i}^{s}$ has conversation $conv'$, $\Pi_{i,j}^{s}$ has conversation *conv*, meanwhile $conv'$ matches *conv*.

Entity Authentication

Any mutual authentication protocol must have at least 3 round message exchanges. The definition of a mutual authentication is similar to [1] as follows:

Definition 6.1 (Secure mutual authentication) Give a protocol Π between partners A and B. We say that Π is a secure mutual authentication protocol in the presence of any polynomial time adversary I if

1) (Matching conversations $\Rightarrow$ acceptance.) The oracles $\Pi_{A,B}^{s}$ and $\Pi_{B,A}^{s'}$ have matching conversations, then both oracles accept.

2) (Acceptance $\Rightarrow$ matching conversations.) The adversary I cannot win with a non-negligible probability in k. Here the adversary wins if the oracles $\Pi_{A,B}^{s}$ and $\Pi_{B,A}^{s'}$ both reach the "accept" decision while they do not have matching conversations in oracles.

Explanation. The first condition says that if one party's messages are faithfully relayed to one another, then the party accepts the authentication of one another. The second condition says that if oracles $\Pi_{A,B}^{s}$ and $\Pi_{B,A}^{s'}$ have reached the "accept" decision, then $\Pi_{A,B}^{s}$ or $\Pi_{B,A}^{s'}$ must have a matching conversation in both oracles.

Note that an oracle's matching partner is unique based on the Definition 6.1. More formally, let $\textbf{Multiple-Match}^{E}(k)$ be the event that some $\Pi_{A,B}^{s}$ accepts in the presence of any polynomial time adversary I, and there are at least two distinct oracles $\Pi_{B,A}^{s'}$ and $\Pi_{B,A}^{s''}$ which have had matching conversations with $\Pi_{A,B}^{s}$.

Proposition 6.1 Suppose the protocol Π between partners A and B is a secure mutual authentication protocol. Let I be any polynomial time adversary. Then the probability of Multiple-Match$^E(k)$ is negligible.

The probability of Multiple-Match$^E(k)$ is at most $l^2 * 2^{-k}$ where l is the (polynomial) number of oracle calls of the adversary I and k is the security parameter.

Bellare and Rogaway demonstrate their formal proof technique by providing a simple mutual entity authentication protocol named MAP1 and conducting its proof of security. They also consider the correctness for authenticated key establishment protocols. For more details, please refer to references [1, 7].

6.2 Security definition of SK-security

Bellare and Rogaway's idea of provable security under a computational model originates from the seminal work of Goldwasser and Micali[3]. Goldwasser and Micali propose a well-known public-key probabilistic encryption scheme named the Goldwasser-Micali cryptosystem, GM cryptosystem, which possesses the property of semantic security assuming the intractability of the quadratic residuosity problem. The semantic security is a stronger security notion: Whatever is efficiently computable about the plaintext given the ciphertext, is also efficiently computable without the ciphertext[8].

The semantic security, which is also known as IND-CPA security or polynomial-time indistinguishability, means that a ciphertext does not leak any useful information about the plaintext to any attacker whose computational power is polynomially bounded. They observed that in many applications, messages may contain certain apriori information which may be useful for an attack. Goldwasser and Micali point out that public-key cryptosystems which are based on direct applications of one-way trapdoor functions are in general very weak for hiding such messages[3, 8]. The semantic security notion tries to meet the need for a general fix of this much bigger problem.

There, a security property (one of several confidentiality qualities) is argued under a given attacking scenario (one of several attacking games each of which models, with sufficient generality and precision, one of some typical behavior of a real-world attacker against public-key encryption schemes). A proof of security for public-key encryption schemes with respect to an alleged attack involves demonstrating an efficient transformation (called a polynomial-time reduction) leading from the alleged attack to a major breakthrough to a well-believed hard problem in computational complexity. It is the wide belief on the unlikelihood of the major breakthrough that should refute the existence of the alleged attack, that is, a proof is given by contradiction[8].

The Goldwasser-Micali scheme[3] can be described in a general setting by using the notion of a trapdoor predicate. Briefly, a trapdoor predicate is

a Boolean function $B : \{0,1\}^* \rightarrow \{0,1\}$ so that, given a bit v, it is easy to choose an x at random satisfying $B(x) = v$. Moreover, given a bitstring x, computing $B(x)$ correctly with probability significantly greater than $\frac{1}{2}$ is difficult; however, if certain trapdoor information is known, then it is easy to compute $B(x)$. Suppose an entity A's public-key is a trapdoor predicate B, any other entity encrypts a message bit m_i by randomly selecting an x_i so that $B(x_i) = m_i$, and then sends x_i to A. Since A knows the trapdoor information, A can compute $B(x_i)$ to recover m_i, but an adversary can do no better than guess the value of m_i. Goldwasser and Micali proved that if trapdoor predicates exist, then this probabilistic encryption scheme is polynomially secure[3, 9].

In the Bellare-Rogaway model, they also consider the correctness for authenticated session key establishment (session key transport) protocols (for the two-party case[1], also for the three-party case which uses a trusted third party as an authentication server[7]). The security notion of key establishment originates from the Goldwasser-Micali probabilistic encryption[3]. For key establishment protocols, "Malice wins" means a successful guess of the new session key. Since the new session key is randomly chosen by a pseudo-random function, and the transported key is encrypted under the shared long-term key, successful guessing of the session key is similarly hard as making distinction between a pseudo-random function and a truly random function.

SK-Security notation is an important notion in the authentication protocol field. Canetti and Krawczyk put forward the CK model[10], including the SK-Security notation, for the analysis of key establishment protocols that results from the combination of two previous works in this area: [1] by Bellare and Rogaway and [2] by Bellare, Canetti and Krawczyk.

6.2.1 Protocol and adversary models in CK model

Canetti and Krawczyk extend and refine the formalism in the approach of [10], where a general framework for studying the security of session-based multi-party protocols over insecure channels is introduced. Let's review the CK model.

6.2.1.1 Protocol notations

$P_1, P_2, ..., P_n$: a set of parties (probabilistic polynomial-time machines) interconnected by point-to-point links over which messages can be exchanged.

Protocols: collections of interactive procedures, run concurrently by parties $P_1, P_2, ..., P_n$, which specify a particular processing of incoming messages and the generation of outgoing messages.

Message-driven protocols: Protocols are initially triggered at a party by an external "call" and later by the arrival of messages. Upon each of these events, and according to the protocol specification, the protocol processes

information and may generate and transmit a message and/or wait for the next message to arrive.

Session: Each copy of a protocol run at a party is a session. Technically, a session is an interactive subroutine executed inside a party. Each session is identified by the party that runs it, the parties with whom the session communicates and by a session-identifier (session ID). Several copies of protocols may be simultaneously run by each party. Each invocation of a protocol (or session) at a given party creates a local state for that session during execution, and produces local outputs by that party. When a session ends its run we call it complete and assume that its local state is erased.

Session key: Key-Establishment (KE) protocols are message-driven protocols where the communication takes place between pairs of parties and which return, upon completion, a secret key called a session key. It is required that the calling protocol makes sure that the session ID's of no two KE sessions in which the party participates are identical. Furthermore, we leave it to the calling protocol to make sure that two parties that wish to exchange a key will activate matching sessions.

Upon activation, the partners P_i and P_j of two matching sessions exchange messages, and eventually generate local outputs that include the name of the partners of the session, the session identifier (session ID), and the value of the computed session key. A key establishment event is recorded only when the exchange is completed. Note that a session can be completed at one partner but not necessarily at the other.

6.2.1.2 Adversary models

Adversarial setting determines the capabilities and possible actions of the attacker. In CK model, the adversary model is given as generic as possible (as opposed to, say, merely representing a list of possible attacks). The CK model follows the general adversary formalism of [2] but specializes and extends the adversarial model here for the case of KE protocols.

1. The unauthenticated-links adversarial model (UM)

Basic attacker capabilities Consider a probabilistic polynomial-time (PPT) attacker that has full control of the communication links. The formalism represents this ability of the attacker by letting the attacker be the one in charge of passing messages from one party to another. The attacker also controls the scheduling of all protocol events including the initiation of protocols and message delivery.

Obtaining secret information All the secret information stored at a party is potentially vulnerable to break-ins or to other forms of leakage. The attacker may obtain secret information stored in the party's memories via explicit attacks. However, when defining security of a protocol, it is important to guarantee that the leakage of some form of secret information has the least possible effect on the security of other secrets. In order to be able to differentiate between various vulnerabilities and to be able to guarantee as

much security as possible in the event of information exposures, three attacks categories are classified depending on the type of information accessed by the adversary:

1) *Session-state reveal.* The attacker provides the name of a party and a session identifier of a yet incomplete session at that party and receives the internal state of that session. What information is included in the local states of a session is to be specified by each KE protocol. Typically, the revealed information will include all the local state of the session and its subroutines, except for the local state of the subroutines that directly access the long-term secret information.

2) *Session-key query.* The attacker provides a party's name and a session identifier of a completed session at that party and receives the value of the key generated by the named session. This attack provides the formal modeling for leakage of information on specific session keys that may result from events such as break-ins, cryptanalysis, and careless disposal of keys.

3) *Party corruption.* The attacker can decide at any point to corrupt a party, in which case the attacker learns *all* the internal memory of that party including long-term secrets (such as private keys or master shared keys used across different sessions) and session-specific information contained in the party's memory (such as internal state of incomplete sessions and session-keys corresponding to completed sessions).

If a session is subject to any of the above three attacks then the session is called locally exposed. If a session or its matching session is locally exposed then we call the session exposed.

Session expiration. Session expiration means that a session key (and any related session state) is erased from that party's memory. The value of an expired session key cannot be found via any of the above session-state reveal, session-key query and party corruption attacks if these attacks are performed after the session expired.

2. The authenticated-links adversarial model (AM)

Authenticated-links adversarial model The attacker is restricted to only deliver messages truly generated by the parties without any change or addition to them. This is the fundamental difference between UM adversarial model and AM adversarial model.

Emulation A notion introduced in order to capture the equivalence of functionality between protocols in different adversarial models, particularly between the UM and AM adversarial models.

Authenticator A special algorithm acts as an automatic "compiler" that translates protocols in the AM adversarial model into equivalent (or "as secure as") protocols in the UM adversarial model.

6.2.2 SK-security in CK model

The security of a key-exchange protocol was defined by Canetti and Krawczyk, it's also called session-key security in CK model.

Session-key security (or SK-security), focus on ensuring the security of an individual session-key as long as the session key value is not obtained by the attacker via an explicit key exposure. To capture the idea that the attacker "does not learn anything about the value of the key" from interacting with the key-establishment protocol and attacking other sessions and parties, the CK model formalizes SK-security via the infeasibility to distinguish between the real value of the key and an independent random value as that in the semantic-security approach[1]. The formulation of SK-security is very careful about tuning the definition to offer enough strength as required for the use of key-establishment protocols to realize secure channels, as well as being realistic enough to avoid over-kill requirements which would prevent researchers from proving the security of very useful protocols[10].

First, define an "experiment" where the attacker I chooses a session which is "tested" about information it learned on the session-key; specifically, ask the attacker to differentiate the real value of the chosen session key from a random value.

For the sake of this experiment we extend the usual capabilities of the adversary I in the UM by allowing it to perform a test-session query. That is, in addition to the regular actions of I against a key-exchange protocol Π, we let I choose, at any time during its run, a test-session among the sessions that have been completed, and are unexpired and unexposed at the time. Let k be the value of the corresponding session key. We toss a coin b, $b \xleftarrow{R} \{0,1\}$. If $b = 0$, we provide the attacker I with the value k. Otherwise we provide the attacker I with a value r randomly chosen from the probability distribution of keys generated by the protocol Π. The attacker I is now allowed to continue with the regular actions of a UM adversary but is not allowed to expose this test-session (namely, it is not allowed session-state reveals, session-key queries, or partner's corruption on this test-session or its matching session). At the end of its run, the attacker I outputs a bit b' (as its guess for b). We will refer to an attacker that is allowed test-session queries as a KE-adversary.

Definition 6.2 (SK-Security[10]) A KE protocol Π is called SK-secure if the following properties hold for any KE-adversary I in the UM.

1) Protocol Π satisfies the property that if two uncorrupted parties complete matching sessions then they both output the same key.

2) The probability that the adversary I guesses correctly the bit b (i.e., outputs $b' = b$) is no more than $1/2$ plus a negligible fraction in the security parameter.

The first condition is a "consistency" requirement for sessions completed by two uncorrupted parties. The second condition is the "core property"

for SK-security. Definition 6.2 is very powerful and can be shown to ensure many specific properties that are required from a good key-exchange protocol. However, a key establishment protocol where all key-sessions "hang" and never return also satisfies the definition. For simplicity, Canetti et al choose to leave the analysis of the termination properties of protocols out of the scope of the definition of security. This is also the case in this book.

6.3 Authentication based on trusted freshness

In this section, we will introduce the security definitions based on trusted freshness. Some notations and security definitions in [1, 3] are adopted in the trusted freshness security definition. Recall that the central ingredient in the trusted freshness security analysis approach is the freshness principles introduced in chapter 4 of this book. Let's review and introduce some notations related to trusted freshness security analysis.

— Principals, probabilistic polynomial time machines, which are interconnected by point-to-point links over which messages can be exchanged.
— Trusted Third Party (TTP), a principal that provides a centralized authentication service in an open system.
— Freshness identifier (or TVP), a unique freshness component generated for a particular protocol run, it can be a nonce, a timestamp, a session key or a shared part of a session key.
— Protocol, a communication procedure which is run between or among co-operative principals.
— Message-driven protocols, protocols are initially triggered at a party by an external "call" and later by the arrival of messages.
— Challenge-Response protocol, in a challenge-response mechanism, one participant can verify the lively correspondence of the intended opposite partner by inputting a freshness identifier (challenge) to a composition of a protocol message and the composition involves a cryptographic operation (response) performed by the intended opposite partner.
— Session, a copy of a protocol run at a party, several copies of any protocol may be simultaneously run by each party.

6.3.1 Trusted freshness

In the context of communication protocols, that a freshness identifier is fresh means that the identifier has not been used previously, and originated within an acceptably recent time. Formally, *fresh* typically means *recent*, and it is in the sense of having originated subsequent to the beginning of the current protocol instance. Note that such freshness alone does not rule out interleaving

attacks using parallel sessions[8, 9].

Definition 6.3 (Freshness) Given a protocol Π between partners A and B. A component of the protocol Π is fresh if the component is guaranteed to be new from the viewpoint of one party A or B.

Classification of freshness component

Three freshness component categories are classified depending on the type of the component:

1) Timestamp Recording the time of creation, transmission, receipt or existence of information.

The party originating a message obtains a timestamp from its local (host) clock, and binds it to a message. Upon receiving a time-stamped message, the second party obtains the current time from its own (host) clock, and subtracts the timestamp received. The timestamp difference should be within the acceptance window, and each party should maintain a "loosely synchronized" clock which must be appropriate to accommodate the acceptance window used.

The time clock must be secure to prevent adversarial resetting of a clock backwards so as to restore the validity of old messages, or setting a clock forward to prepare a message for some future point in time.

2) Nonce A value originated subsequent to the beginning of the current protocol instance, and it is used no more than once for the same purpose.

In a challenge-response protocol, the term nonce is most often used to refer to a "random" number which is sampled from a sufficiently large space, but the required randomness properties vary. A key parameter or the shared parts of a key may be viewed as a nonce in some cases. If random numbers are chosen by A and B, respectively, then the random numbers together with a signature may provide a guarantee of freshness and entity authentication.

3) Sequence number A value provided by a never-repeated sequential counter, and it serves as a unique number identifying a message and is typically used to detect message replay. A message is accepted only if the sequence number therein has not been used previously (or not used previously within a specified time period). Sequence number changes on every new protocol instance or new message depending on different purposes.

Each party should maintain the sequence number pairwise of the originator and the receiver, and be sufficient to determine previously used and/or still valid sequence numbers. Distinct sequences are customarily necessary for messages from A to B and from B to A.

Sequence numbers may provide uniqueness, but not (real-time) timeliness, and thus are more appropriate to detect message replay than entity authentication. Sequence numbers may also be used to detect the deletion of entire messages; they thus allow data integrity to be checked over an ongoing sequence of messages, in addition to individual messages.

Note that a sequence number is not natively fresh even for the sequence

number generator.

The cost for a random number or a sequence number to provide a freshness guarantee is an additional message more than that for the timestamp in the one-pass technique.

Definition 6.4 (Trusted freshness) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the freshness identifier γ is fresh, or in other words, a trusted freshness, if for a participant A,

1) the freshness identifier γ originates in the participant A itself.

2) the freshness identifier γ is a timestamp and the timestamp difference between the initiator and the receiver is within the acceptance window.

3) A has corroborative evidence that γ is fresh. Here the corroborative evidence may be a signature, a MAC or other one-way transformation including the freshness identifier.

The first sufficient condition of Definition 6.4 is based on the randomization of A's nonce, which has been sampled at random from a sufficiently large space and so no one can predicate the value before sampling; the second sufficient condition of Definition 6.4 is based on a "loosely synchronized" clock, and the timestamp difference is within the acceptance window, hence the "recent" property could be checked by the opponent party; The third sufficient condition of Definition 6.4 is based on the security property of the cryptographic algorithms, and it is widely used and more useful in challenge-response protocols. Note that the freshness of a freshness identifier could be given via mathematical proofs.

Theorem 6.1 (Generation Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. If a freshness identifier γ is a nonce or a timestamp, and it is generated by the participant A itself, then A believes that γ is a trusted freshness.

Proof A freshness identifier γ originating in the participant A could be a nonce, a timestamp or a sequence number. Let's consider the freshness of γ in these three cases:

(1) The freshness identifier γ is a nonce.

Let's recall the supposition: the term nonce is most often used to refer to a "random" number which is sampled from a sufficiently large space.

The "random" numbers are in fact pseudo-random numbers, they are generated by a pseudorandom number generator and they have a distribution totally determined (i.e., in a deterministic fashion) by a seed. Yet, a good pseudo-random number generator yields pseudorandom numbers which are polynomially indistinguishable from truly random numbers.

Recall that the adversary I is a probabilistic polynomial-time machine which has full control of the communication links. Hence, the adversary I

couldn't distinguish the distribution of the random variables output from a pseudorandom number generator from the uniform distribution of strings (truly random numbers) which are of the same length as those of the pseudorandom variables.

Hence, if a freshness identifier γ is generated by the participant A itself, then A believes that γ is recent and it is a "random" number. Since γ is a "random" number, it couldn't be guessed by a probabilistic polynomial-time attacker I, and we can guarantee that the freshness identifier γ is used no more than once for the same purpose. That is, the freshness identifier γ is a trusted freshness.

(2) The freshness identifier γ is a timestamp.

Since the freshness identifier γ is generated by the participant A itself, then A believes that γ is recent. Recall the supposition that the timestamp difference between the initiator and the receiver is within the acceptance window, so there is no time gap for the freshness identifier γ to be used for other purpose. That is, the freshness identifier γ is used no more than once for the same purpose, hence the timestamp γ is a trusted freshness.

(3) The freshness identifier γ is a sequence number.

The sequence number is a value provided by a never repeated sequential counter, and it is typically used to detect message replay. Since the freshness identifier γ is generated by the participant A itself, A believes that γ is recent. However, the sequence number γ may be guessed even by a probabilistic polynomial-time attacker I, so I could obtain the intending response messages from sending request messages to the victim oracle, and the attacker I may replay the achieved messages including γ for other purpose. Hence, we do not regard a sequence number as a trusted freshness.

Theorem 6.2 (Timestamp Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. If a freshness identifier γ is a timestamp, and it is received by A from the opponent B, then A believes that γ is a trusted freshness.

Proof Since the freshness identifier γ is a timestamp, and the timestamp difference between the initiator and the receiver is within the acceptance window, so A believes that γ is recent and there is no time gap for the freshness identifier γ to be used for other purpose. That is, the freshness identifier γ is used no more than once for the same purpose, hence γ is a trusted freshness.

Recall the term, maximal term, signed term and similar term notions in Chapter 4, the follows is a example.

Example 6.1 Here is another illustration of terms. Suppose there exists a message $B \rightarrow A : \{B, A, N_A, \{N_A, N_B, A, B\}_K\}$, the principal B believes that N_B is a trusted freshness identifier, and the principal A believes that N_A is a trusted freshness identifier too.

Upon sending this message, B has the terms:

1	$+\{...N_B...\}$	Definition 4.10 (1)
2	$+\{...N_B...\}_K$	Term 1 and Definition 4.10 (3)
3	$+\{...N_A, N_B...\}_K$	Term 2 and Definition 4.10 (2),(3)
4	$+\{...N_B, A...\}_K$	Term 2 and Definition 4.10 (2),(3)
5	$+\{...N_B, B...\}_K$	Term 2 and Definition 4.10 (2),(3)
6	$+\{...N_A, N_B...\}_K$	Term 2 and Definition 4.10 (2),(3)
7	$+\{...N_A, N_B, A...\}_K$	Term 6 and Definition 4.10 (2),(3)
8	$+\{...N_A, N_B, B...\}_K$	Term 6 and Definition 4.10 (2),(3)
9	$+\{...N_A, N_B, A, B...\}_K$	Term 8 and Definition 4.10 (2),(3)

Upon sending this message, A has the terms:

1	$-\{...N_A...\}$	Definition 4.10 (1)
2	$-\{...N_A...\}_K$	Term 1 and Definition 4.10 (3)
3	$-\{...N_A, N_B...\}_K$	Term 2 and Definition 4.10 (2),(3)
4	$-\{...N_A, A...\}_K$	Term 2 and Definition 4.10 (2),(3)
5	$-\{...N_A, B...\}_K$	Term 2 and Definition 4.10 (2),(3)
6	$-\{...N_A, N_B...\}_K$	Term 2 and Definition 4.10 (2),(3)
7	$-\{...N_A, N_B, A...\}_K$	Term 6 and Definition 4.10 (2),(3)
8	$-\{...N_A, N_B, B...\}_K$	Term 6 and Definition 4.10 (2),(3)

$\{...N_A, N_B, A, B...\}_K$ is the maximal term of the message $\{N_A, N_B, A, B\}_K$ sent from B to A.

To detect parallel attack, the maximal term of each message in the same protocol should not be the same.

6.3.2 Liveness of principal

The notation of the principal's liveness originates from [8], which means the presence of the intended partner. We try to make the liveness property more specific by giving the principal's liveness definition and a liveness rule.

Definition 6.5 (Liveness of a principal) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The liveness of B means that B is the intended partner of the communication from the point of view of A, and B is responsive to the communications in the current protocol run.

Theorem 6.3 (Liveness Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the liveness of B has been

authenticated by A if A has corroborative evidence that B has participated in this protocol run.

Suppose γ is a trusted freshness from the point of view of A. A may have the liveness of B from receiving the following "loose" one-way transformation,

1) A signature including the trusted freshness γ, which is signed by B.

2) An encryption including the trusted freshness γ, which is encrypted with MAC by B using the shared long-term key between A and B.

3) Other one-way transformation of a message including the trusted freshness identifier γ, and this cryptographic operation can provide evidence of B's generation of this one-way transformation.

Proof Let's consider the liveness of B from the point of view of A in the following three scenarios:

(1) Signature including the trusted freshness identifier γ by B.

Suppose B is not alive and is not responsive to the communications in the current protocol run. Since the signature (it is believed to be signed by B's private key) includes the trusted freshness identifier γ, then this signature could not be a replay of an old recorded message. So, it must be the adversary I who has recently constructed the signature including the trusted freshness identifier γ. That is to say, the adversary I has the ability to construct the signature without knowing the corresponding private key. This has translated an advantage for an alleged attack on the protocol to a similar (up to polynomial difference) advantage for inverting the signature used in the scheme. This contradicts the wide belief that there exists no efficient algorithm for inverting a signature.

Hence, if A has received the signature including the trusted freshness identifier γ from B, then A is assured that B is alive and responsive to the communications in the current protocol run.

(2) Encryption including the trusted freshness identifier γ under the shared long-term key between $A$and B.

Recall that the maximal term of each message in the same protocol should not be the same, so this one-way transformation could not be a replay of encryption by A itself. Other proof procedures are similar to the case (1).

(3) One-way transformation including the trusted freshness identifier γ, and this cryptographic operation can provide evidence of B's generation of this one-way transformation.

The proof procedures are similar to the case (1).

The second condition of Theorem 6.3 also includes the case – keyed hash where the encryption algorithm is a hash function but not a traditional block cipher, and the key is the shared long-term key between A and B.

Definition 6.6 (Liveness of a principal with origin) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The liveness of B means that B is the intended partner of the communication from the point of view of A, and B is specially responsive to the communication with A in

the current protocol run.

Theorem 6.4 (Origin Liveness Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the Origin liveness of B has been authenticated by A if A has specially corroborative evidence that B has participated in this protocol run with this origin participant A.

Suppose γ is a trusted freshness from the point of view of A. A may have the origin liveness of B from receiving the following "loose" one-way transformation.

1) A signature including the trusted freshness γ and the origin identity A, which is signed by B.

2) An encryption including the trusted freshness γ and the origin identity A, which is encrypted with MAC by B using the shared long-term key between A and B.

3) Other one-way transformation of a message including the trusted freshness identifier γ and the origin identity A, and this cryptographic operation can provide evidence of B's generation of this one-way transformation.

6.3.3 Confidentiality of freshness identifier

Definition 6.7 (Confidentiality of freshness identifier) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the freshness identifier is confidential if the adversary I could not know γ from the protocol run.

Theorem 6.5 (Confidentiality Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. Suppose all freshness identifiers are confidential at the beginning of a protocol run. The confidential property may change in the following two scenarios:

1) The freshness identifier is transmitted in plain text.
2) The freshness identifier is transmitted in an encryption whose decryption key is known by the adversary I.

Proof Omitted for its obviousness.

6.3.4 Freshness of freshness identifier

Definition 6.8 (Freshness of a Freshness Identifier) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the

freshness identifier γ is fresh if γ is new for this protocol run.

Theorem 6.6 (Freshness Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the freshness of γ has been authenticated by A if A has corroborative evidence that γ is new for this protocol run. The corroborative evidence includes the following three scenarios:

1) The freshness identifier γ originates in the participant A itself for this protocol run.

2) The freshness identifier γ is a timestamp and the timestamp difference between the initiator and the receiver is within the acceptance window.

3) A has corroborative evidence that γ is fresh. Here the corroborative evidence may be a signature, a MAC or other one-way transformation to assure the freshness of the freshness identifier.

Proof Refer to proofs in Theorem 6.4 and Theorem 6.1.

6.3.5 Association of freshness identifier

Definition 6.9 (Association of Freshness Identifier) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the freshness identifier γ is associated with A (and/or B) if γ is generated for the protocol run related with A (and/or B).

Theorem 6.7 (Association Rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the freshness identifier γ is associated with A (and/or B) from the point of view of A if A has corroborative evidence that γ is generated for this protocol run related with A (and/or B). The corroborative evidence includes the following scenarios:

1) A sends or receives a message which is encrypted under the shared long-term key between two parties, then the trusted freshness identifier γ in this message is related to these two parties.

2) A sends or receives a message including the identity of B, which is encrypted under the shared long-term key between A and the trusted third party S, then the trusted freshness identifier γ in this message is related to B.

3) A receives an encryption under the public-key of A, then the trusted freshness identifier γ in this encryption is related to A.

4) A sends an encryption including the identity of A using the public-key of the opponent B, then the trusted freshness identifier γ in this encryption is related to A.

5) A receives a signature of B, then the trusted freshness identifier γ in this signature is related to B.

6) A receives a signature of B including the identity of A, then the trusted freshness identifier γ in this signature is related to A.

7) Other one-way transformation including the trusted freshness identifier γ, and this cryptographic operation can provide evidence of B's (and/or A's) association with γ, then the trusted freshness identifier γ in this one-way transformation is related to B (and/or A).

Proof We show that 1) of the theorem is satisfied. Suppose the two parties in case 1) are A and B. If A receives a message including the trusted freshness identifier which is encrypted under the shared long-term key between A and B, then A knows that it must be a message originating from B and it could not be a replay one. Since the received message could only be decrypted by both A and B, and the adversary could do nothing but replay this one, hence A believes that the trusted freshness identifier γ in this message is related to these two parties A and B.

The proofs for 2) to 7) of the theorem are similar, hence omitted for concision.

The encryption algorithm in 1) and 2) of Theorem 6.7 may be block cipher or keyed hash; In case 3) of Theorem 6.7, if A receives an encryption including the identity of B under the public-key of A, then the freshness identifier γ in this encryption needn't be related to B, since this message could even be generated by the adversary I.

6.3.6 Security analysis based on trusted freshness

We mainly discuss Challenge-Response authentication protocols.

6.3.6.1 Notion

— Unilateral entity authentication: the identity of one protocol participant is authenticated.
— Mutual entity authentication: the identities of both protocol participants are authenticated to each other.
— Unilateral authenticated key transport: the identity of one protocol participant is authenticated, and the opposite unauthenticated party believes that the session key generated by the authenticated participant or a TTP can provide a secure channel over an insecure network.
— Mutual authenticated key transport: the identities of both protocol participants are authenticated to each other, and both protocol participants believe that the new session key generated by one of the participants or a TTP can provide a secure channel over an insecure network.

— Mutual authenticated key exchange (or key agreement): the identities of both protocol participants are authenticated to each other, and both protocol participants believe that the new session key which is the output of a function of all protocol participants' random input can provide a secure channel over an insecure network.

6.3.6.2 Hypothesis

Suppose we have a probabilistic polynomial-time (PPT) attacker I that has full control of the communication links as described in Dolev-Yao threat model[11]. Besides this, we suppose that the Dolev-Yao attacker I in this book is allowed to perform a kind of cryptanalysis training course, and I can also launch the adaptive chosen ciphertext attacks (CCA2) without limitations.

Suppose we have cryptographic primitives with security against indistinguishable adaptive chosen ciphertext attack (IND-CCA2). That is, in IND-CCA2 security strength, the failures in cryptographic protocols are not in any way related to the strength or weakness of the particular cryptographic primitives used, but related to the protocol logic flaws, which permits the attacker to break the security goals of cryptographic protocols without necessarily breaking the particular cryptographic primitives used. And we also suppose that a legitimate party is either totally corrupted or totally secure.

Suppose that each participant has his own private key and all other parties' public-keys (respectively, the shared long-term key between co-operative principals or the trusted third party) in public-key case (respectively, in shared key case), which are deployed safely before the cryptographic protocol run via authenticated channel or even traditional communication means. Furthermore, private keys and shared keys are commonly assumed to be too long to guess in a computationally feasible way.

Suppose that all freshness identifiers are confidential at the beginning of the protocol run.

In general, an authentication protocol is considered flawed if a principal concludes a normal run of the protocol with its intended communication partners while the intended partner would have a different conclusion.

6.3.6.3 Notation

— ρ, arbitrary principal, ranges over the participants of the protocol run.
— P_i or P_j, a principal indexed by subscript in a protocol run.
— S, trusted third party.
— t, arbitrary time, a moment, not a period of time.
— $t_0, t_1, t_2, ..., t_\$$, various time points. t_0 means time before the start of a protocol run, t_i means time at message i ($i = 1, 2, \dots$) exchange, and $t_\$$ means time at the termination of a protocol run respectively.
— N or N', arbitrary freshness identifier, it can be a nonce, a timestamp, a session key or the shared part of a session key.
— N_{P_i}, a freshness identifier invented by subscript principal P_i.

— k, a cryptographic key; k^{-1}, the inverse of k. In shared key case, k and k^{-1} are equal.
— K, a long-term key, it may be a secret key in shared key schemes, or a public-key and a private key in public-key schemes.
— $K_{P_iP_j}$, the long-term key shared by principal P_i and P_j in shared key case.
— K_{P_i} and $K_{P_i}^{-1}$, the public-key and private key subscripted by the principal identity respectively in public-key case.
— φ, ψ or Γ, a fact which has the value of *True* or *False* respectively.
— $\neg$, it is the same as negation used in logic.
— $Key(P_i, k)$, the principal P_i has the knowledge of key k.
— $Belief(P_i, \varphi)$, the principal P_i asserts that the fact φ is *True*. For example, $Belief(P_i, Key(P_j, k))$ means that the principal P_i asserts that the principal P_j knows the key k.
— $Existing(P_j)$, the intended partner P_j is in lively correspondence in this protocol run; $Belief(P_i, Existing(P_j))$, the principal P_i asserts that the intended partner P_j is in lively correspondence in this protocol run.
— $Originexisting(P_j)$, the intended partner P_j is specially in lively correspondence in this protocol run with the origin participant P_i; $Belief(P_i, Originexisting(P_j))$, the principal P_i asserts that the intended partner P_j is specially in lively correspondence in this protocol run with the origin participant P_i.
— $Secret(N)$, the freshness identifier N is confidential in this protocol run; $Belief(P_i, Secret(N))$, the principal P_i asserts that the freshness identifier N is confidential in this protocol run.
— $Fresh(N)$, the freshness identifier N is fresh in this protocol run; $Belief(P_i, Fresh(N))$, the principal P_i asserts that the freshness identifier N is fresh in this protocol run.
— $Associate(N, P_1, P_2, P_3, ...)$, the freshness identifier N is associated with a participant P_1, (or with both P_1 and P_2, or with P_1, P_2 and P_3...) in this protocol run; $Belief(P_i, Associate(N, P_1, P_2, P_3, ...))$, the principal P_i asserts that the freshness identifier N is associated with a participant P_1, (or with both P_1 and P_2, or with P_1, P_2 and P_3...) in this protocol run. For example, $Belief(P_i, Associate(N_A, A, B))$ means the principal P_i asserts that the freshness identifier N_A is associated with the principals A and B.

6.3.7 Definition of security

The security definition under computational model provides a high confidence of the security of a cryptosystem[1, 3, 8, 13]. Recall the notations "conversation", "matching conversations", and we make tiny changes: only one-way transformation that includes a trusted freshness identifier is considered as an

efficient message of a conversation in our security analysis of cryptographic protocols based on trusted freshness. That is to say, we only concern with the fresh messages but omit the message parts that do not contribute to our protocol security properties to be proved in the trusted freshness approach.

Recall that at the end of a protocol run, each participant makes a decision about the authentication of the intended partner: "accept", "reject", or "undetermined"[8]. Given a protocol Π between the principal A and the principal B, if a principal like A with a conversation *conv* believes that B always has a conversation *conv*′ which matches *conv* whenever they are allowed to complete a protocol run, then this authentication protocol is secure from the point of view of A. Here the attacker wins if the principal A has reached "accept" decision while B does not have a matching conversation in B.

Semantic security is widely accepted in the cryptographic area, and we follow the probabilistic indistinguishability definitional approach in [3] to define confidentiality security. In this book, that the attacker has broken the scheme means that without breaking any cryptographic algorithm and knowing the corresponding key, the attacker can still learn something about the established new session key under the run of a cryptographic protocol. Here we define "learn" as distinguishing the value of a key generated by the cryptographic protocol from an independent randomly chosen key.

Based on the security definition of authenticity, we have presented the Unilateral entity Authentication Secure definition (UA-Secure) (Definition 4.3) and Mutual entity Authentication Secure definition (MA-Secure) (Definition 4.4); based on the security definition of authenticity and confidentiality, we have presented the Unilateral authenticated Key Secure (UK-Secure) (Definition 4.5) and Mutual authenticated Key Secure (MK-Secure[10]) (Definition 4.6).

In Chapter 4, we have also presented the security properties to clarify whether a cryptographic protocol is adequate for the security goals or not. Here, four formal security specifications will be given based on Theorem 4.1, Theorem 4.2, Theorem 4.3 and Theorem 4.4.

Theorem 6.8 (UA-Secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The authentication protocol Π is UA-Secure if and only if the following property holds for one of the participants, say P_i: P_i has Existing(P_i, P_j), that is, P_i believes that the intended opposite participant P_j is in lively correspondence with P_i in this protocol run.

Proof We show that the authentication protocol Π is UA-Secure if and only if P_i has *Existing*(P_j).

(1) Sufficiency proof. We show that if P_i has *Existing*(P_j), then the protocol Π meets Definition 4.3, that is UA-Secure. Suppose P_i wants to authenticate the identity of the opponent partner P_j. This try could be made by P_i or be a reply to the message sent by the opponent partner P_j. Recall

that only one-way transformation that includes a trusted freshness identifier is considered as an efficient message of a conversation in our trusted freshness. If P_i has $Existing(P_j)$, that is P_i believes the liveness of the intended opposite principal P_j, according to the liveness rule (Theorem 6.3), P_i must have invented a challenge N_{P_i} for this particular protocol run, and received a one-way transformation that includes N_{P_i}, where the one-way transformation can only be accomplished by the principal P_j. Recall that we have a cryptographic algorithm under IND-CCA2, if P_i sees the conversation $conv_{P_i} = (\tau_0, \cdots, N_{P_i}), (\tau_2, \{N_{P_i}\}_{K_{P_iP_j}}, \cdots)$ in shared key case or $conv_{P_i} = (\tau_0, \cdots, N_{P_i}), (\tau_2, \{N_{P_i}\}_{K_{P_j}^{-1}}, \cdots)$ in public-key case, then P_i sees that the uniformly random string $\{N_{P_i}\}_{K_{P_iP_j}}$ or $\{N_{P_i}\}_{K_{P_j}^{-1}}$ is computed using N_{P_i} invented by P_i itself; it can therefore conclude that the probability for this bit string not having been computed by its intended partner (in other words, having been computed by the attacker) is at the level of 2^{-k}. Consequently, P_i can conclude that its intended partner has a conversation which is prefixed by $conv_{P_j} = (\tau_1, N_{P_i}, \{N_{P_i}\}_{K_{P_iP_j}})$ or $conv_{P_j} = (\tau_1, N_{P_i}, \{N_{P_i}\}_{K_{P_j}^{-1}})$. This essentially shows that there exists a conversation $conv_{P_j}$ matching $conv_{P_i}$, and the conversation $conv_{P_j}$ has been computed by the intended partner P_j in an overwhelming probability (in the security parameter $K_{P_iP_j}$ or $K_{P_j}^{-1}$). According to the security definition of authentication, hence P_i believes that P_j is in lively correspondence with P_i in the matching session.

(2) Necessary proof. We show that if the protocol Π is UA-Secure, then P_i has $Existing(P_j)$. Suppose the UA-Secure protocol Π doesn't hold the listed security property $Existing(P_j)$, that is, P_i does not believe the liveness of the intended opposite principal P_j, then, according to the liveness rules (Theorem 6.3), P_i has either sent a compromised or an old challenge N_{P_i} to the intended opposite partner P_j (namely, the attacker can replay a recorded stale message to P_i by impersonating P_j), or P_i does not require a response to P_i's challenge (namely, the attacker can launch an attack directly). So the protocol Π cannot be UA-secure, hence there exists a contradiction with the initial assumption that the protocol Π is UA-Secure. Typical examples include Otway-Rees protocol[13], revised Woo-Lam protocol[14].

Therefore, we can conclude that the listed UA-secure security property is not only sufficient but also necessary for the protocol Π to be UA-secure.

Theorem 6.9 (MA-Secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The authentication protocol Π is called MA-Secure if the following properties hold for the participants P_i and P_j:

1) P_i has $Existing(P_j)$, that is, P_i believes that the intended opposite participant P_j is in lively correspondence with P_i in this protocol run;

2) P_j has $Existing(P_i)$, that is, P_j believes that the intended opposite participant P_i is in lively correspondence with P_j in this protocol run;

Proof Similar to Theorem 6.8, omitted.

Theorem 6.10 (Origin UA-secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The authentication protocol Π is Origin UA-Secure if and only if the following property holds for one of the participants, say P_i: P_i has Originexisting(P_j), that is, P_i believes that the intended opposite participant P_j is specially in lively correspondence with this origin participant P_i in this protocol run.

Theorem 6.11 (Origin MA-secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The authentication protocol Π is called Origin MA-Secure if the following properties hold for the participants P_i and P_j:

1) P_i has Originexisting(P_j), that is, P_i believes that the intended opposite participant P_j is specially in lively correspondence with this origin participant P_i in this protocol run;

2) P_j has Originexisting(P_i), that is, P_j believes that the intended opposite participant P_i is specially in lively correspondence with this origin participant P_j in this protocol run;

Unilateral entity authentication secure (UA-secure) and Mutual entity authentication secure (MA-secure) are the most common cases in real world applications for identity authentication and access control services, which may suffer the replay attacks and at last may provide false identity authentication and access control. In deed, Origin Unilateral entity authentication secure (Origin UA-secure) and Origin Mutual entity authentication secure (Origin MA-secure) could meet these real world application requirements.

Theorem 6.12 (UK-secure) Given an authentication protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. k is the new session or the shared part of the new session key for this protocol run. The authentication protocol Π is called UK-Secure if the following properties hold for one of the participants, say P_i:

1) P_i has $Existing(P_j)$. That is, P_i believes that the intended opposite participant P_j is in lively correspondence with P_i in this protocol.

2) P_i has $Secret(k)$, $Fresh(k)$ and $Associate(k, P_i, P_j)$. That is, P_i believes that the adversary I could not know the new session key k and k is new for this protocol run between P_i and P_j.

Proof Similar to Theorem 6.13, omitted.

Theorem 6.13 (MK-secure) Given an authentication protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. k is the new session key or the shared part of the new session key for this protocol run. The au-

thentication protocol Π is called MK-Secure if the following properties hold for the participants P_i and P_j:

1) P_i has $Existing(P_j)$. That is, P_i believes that the intended opposite participant P_j is in lively correspondence with P_i in this protocol run.

2) P_j has $Existing(P_i)$. That is, P_j believes that the intended opposite participant P_i is in lively correspondence with P_j in this protocol run.

3) P_i has $Secret(k)$, $Fresh(k)$ and $Associate(k, P_i, P_j)$. That is, P_i believes that the adversary I could not know the new session key k and k is new for this protocol run between P_i and P_j.

4) P_j has $Secret(k)$, $Fresh(k)$ and $Associate(k, P_i, P_j)$. That is, P_j believes that the adversary I could not know the new session key k and k is new for this protocol run between P_i and P_j.

Proof Sufficiency proof. We show that 1) of the Definition 4.6 is satisfied by protocol Π. Since the principal P_i believes the liveness of the intended opposite principal P_j and only one-way transformation that includes a trusted freshness identifier is considered as an efficient message of a conversation in our approach, according to liveness rule (Theorem 6.3), P_i must have generated a challenge N_{P_i} for this particular protocol run, and received a one-way transformation that includes a trusted freshness identifier N_{P_i}, where the one-way transformation can only be accomplished by the principal P_j. Recall that we have an ideal cryptographic algorithm with security against IND-CCA2, if P_i sees the conversation $conv_{P_i} = (\tau_0, \cdots, N_{P_i}), (\tau_2, \{N_{P_i}\}_{K_{P_iP_j}}, \cdots)$ in shared key case or $conv_{P_j} = (\tau_1, N_{P_i}, \{N_{P_i}\}_{K_{P_j}^{-1}})$ in public-key case, then P_i sees that the uniformly random string $\{N_{P_i}\}_{K_{P_iP_j}}$ or $\{N_{P_i}\}_{K_{P_j}^{-1}}$ is computed using N_{P_i} invented by P_i itself; it can therefore conclude that the probability for this bit string not having been computed by its intended partner (in other words, having been computed by the attacker) is at the level of 2^{-k}. Consequently, P_i can conclude that its intended partner has a conversation which is prefixed by $conv_{P_j} = (\tau_1, N_{P_i}, \{N_{P_i}\}_{K_{P_iP_j}})$ or $conv_{P_j} = (\tau_1, N_{P_i}, \{N_{P_i}\}_{K_{P_j}^{-1}})$. This essentially shows that there exists a conversation $conv_{P_j}$ matching $conv_{P_i}$ and the conversation $conv_{P_j}$ has been computed by the intended partner in an overwhelming probability (in the security parameter $K_{P_iP_j}$ or $K_{P_j}^{-1}$). According to the security definition of authentication, P_i believes that P_j is in lively correspondence with P_i in this session. Similarly, P_j believes that P_i is in lively correspondence with P_j in this session.

Since principal P_i believes the freshness of the new session key k, according to the Freshness Rule (Theorem 6.6), k must be a new generated session key for this run. Since principal P_i believes the association of the new session key k with the principals P_i and P_j, according to the Association Rule (Theorem 6.7), k must be a session key for a particular protocol run between P_i and P_j. Up to now, k must be a new generated session key for a particular protocol run between P_i and P_j, hence k is the same key for both P_i and P_j, and it is different from other generated keys in any other sessions. That is to say, if

two uncorrupted parties complete matching sessions then they both output the same key.

We show that 2) of the Definition 4.6 is also satisfied by protocol Π. Recall that the attacker I is a PPT machine who has full control of the communication links and I is allowed to perform a kind of cryptanalysis training course. We can specify that l is an upper bound of the session number for training invoked by I in any interactions. Let k be the value of the corresponding session key selected randomly in this protocol Π. Let P_i play the game with the attacker I as in Definition 4.6. Let Bad be the events that the information of k may leak during the cryptanalysis training courses. Let I_{wins} denote the event that I makes a correct guess of the challenge bit b. It is clear that in absence of the event Bad, due to the uniform randomness of the selected session key k, the challenge bit b is independent of the challenge ciphertext b'. Thus we have

$$\text{Prob}[I_{wins}|\overline{Bad}] = \frac{1}{2}.$$

Since

$$\text{Prob}[I_{wins}|\overline{Bad}] = \frac{\text{Prob}[I_{wins} \cap \overline{Bad}]}{\text{Prob}[\overline{Bad}]},$$

we have

$$\text{Prob}[I_{wins} \cap \overline{Bad}] = \frac{1}{2}\text{Prob}[\overline{Bad}] = \frac{1}{2}(1 - \text{Prob}[Bad]).$$

While

$$\text{Prob}[I_{wins}] = \text{Prob}[I_{wins} \cap Bad] + \text{Prob}[I_{wins} \cap \overline{Bad}],$$

therefore

$$\begin{aligned}\text{Prob}[I_{wins}] &\leqslant \text{Prob}[Bad] + \text{Prob}[I_{wins} \cap \overline{Bad}] \\ &= \text{Prob}[Bad] + \frac{1}{2}(1 - \text{Prob}[Bad]) = \frac{1}{2}(1 + \text{Prob}[Bad]).\end{aligned}$$

Since we have an ideal cryptosystem, even the attacker I has invoked l times cryptanalysis training course, the probability that the information of k may be leaked to I by the underlying cryptosystem (say $Bad1$) is negligible in the security parameter, that is $\text{Prob}[Bad1] \leqslant l * Adv$ where Adv is a negligible fraction. Since principal P_i believes the liveness of the intended partner P_j, the probability that the information of k might be leaked by P_j to I (say $Bad2$) is 0. Since principal P_i believes the association of the new session key k with the principals P_i and P_j, the probability that the attacker I could persuade P_j to believe a key between I and P_j (or I and P_i) to be the key k between P_i and P_j (say $Bad3$) is 0. Then we have

$$\text{Prob}[Bad] = \text{Prob}[Bad1] + Prob[Bad2] + Prob[Bad3] \leqslant l * Adv,$$

therefore

$$\text{Prob}[I_{wins}] \leqslant \frac{1}{2}(1 + \text{Prob}[Bad]) \leqslant \frac{1}{2}(1 + l * Adv) = \frac{1}{2} + \frac{l * Adv}{2}.$$

Since attacker I is a PPT attacker, then $(l * Adv)/2$ is negligible. Therefore, the probability that I guesses correctly the bit b is no more than $1/2$ plus a negligible fraction in the security parameter.

Necessary proof. Suppose that a MK-secure protocol Π doesn't hold the listed security properties.

If a participant like P_i (or P_j) does not believe the liveness of the intended opposite principal P_j (or P_i), then, according to Theorem 6.3, P_i (or P_j) has sent either a compromised or an old challenge to the intended opposite partner P_j (or P_i) (That is to say, the attacker can replay a recorded stale message to P_i (or P_j) by impersonating P_j (or P_i), or does not require a response to P_i's (or P_j's) challenge (That is to say, the attacker can launch an attack directly). This contradicts the initial assumption. The typical examples are Otway-Rees protocol[13], revised Woo-Lam protocol[14]. Hence the protocol Π cannot be MK-secure.

If a participant like P_i (or P_j) believes that the confidentiality of the new session key k is open, then, according to Lemma 4.2, the attacker wins with the probability of 1 when playing the game in Definition 4.6. Hence the protocol Π cannot be MK-secure.

If a participant like P_i (or P_j) does not believe the freshness of the session key k, then, according to Lemma 4.3, the attacker can replay a recorded message including a compromised key k' as response to P_i (or P_j). A typical example is the Needham-Schroeder shared key protocol[15]. Hence the protocol Π cannot be MK-secure.

If a participant like P_i (or P_j) does not believe the association of the session key k with the co-operative participants, then, according to Lemma 4.4, the attacker may cheat a legitimate participant by confusing a key between the attacker and another to be the key between two legitimate participants. A typical example is the Needham-Schroeder public-key protocol[8]. Hence the protocol Π cannot be MK-secure.

Therefore, we can conclude that the listed MK-secure security properties are not only substantial but also necessary for the protocol Π to be MK-secure.

6.3.8 Non-repudiation based on trusted freshness

Non-repudiation means a principal could not deny his sending or receiving a message. These non-repudiation services (protection against false denials) relate to the transfer of a message from an originator to a recipient. Non-repudiation is an important security service for many applications and it is

a necessary security requirement in electronic commerce. Non-repudiation mechanisms are specified for non-repudiation of origin (denial of being the originator of a message), non-repudiation of delivery (denial of having received a message). Commonly used *fairness* security in electronic commerce protocols can also be derived from non-repudiation.

When disputes arise due to a principal denying that certain actions (having sent or received a message) were taken, a trusted third party can be called upon to make an arbitration: give the proof of message origin, or the proof of message delivery. For example, a principal may authorize a purchase of a car and later he may deny such an authorization granted. A procedure involving a third party is needed to resolve the dispute. What evidence would be submitted to the third party, and what precise process the third party is to follow to render judgement on disputes should be specified in detail for a non-repudiation service.

Digital signatures can provide the non-repudiation service because a signature of a message is verifiable universally. The non-repudiation service provided by a digital signature means a proof of knowledge that a signer owns exclusively a private key (knowledge) which has enabled him (her) to issue the signature. The non-repudiation aspect of digital signatures is a primary advantage of public-key cryptography.

Symmetric techniques (including encipherment and keyed one-way functions) typically cannot provide non-repudiation service effectively, since the data integrity techniques based on a shared secret key (e.g., MACs) typically involve mutual trust and do not address true (single-source) data origin authentication, that is, either party sharing the secret key can equally originate a message using the shared key.

If the resolution of subsequent disputes is a potential requirement, then either an on-line trusted third party is in a notary role, or asymmetric techniques should be used.

Definition 6.10 (Non-repudiation) A crypto protocol Π can provide non-repudia-tion service if any attacker I cannot win with a non-negligible probability in Dolev-Yao threat model. Here the attacker wins if a principal A has reached the conclusion of message origin of B, or message delivery of B while B has not sent or received the message.

Rule 6.1 (Non-repudiation rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the protocol Π can provide non-repudiation service if A has corroborative evidence that the dispute message m originates from B, or the dispute message m has been received by B. The corroborative evidence includes these following scenarios:

1) The dispute message m is a signature of its originator B.

2) The dispute message m is guaranteed known by the dispute opponent B via the message composition involving a cryptographic operation (response) that could only be performed by the intended opposite partner B.

Note: Suppose the trusted third party S is secure, then the corroborative evidence could be an encryption sent by B to the trusted third party encrypted under the shared key between B and S.

Definition 6.11 (Real time non-repudiation) A cryptographic protocol Π can provide realtime non-repudiation service if any attacker I cannot win with a non-negligible probability for a particular protocol run in Dolev-Yao threat model. Here the attacker wins if a principal A has reached the conclusion of message origin of B, or message delivery of B for this particular protocol run while B has not sent or received the message.

Rule 6.2 (Real time non-repudiation rule) Given a protocol Π between partners A and B in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say that the protocol Π can provide non-repudiation service if A has corroborative evidence that the dispute message m originates from B for this particular protocol run, or the dispute message m has been received by B for this particular protocol run. The corroborative evidence includes these following scenarios:

1) The dispute message m including a timestamp is a signature of its originator B.

2) The dispute message m is guaranteed known by the dispute opponent B via B's cryptographic operation (response), which could only be performed by B, on the message composition including a timestamp.

3) The dispute message m is guaranteed known by the dispute opponent B via the message composition involving a cryptographic operation (response) that could only be performed by B, and the cryptographic operation includes a timestamp.

Here are some notations supplemented in trusted frehsness approach:

— $Originate(\rho, m, P_j)$, every principal ρ or at least a dispute judger could assert that the dispute message m originates from P_j.
— $Respond(\rho, m, P_j)$, every principal ρ or at least a dispute judger could assert that the dispute message m has been received by P_j.
— $RealTimeOriginate(\rho, m, P_j)$, every principal ρ or at least a dispute judger could assert that the dispute message m originates from P_j for a particular protocol run.
— $RealTimeRespond(\rho, m, P_j)$, every principal ρ or at least a dispute judger could assert that the dispute message m has been received by P_j for a particular protocol run.

Theorem 6.14 (Non-repudiation secure) Given an authentication protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. m is a dispute message between partners P_i and P_j. We say that the protocol Π can provide non-repudiation service if either of the following properties holds for one of the participants, say P_i:

1) P_i has Originate(P_i, m, P_j). That is, P_i has corroborative evidence that

the dispute message m is generated by P_j.

2) P_i has Respond(P_i, m, P_j). That is, P_i has corroborative evidence that the dispute message m has been received by P_j.

Proof We show that 1) of the Theorem 6.14 is satisfied. Suppose that an adversary I can win with a non-negligible probability to make the principal P_i believe that a forged message m originates from the opponent partner P_j. Recall that P_i has $Originate(P_i, m, P_j)$. That is, P_i has corroborative evidence that the dispute message m is generated by P_j. Suppose the corroborative evidence is that the dispute message m is a signature of its originator P_j. Hence, the forged message m is a successfully constructed signature of its originator P_j. The adversary's ability for signature forgery can be fully translated to one for inverting the hard function (i.e., the underlying one-way trapdoor function). This contradicts the assumed well-known intractability problem (i.e., factoring of RSA moduli).

Therefore, we can conclude that $Originate(P_i, m, P_j)$ is substantial for the protocol Π to provide non-repudiation service.

We show that 2) of the Theorem 6.14 is satisfied. Suppose that an adversary I can win with a non-negligible probability to make the principal P_i believe that a forged message m has been received by the opponent partner P_j. Recall that P_i has $Respond(P_i, m, P_j)$. That is, P_i has corroborative evidence that the dispute message m has been received by P_j. The corroborative evidence is that the dispute message m is known by the dispute opponent P_j via a cryptographic operation that could only be performed by the intended opposite partner (i.e., a decryption using P_j's private key of receiving an encryption including m). Hence, if the adversary wins, then the adversary could decrypt a public-key encryption without knowing the corresponding private key in public-key case. The adversary's ability for decryption without corresponding key can also be fully translated to one for inverting the hard function. This also contradicts the assumed well-known intractability problem.

Therefore, we can conclude that $Respond(P_i, m, P_j)$ is also substantial for the protocol Π to provide non-repudiation service.

Theorem 6.15 (Realtime non-repudiation secure) Given an authentication protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. m is the dispute message for this protocol run. We say that the protocol Π can provide non-repudiation service if either of the following properties holds for one of the participants, say P_i:

1) P_i has $RealTimeOriginate(P_i, m, P_j)$. That is, P_i has corroborative evidence that the dispute message m is generated by P_j.

2) P_i has $RealTimeRespond(P_i, m, P_j)$. That is, P_i has corroborative evidence that the dispute message m has been received by P_j.

Proof We show that 1) of the Theorem 6.15 is satisfied. The proof that

the Non-repudiation property, the dispute message m, originates from P_j, in this theorem is similar to that proof in Theorem 6.14, hence omitted.

We show that the real time property in this theorem is also satisfied. Recall that P_i has $RealTimeOriginate(P_i, m, P_j)$. That is, P_i (respectively, P_j) has corroborative evidence that the dispute message m has been sent by P_j in a "loosely synchronized" clock time (the timestamp records the time of sending). In deed, this timestamp is obtained by the party P_j from its local (host) clock, and is bound to the dispute message m. Since each party has maintained a "loosely synchronized" clock which is appropriate to accommodate the acceptance window used, if the timestamp difference is within the acceptance window, then P_i believes that this dispute message m is a real time message.

Therefore, we can conclude that $RealTimeOriginate(P_i, m, P_j)$ is sufficient for the protocol Π to provide real time non-repudiation service.

From the proof of the real time property in 1) of Theorem 6.15 and also the proof of delivery property in 2) of Theorem 6.14, we can obtain the proof of 2) real time non-repudiation in Theorem 6.15 similarly.

References

[1] Bellare M, Rogaway P (1993) Entity Authentication and Key Distribution. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 773, pp 232 – 249, Springer

[2] Bellare M, Canetti R, Krawczyk H (1998) A Modular Approach to the Design and Analysis of Authentication and Key-exchange Protocols. In: Proceedings of the 30th STOC, Dallas, 23 – 26 May 1998

[3] Goldwasser S, Micali S (1984) Probabilistic Encryption. Journal of Computer and System Sciences 28(2): 270 – 299

[4] Goldwasser S, Micali S, Rivest R (1988) A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. SIAM Journal of Computing 17(2): 281 – 308

[5] Blum M, Micali S (1984) How to Generate Cryptographically Strong Sequences of Pseudorandom Bits. SIAM Journal on Computing 13(4): 850 – 864

[6] Yao AC (1982) Theory and Applications of Trapdoor Functions. In: Proceedings of the IEEE 23rd Annual Symposium on the Foundations of Computer Science, Chicago, 3 – 5 Nov 1982

[7] Bellare M, Rogaway P (1995) Provably Secure Session Key Distribution – the Three Party Case. In: Proceedings of the 27th ACM Symposium on the Theory of Computing, Las Vegas, 29 May – 1 June 1995

[8] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[9] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[10] Canetti R, Krawczy H (2001) Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In: EUROCRYPT'01 Proceedings of the

International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045, pp 453 – 474, Springer

[11] Dolev D, Yao AC (1983) On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2): 198 – 208

[12] Goldreich O (2003) Foundations of Cryptography. Cambridge University Press, New York

[13] Otway D, Rees O (1987) Efficient and Timely Mutual Authentication. Operating Systems Review 21(1): 8 – 10

[14] Abadi M, Needham R (1996) Prudent Engineering Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering 22(1): 6 – 15

[15] Lowe G (1995) An Attack on the Needham-Schroeder Public Key Authentication Protocol. Information Processing Letters 56(3): 131 – 133

7 Formalism of Protocol Security Analysis

Abstract Formal methods are natural extensions to informal ones that have been used to analyze cryptographic protocols. First, some famous formalisms such as BAN logic, model checking and strand space are briefly introduced; then a belief multiset formalism is put forward based on the trusted freshness notion in Chapters 4, 5 and also 6, and the formalism is simple and precise for automation of security analysis.

Formal methods are natural extensions to informal ones, and they have proven to be useful in finding flaws and redundancies in cryptographic protocols. A formal method usually supports a symbolic system or a description language for modeling and specifying a system's behavior so that the behavior can be captured and reasoned about by applying logical and mathematical methods in a rigorous manner. Sometimes, a formal method is an expert system which captures human experience or even tries to model human ingenuity. A common characteristic of formal methods is that they take a systematic, sometime an exhaustive approach to a problem. Therefore, formal methods are particularly suitable for the analysis of complex systems[1].

In the area of formal analysis of cryptographic protocols, two distinct approaches are identified. One can be referred to as formal reasoning about holding of some desirable, or secure properties (Correctness proof of a protocol); the other can be referred to as systematic search for some undesirable, or dangerous properties (Finding flaws of a protocol). In the first approach, a protocol to be analyzed must be very carefully chosen or designed so that it could be proved to be correct and secure. In the latter approach, a protocol is often considered error-prone although it has been carefully chosen or designed. So in this latter approach, analysis is to, in terms of systematic, or exhaustive approach, search for errors[1–3].

The correctness analysis approach tries to establish that the protocol is indeed correct with respect to a set of desirable properties which have also been carefully formalized. Because of the carefully chosen protocols to be analyzed, a formal proof is often specially tailored to the target protocol and may hence need to have much human ingenuity involvement, although the proof methodology can be more general. The correctness analysis approach further branches to two schools: a computational school (i.e., the Goldwasser-Micali

provable security approach) and a symbolic manipulation school (i.e., the Burrows-Abadi-Needham logic). In the latter school, security properties are expressed as a set of abstract symbols which can be manipulated, sometimes by a formal logic system, sometimes by a mechanical tool called a theorem prover, toward a YES/NO result[1]. Logic-based methods attempt to reason that a protocol is correct by evolving a set of beliefs held by each party, and to eventually derive a belief that the protocol goals have been obtained. The most popular symbolic method is the Burrows-Abadi-Needham (BAN) logic[4].

The fault finding analysis approach tries to formalize a protocol, to formalize the behavior of an adversary and to find path from initial state to the insecure state. For example, the insecure state means the message ends up in Malice's set of knowledge for the case of secrecy of a message, or the insecure state means a wrong identity ends up in the set of accepted identities of an honest principal for the case of entity authentication[1]. The most popular fault finding method is Model Checking[5].

Note that the security proofs provided are within the specified formal system, and cannot be interpreted as absolute proofs of security. Some of these security analysis techniques are also unwieldy, or applicable only to a subset of protocols or classes of attack. Many require (manually) converting a concrete protocol into a formal specification, a critical process which itself may be subject to subtle flaws[2].

We have introduced the idea whether a cryptographic protocol is secure based on trusted freshness in Chapters 4, 5 and also 6. This idea is useful in that it has pointed out a freshness principle and the specifications to guarantee the security objects of a protocol. This trusted freshness idea could be implemented via a correctness proof approach and also via a fault finding approach.

In this chapter, we will first review some famous formalisms including BAN logic, Model checking method, and then give a logic-based belief multiset formalism derived from the trusted freshness which is simple and precise for automation of security analysis. We have also noticed that other formalisms for protocol analysis could be presented based on the freshness principle, for example, a formalism using the notion of Strand[6].

7.1 BAN logic

The BAN logic[4] is a "Logic of Authentication" proposed by Burrows, Abadi and Needham. The BAN logic provides a set of logical formulae to model the basic actions of protocol participants and the meanings of the basic protocol components. It focuses on the beliefs of trustworthy parties involved in authentication protocols and on the evolution of these beliefs as a consequence of communication.

Since BAN logic operates at an abstract level, it does not consider errors introduced by concrete implementation of a protocol, or errors such as deadlocks, or even inappropriate use of cryptosystems.

In this section we will briefly review the BAN logic, and we refer the interested reader to [4] for details of the logic and its semantics.

7.1.1 Basic notation

BAN logic distinguishes several sorts of objects: principals, encryption keys, and formulas (also called statements). Messages are identified with statements in the logic. Typically, the symbols A, B, and S denote specific principals; K_{AB}, K_{AS} and K_{BS} denote specific shared keys between A and B, A and S, and B and S respectively; K_A, K_B and K_S denote specific public-keys, and K_A^{-1}, K_B^{-1} and K_S^{-1} denote the corresponding secret keys for A, B and S respectively; N_A, N_B and N_C denote specific statements. The symbols P, Q, and R range over principals; X and Y range over statements, and K ranges over encryption keys. The only propositional connective is conjunction, denoted by a comma. In addition to conjunction, BAN logic uses the following constructs:

— P *believes* X: P believes X, or P would be entitled to believe X.
— P *sees* X: P sees X, that is, someone has sent a message containing X to P, who can read and repeat X.
— P *said* X: P once said X, that is, the principal P at some time sent a message including the statement X.
— P *controls* X: P has jurisdiction over X, that is, the principal P is an authority on X and should be trusted on this matter.
— *Fresh*(X): The formula X is fresh, that is, X has not been sent in a message at any time before the current run of the protocol.
— $P \overset{K}{\longleftrightarrow} Q$: P and Q may use the shared key K to communicate. The key K is good, in that it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.
— $\overset{K}{\longmapsto} P$: P has K as a public-key, that is, the matching secret key (denoted K^{-1}) will never be discovered by any principal except P or a principal trusted by P.
— $P \overset{X}{\rightleftharpoons} Q$: The formula X is a secret known only to P and Q, and possibly to principals trusted by them. Only P and Q may use X to prove their identities to one another.
— $\{X\}_K$: This represents the formula X encrypted under the key K. Formally, $\{X\}_K$ is a convenient abbreviation for an expression of the form $\{X\}_K$ from P.
— $\langle X \rangle_Y$: This represents X combined with the formula Y; it intends that Y be a secret and that its presence prove the identity of whoever utters

$\langle X \rangle_Y$.

7.1.2 Logical postulate

The *present* epoch begins at the start of the particular run of the protocol under consideration. All messages sent before this time are considered to be in the *past*, and the authentication protocol should be careful to prevent any such messages from being accepted as *recent*. All beliefs held in the *present* are stable for the entirety of the protocol run.

1) Message-meaning rules:

— For shared keys case:

$$\frac{P \textit{ believes } Q \overset{K}{\longleftrightarrow} P,\ P \textit{ sees } \{X\}_K}{P \textit{ believes } Q \textit{ said } X}.$$

That is, if P believes that the key K is shared with Q and sees X encrypted under K, then P believes that Q once said X.

— For public-keys case:

$$\frac{P \textit{ believes} \overset{K}{\longmapsto} Q,\ P \textit{ sees } \{X\}_{K^{-1}}}{P \textit{ believes } Q \textit{ said } X}.$$

That is, if P believes that K and K^{-1} are the key pair of Q and sees X signed under K^{-1}, then P believes that Q once said X.

— For shared secrets case:

$$\frac{P \textit{ believes } Q \overset{Y}{\rightleftharpoons} P,\ P \textit{ sees } \langle X \rangle_Y}{P \textit{ believes } Q \textit{ said } X}.$$

That is, if P believes that the secret Y is shared with Q and sees $\langle X \rangle_Y$, then P believes that Q once said X.

2) Nonce-verification rule:

$$\frac{P \textit{ believes } \textit{fresh}(X), P \textit{ believes } Q \textit{ said } X}{P \textit{ believes } Q \textit{ believes } X}.$$

That is, if P believes that X could have been uttered only recently (in the present) and that Q once said X (either in the past or in the present), then P believes that Q believes X.

3) Jurisdiction rule:

$$\frac{P \textit{ believes } Q \textit{ controls } X,\ P \textit{ believes } Q \textit{ believes } X}{P \textit{ believes } X}.$$

The jurisdiction rule states that if P believes that Q has jurisdiction over X then P trusts Q on the truth of X.

4) Seeing rules:

$$\frac{P \textit{ sees } (X,Y)}{P \textit{ sees } X}, \frac{P \textit{ sees } \langle X \rangle_Y}{P \textit{ sees } X}, \frac{P \textit{ believes } Q \overset{K}{\longleftrightarrow} P,\ P \textit{ sees } \{X\}_K}{P \textit{ sees } X},$$
$$\frac{P \textit{ believes } \overset{K}{\longmapsto} P,\ P \textit{ sees } \{X\}_K}{P \textit{ sees } X}, \frac{P \textit{ believes } \overset{K}{\longmapsto} Q,\ P \textit{ sees } \{X\}_{K^{-1}}}{P \textit{ sees } X}.$$

Those rules state that if a principal sees a formula, then he also sees its components, provided he knows the necessary keys.

5) Fresh rule:

$$\frac{P \textit{ believes } \textit{fresh}(X)}{P \textit{ believes } \textit{fresh}(X,Y)}.$$

This rule states that if one part of a formula is fresh, then the entire formula must also be fresh.

7.1.3 Steps for security analysis based on BAN logic

1) Idealize protocols: This is a process to transform protocol messages into logical formulae in BAN logic. That is, the protocol messages are "idealized". There are not any operational rules for the process of the protocol idealization but to omit cleartext communication simply because it can be forged, and so its contribution to an authentication protocol is mostly one of providing hints as to what might be placed in encrypted messages. The idealized messages are of the form $\{X_1\}_{K_1}$, ..., $\{X_n\}_{K_n}$.

2) Formulate the premises and goals: The BAN logic starts by formulating a set of premises which are protocol assumptions including the belief premises and the state premises. Typically, the assumptions state what keys are initially shared between the principals, which principals have generated fresh nonce, and which principals are trusted in certain ways. The goals of authentication protocols are also indicated at the start of a communication. For example, an authentication is complete between A and B if there is a new session key K so that A believes $A \overset{K}{\longleftrightarrow} B$, B believes $A \overset{K}{\longleftrightarrow} B$.

3) Establish the beliefs: The BAN logic axioms are applied to the premises and the logical formulae with an aim of establishing a desired property such as a good-key statement. Particularly, the formula before the first message represents the beliefs of the principals at the start of the protocol. Step by step, we can follow the evolution from the initial beliefs to the final ones – from the original assumptions to the conclusions.

4) Verify the conclusion: Once the final beliefs have been achieved, we can verify whether the goals of authentication protocols are held or whether there exist flaws in the protocol.

7.1.4 BAN-like logic

The BAN logic is a simple but effective formal approach. Although the BAN logic does not have a mechanism for directly finding a flaw in a protocol, it has been very successful in uncovering implicit assumptions that were missing from protocol specifications but are actually necessary in order for the statement of the desired goal to hold true. A missing assumption can often lead to a discovery of a flaw. However, flaw-finding in this way depends highly on the analyzer's experience, insight or even luck[1].

The idealization steps of a protocol in BAN logic are quite straightforward. Mao observes that the BAN logic provides a context-free procedure for protocol idealization. For instance, the protocol step may tell B that K_{AB} is a key to communicate with A which is encrypted under the shared key K_{BS} between B and the trusted third party S. This step is idealized as $A \Rightarrow B : \{A \overset{K}{\longleftrightarrow} B\}_{K_{BS}}$ that means the key K is a good key for communication between A and B. This is in fact a dangerous simplification for the meaning is induced from a context-free procedure. The idealization of a protocol in BAN logic has made some differences, by omitting or adding some meaning, between the original protocol and the idealized protocol.

The BAN logic is also lack of a formal definition for an underlying semantics upon which the soundness of the axiom systems is based[1].

More important, the BAN logic believes that if X has not been sent in a message at any time before the current run of the protocol, then X is fresh. The BAN logic believes that this is usually true for a nonce while it may not be in fact. Hence, the correctness proof of a protocol in the BAN logic may not convince the security of the protocol, and the examples include the Needham-Schroeder public-key protocol[7, 8].

To improve the BAN logic, a large body of extensions have been proposed, including the GNY logic, the AT logic, the SVO logic, etc. They are usually referred to as BAN-like logic.

The GNY logic[9] was proposed by Gong, Needham and Yahalom in 1990. The GNY logic extension includes the great change of the BAN logic axiom systems, and the notion of a message being possessed by a principal as a result of inventing or recognizing the messages. The BAN logic has only more than 20 axioms while GNY logic has eight classes and total 72 axioms. These axioms are more detailed in describing the characteristics of the cryptographic functions and the logic formulae.

The AT logic[10] was proposed by M. Abadi and M. R. Tuttle in 1991. It has identified many sources of the past confusion for the semantics of the BAN logic. And it has improved the BAN logic's syntax and inference rules, and has extended its applicability. The greatest improvement made by the AT logic is its treatment of belief as a form of resource-bounded, defeasible knowledge.

The SVO logic[11] was proposed by P.F. Syversoon and P.C. van Oorschot

in 1994. It takes the advantage of BAN logic, GNY logic and AT logic, and its rules and axioms are simple and efficient. The SVO logic is good at examining public-key based authenticated key establishment protocols and it has been used to analyze three Diffie-Hellman based key agreement protocols which include the STS Protocol[1].

Although the BAN logic, including the BAN-like logic, lacks a formal semantic model for the soundness of its logic, it is undoubtedly an important seminal work. It has inspired the start of formal approaches to the analysis of authentication protocols.

7.2 Model checking

Model checking is a methodology that models a complex system into a (finite) state system, and the analysis of the behavior of a complex system usually involves a state space exploration to check whether or not certain properties will be satisfied. It could be a methodology for guarding against certain undesirable properties so that they never occur, or one for making sure that certain desirable properties do eventually occur. The model checking technique could be applied to the analysis of authentication protocols.

Indeed, a model checking technique can only deal with systems which can be modeled into finite state systems. The protocol analysis based on model checking models an authentication protocol into a (finite) state system, and the operational behavior of a finite state system is modeled by a finite state transition system which can make state transitions by interacting with its environment on a set of events. Not only the behavior of the legitimate participants' roles specified in a protocol will be modeled, but also some typical behavior of the adversary will be modeled. Each state of the state system is interpreted mechanically into (or assigned with) a logical formula, and the target property of a protocol is also explicitly interpreted into a logical formula. Trace is a sequence of events that represents a valid history of a system running the protocol. The procedure of finding the trace is to use a state exploration tool to discover if the system can enter an insecure state, that is, whether there is an attack upon the protocol. The protocol analysis based on model checking checks all possible traces of whether an undesirable property of a protocol will be satisfiable, and here satisfiability means that the target formula is a logical consequence of a formula in a trace. If there exists a trace to a target formula modeling an undesirable property, then this provides an explicit description of a system error. Therefore, a model checking approach can work in the mode for finding an error in a system.

The model checking methodology is efficient in finding flaws in authentication protocols, e.g., Lowe found the famous attack on the Needham-Schroeder public-key protocol using the notion CSP (Communication Sequence Process) and the model checker FDR in 1996[12]. The famous model checking

approaches include the FDR model checker, Interrogator system[13], MurΦ model checker[14], NRL (Naval Research Laboratory) protocol analyzer[15–17], etc.

The main problem of the model checking methodology is how to produce a finite state system running the protocol without increasing or reducing the security of the protocol. Model checking has proved to be a very successful approach to analyze security protocols, but it tends to model a protocol by a huge state space, hence model checking methods frequently face a "state explosion" problem where a protocol maps to a system of too many states so that the computing resources cannot cope with it. In authentication protocol analysis, this limitation requires that an attacker should be a computationally bounded principal: behavior which relates to unbounded computational power will not be considered. To avoid state explosion, model checking models a protocol to a particular small system, and suppose that if there is no attack on the particular small system analyzed, there may not be an attack on some larger system running the same protocol under some sufficient conditions. In other words, for such protocols, model checking is complete. This simplicity may lead flaws in security analysis. And more, usually, only some typical behavior of an attacker will be modeled while the ability of an attacker is developing.

7.3 Theorem proving

A theorem proving approach can be described as a set of algebraic or logical formulae which are defined for use in system behavior description or in statement construction (premises or consequences to be derived), a set of axioms which are postulated to derive new formulae from known ones, a set of theorems which are desired behavior or properties of a system to be proved, and the proofs of theorems by using premise axioms and other already proved theorems. Sometimes, the proof process in a theorem proving approach can be mechanized if there are certain rules for applying axioms or theorems. The proof tool is then called a (mechanical) theorem prover.

Strand space[6] is a famous theorem proving approach proposed by Fabrega, Herzog and Guttman in 1998 for stating and proving correctness properties of cryptographic protocols. In strand space approach, a strand is a sequence of events; it represents either the execution of legitimate party in a security protocol or else a sequence of actions by an attacker. For a legitimate participant, each strand is a sequence of messages it sends and receives; it represents the action of that party in a particular run of the protocol. A collection of strands for various legitimate protocol parties with the attacker strands defines strand space. A strand space represents a graph structure generated by the causal interaction. The set of actions that principals can take during the execution of a protocol includes actions such as send (denoted by

+) and receive (denoted by -). A bundle represents a full protocol exchange that consists of a number of strands hooked together where one strand sends a message and another strand receives the same message. A node is a pair $\langle s, i\rangle$ with $s \in \Sigma$ (Σ is a set of all strands) and i is an integer representing the node position. There is an edge $n_1 \rightarrow n_2$ if and only if $term(n_1) = +a$ and $term(n_2) = -a$ for some $a \in$ *set of all messages*. When $n_1 = \langle s, i\rangle$ and $n_2 = \langle s, i+1\rangle$ are nodes, there is an edge $n_1 \Rightarrow n_2$. If C is a bundle, then the C-height of a strand s is the largest i so that $\langle s, i\rangle \in C$. Bundle is a finite acyclic subgraph that captures the natural causal precedence relation among nodes as defined by the edges $\rightarrow$ and $\Rightarrow$. If there are terms a_1 and a_2, then a_1 is a subterm if it appears in a_2. A term t originates on a node n in a strand s if n is the first positive node containing t in s. The correct proof of a cryptographic protocol in strand space is expressed in terms of the connections between strands of different kinds and it is given in the combination with a range of other protocols. For details of strand space, refer to [6, 18].

Strand space provides a clear semantics to the assumption that certain data items, such as nonce and session keys, are fresh and never arise in more than one protocol run; it gives a detailed model of an attacker which is independent of the concrete protocol analyzed; it also gives various correct definitions, hence the correctness proofs have become convenient. Furthermore, strand space can be used not only by hand but also as a special purpose tool because of its simplicity and elegance[19]. Theoretical results can also be expressed using strand space as a framework. However, the proof process under strand space is related to the concrete formalization of an attacker as model checking does, although the attacker model is independent of the concrete protocol analyzed.

7.4 Belief multisets based on trusted freshness

Review that we have presented a freshness principle in Chapter 5, also a manual analysis method based on the freshness principle in Chapter 6, and we will now introduce a belief multiset formalism[20–23] in this section to show the efficiency and the rigorous security analysis idea based on trusted freshness, which is suited for automation.

In this section we describe the syntax and semantics of the belief multiset formalism, also its rules, and the transformations that we apply to protocols before their formal analysis.

7.4.1 Belief logic language

Let's review the notation introduced in Section 3. The notations in the belief multiset formalism are indicated in Table 7.1.

Table 7.1 Notations

Notations	Description
ρ	Arbitrary principal, ranges over the participants of the protocol run.
P_i or P_j	A principal indexed by subscript in a protocol run.
S	Trusted third party.
τ	Arbitrary time, a moment, not a period of time.
$t_0, t_1, t_2, \ ..., \ t_\$$	Time before the start of a protocol run, time at the first message round (e.g., Message 1)..., time at the termination of a protocol run respectively.
N or N'	Arbitrary freshness identifier, it can be a nonce, a timestamp, a session key or a shared part of a session key.
N_{P_i}	A freshness identifier invented by subscript principal P_i.
k	A cryptographic key; k^{-1}, the inverse of k. In shared-key case, k and k^{-1} are equal.
K	A long-term key, it may be a private key in shared-key schemes, a public-key or a private key in public-key schemes.
$K_{P_iP_j}$ (or K_{P_iS})	The long-term key shared between the principal P_i and P_j (or S) in shared-key case.
K_{P_i}, $K_{P_i}^{-1}$	The public-key and private key subscripted by principal identity respectively in public-key case. For example, K_{P_i} and $K_{P_i}^{-1}$, the key pairs of the principal P_i.
φ, ψ or Γ	A fact, which has the value of *True* or *False*.
$\neg$	A logical operator which is the same as negation used in logic.

A belief is a trust about the security property of a cryptographic protocol in belief multiset formalism, the beliefs defined in this book include fragment belief, expectation belief, liveness belief, confidentiality belief, freshness belief and association belief, and the types of beliefs could be further extended to meet various applications in the future.

Definition 7.1 $\sim \{...N, N'...\}_k$ is a fragment belief owned by someone such as P_i who asserts the binding of a new freshness identifier N' with a trusted freshness identifier N.

Definition 7.2 $\prec \{N, P_j\}$ is an expectation belief owned by someone such as P_i who asserts that only the partner P_j can obtain the freshness identifier N from a one-way transformation that is sent by P_i.

$\prec \{N, P_i, P_j\}$ is also an expectation belief owned by someone such as P_i as $\prec \{N, P_j\}$ is, and the explicit identity of P_i in $\prec \{N, P_i, P_j\}$ indicates that N must be associated with the session related to P_i.

Definition 7.3 $Key(P_i, k)$ means the principal P_i has the knowledge of k. On the contrary, $\neg(Key(P_i, k))$ means the principal P_i does not have the knowledge of k.

Definition 7.4 $\langle ...\rho \rangle$ is a liveness belief owned by someone such as P_i about the liveness of a principal ρ, the default is $\langle ...\rho \rangle$. If the intended opposite partner ρ is in lively correspondence in this session, then P_i has $\langle 1\rho \rangle$, that is,

Existing(ρ) (see Definition 6.5).

Definition 7.5 $\langle ...{}_1...{}_2\rho\rangle$ is an origin liveness belief owned by someone such as P_i about the origin liveness of a principal ρ, which is a special type of liveness belief with origin identity requirement. The default of the origin liveness belief is *Belief*(P_i, *Existing*(ρ)).

"$...{}_1$" states the origin principal of the liveness property.

"$...{}_2$" states the liveness property about the opponent participant. If the intended opposite partner ρ is in lively correspondence in this session, and P_i has corroborative evidence that P_j is in lively correspondence with P_i, then P_i has $\langle P_i\ 1\rho\rangle$, that is, *Belief*($P_i$, *Originexisting*($\rho$)) (see Definition 6.6).

Definition 7.6 $\langle ...{}_1...{}_2N...{}_3\rangle$ is a belief owned by someone such as P_i about the freshness identifier N, including the confidentiality belief, freshness belief and association belief. The default of the beliefs about the freshness identifier N is $\langle .....N...\rangle$ (it is usually written as $\langle ...N...\rangle$ for interest of concision).

1) "$...{}_1$" states confidentiality belief owned by someone such as P_i about the freshness identifier N. If freshness identifier N is confidential in this run from the point of view of P_i, then P_i has $\langle 1...N...\rangle$, that is, *Belief*(P_i, *Secret*(N)); if freshness identifier N is open in this run, then P_i has $\langle 0...N...\rangle$.

2) "$...{}_2$" states freshness belief owned by someone such as P_i about the freshness identifier N. If N is fresh, then P_i has $\langle ...1N...\rangle$, that is, *Belief*(P_i, *Fresh*(N)); if the freshness of the freshness identifier N is not clear, then P_i has $\langle .....N...\rangle$. In belief multiset formalism, if N is a long-term key between the principal P_i and P_j in this run, we use $\langle 11NP_iP_j\rangle$ to express this security property.

3) "$...{}_3$" states the association belief owned by someone such as P_i about the freshness identifier N. If freshness identifier N is associated with a participant P_i (or also with P_j) of a protocol run, then P_i has $\langle .....NP_i\rangle$ (or $\langle .....NP_iP_j\rangle$), that is, *Belief*($P_i$, *Associate*($N, P_i$)) or *Belief*($P_i$, *Associate*($N, P_i, P_j$)).

When N is a key in a belief $\langle 1...N...{}_3\rangle$, then only the principals indicated in "$...{}_3$" know this key N.

In the interest of concision, the confidentiality belief, freshness belief and association belief about a freshness identifier are often expressed in the same fact, such as $\langle 01NP_i\rangle$.

Definition 7.7 Suppose ρ is a fact, then $B_{\rho,\tau}(\varphi)$ means that the principal ρ believes φ is true at time τ.

In particular, $B_{P_i,\tau_1}(B_{\rho,\tau}(\varphi))$ means that the principal P_i at time τ_1 believes the fact: the principal ρ believes φ is true.

Example 7.1 Here are some examples of beliefs in the belief multiset formalism:

— $B_{P_i,t_1}(\langle 1P_j\rangle)$: at time t_1, principal P_i believes the liveness of the opponent participant P_j.

— $B_{P_i,t_4}(\langle P_i 1 P_j \rangle)$: at time t_4, principal P_i believes the liveness of the opponent participant P_j and P_j has shown its presence specially for P_i.
— $B_{P_i,t_0}(\langle 11 K_{P_i S} P_i S \rangle)$: at time t_0, principal P_i believes that the fact $\langle 11 K_{P_i S} P_i S \rangle$ is true. That is, P_i believes that $K_{P_i S}$ is a shared long-term key between P_i and the trusted third party S.
— $B_{P_i,t_0}(\langle 1...k_{P_i P_j}... \rangle)$: at time t_0, principal P_i believes that the fact $\langle 1...k_{P_i P_j}... \rangle$ is true, that is, the key $k_{P_i P_j}$ is confidential. This is also a general premise for all freshness identifiers from the start of the protocol run until the confidentiality property has been changed in the subsequent steps of the protocol run.
— $B_{P_j,t_3}(\langle ...k_{P_i P_j} P_i P_j \rangle)$: at time t_3, principal P_j believes that the fact $\langle ...k_{P_i P_j} P_i P_j \rangle$ is true, that is, the key $k_{P_i P_j}$ is associated with both principals P_i and P_j.
— $B_{P_i,t_2}(\langle 11 k_{P_i P_j} P_i P_j \rangle)$: at time t_2, principal P_i believes that the fact $\langle 11 k_{P_i P_j} P_i P_j \rangle$ is true. That is, P_i believes that $k_{P_i P_j}$ is confidential and fresh, and P_i also believes that both P_i and P_j know this key $k_{P_i P_j}$. Actually, this assertion includes four beliefs about the freshness identifier $k_{P_i P_j}$: $B_{P_i,t_2}(\langle 1...k_{P_i P_j}... \rangle)$, $B_{P_i,t_2}(\langle ...1 k_{P_i P_j}... \rangle)$, $B_{P_i,t_2}(\langle ...k_{P_i P_j} P_i \rangle)$ and $B_{P_i,t_2}(\langle ...k_{P_i P_j} P_j \rangle)$.
— $B_{A,t_0}(\neg Key(I, K_B^{-1}))$: at time t_0, A believes that the fact $Key(I, K_B^{-1}))$ is not true, namely, A believes that I doesn't have the key K_B^{-1}. t_0 means the belief $B_{A,t_0}(\neg Key(I, K_B^{-1}))$ is a premise of this protocol run.
— $B_{A,t_1}(\langle 01 N \rho \rangle)$: at time t_1, the principal A believes the fact that $\langle 01 N \rho \rangle$ is true. That is, A believes that N is not confidential, but N is fresh and associated with ρ in the protocol run.

From the above definitions we are able to make the expression of the security properties clear and intuitive.

Definition 7.8 A belief multiset is an unordered collection of beliefs owned by a legitimate participant. We use $b_{\rho,\tau}$ to express the belief multiset owned by the principal ρ at time τ. A belief (such as a liveness belief, a confidentiality belief, a freshness belief and an association belief) is an element of a belief multiset, and we often refer it to a multiset element. In a belief multiset, the number of the multiset elements and the types of the beliefs are not limited. The typical form of a belief multiset is $\lfloor \langle ...N... \rangle, \ ... \ , \langle ...P_j \rangle \rfloor$.

Definition 7.9 We define

$$\begin{aligned} & B_{\rho,\tau}(\lfloor \langle ...N_1... \rangle, ..., \langle ...N_l... \rangle, \langle ... \ P_i \rangle, ..., \langle ... \ P_j \rangle \rfloor) \\ \triangleq \ & B_{\rho,\tau}(\langle ...N_1... \rangle) \wedge ... \wedge B_{\rho,\tau}(\langle ...N_l... \rangle) \wedge B_{\rho,\tau}(\langle ...P_i \rangle) \wedge ... \wedge \\ & B_{\rho,\tau}(\langle ...P_j \rangle), \end{aligned}$$

where $\langle ...N_1... \rangle$ is the belief about the freshness identifier N_1 (especially the new session key or the shared part of a new session key) including the confidentiality belief, the freshness belief and the association belief about N_1; $\langle ...N_l... \rangle$ is the belief about the freshness identifier N_l where l is a natural

number, this belief is similar to that about N_1; $\langle ...P_i \rangle$ and $\langle ...P_j \rangle$ are the liveness beliefs about the principal P_i and P_j respectively.

Note that the types of the multiset elements, the properties of the multiset elements, the number of the multiset elements are various for each belief multiset owned by some principal, hence we use multiset instead of set to express the security properties of a cryptographic protocol.

Example 7.2 Here are some examples of the belief multisets:
— $b_{P_i,t_0} = \lfloor \langle 11K_{P_iS}P_iS \rangle, \langle 11N... \rangle \rfloor$: at time t_0, principal P_i has the confidentiality belief, the freshness belief and the association belief about the shared long-term key K_{P_iS}, and P_i also has the confidentiality belief and the freshness belief about the freshness identifier N. Here $\langle 11K_{P_iS}P_iS \rangle$ and $\langle 11N... \rangle$ are the multiset elements in this belief multiset b_{P_i,t_0}.
— $b_{P_i,t_\$} = \lfloor \langle 11k_{P_iP_j}... \rangle, \langle 11N... \rangle \rfloor$: at time $t_\$$, namely at the end of the protocol run, P_i believes that the new session key $k_{P_iP_j}$ and the freshness identifier N are confidential and fresh, but P_i is not sure whether $k_{P_iP_j}$ is for the principals P_i and P_j or not.
— $b_{A,t_\$} = \lfloor \langle 11kAB \rangle, \langle 1B \rangle \rfloor$: at the end of the protocol run, A believes that the new session key k is confidential, fresh, and associated with both principals A and B, and A also believes that the principal B is in lively correspondence in this session.

7.4.2 Logical postulate

7.4.2.1 Operation rule

Suppose φ and ψ are facts. The facts in belief multiset formalism obey the following operation rules:
R1 : From $\vdash \varphi$ and $\vdash (\varphi \Rightarrow \psi)$, infer $\vdash \psi$
R2 : From $\vdash \varphi$ infer $B_{p,t}\varphi$

$\vdash \varphi$ means that fact φ is valid at all time. For example, φ can be a theorem that is derivable from axioms alone. R1 is the modus ponens and states that if φ can be deduced and $\varphi \Rightarrow \psi$ can be deduced, then ψ can also be deduced. R2 is the generalization rule, which states that if φ is a theorem then the principal ρ believes that the statement φ is true at time t.

Example 7.3 Suppose K_{P_i} is the public-key of the principal P_i, suppose everybody including the adversary I knows this public-key K_{P_i}, namely, $\langle 01K_{P_i}\rho \rangle$ is a theorem, then according to the operation rule R2, an arbitrary principal ρ such as P_j will believe that the statement $\langle 01K_{P_i}\rho \rangle$ is true at the start of the protocol run, that is $B_{P_j,t_0}(\langle 01K_{P_i}\rho \rangle)$.

Definition 7.10 (multiset element operation) Suppose ρ is an arbitrary principal, P_i and P_j are given principals, and N is an arbitrary freshness

identifier in a protocol run. $\alpha, \beta, \delta \in \{0, 1, \text{"..."}\}$, and $a, b, d \in \{0, 1\}$. The multiset element operations are defined as follows:

1) $\cup$ **operator**

— For the liveness belief of the principal ρ,

$$\langle \alpha\rho \rangle \cup \langle \beta\rho \rangle \triangleq \langle \delta\rho \rangle,$$

we have $\delta = 1$ if and only if at least one of $\alpha, \beta = 1$.

— For the freshness belief of the freshness identifier N,

$$\langle \ldots_1 \alpha N \ldots_3 \rangle \cup \langle \ldots_1 \beta N \ldots_3 \rangle \triangleq \langle \ldots_1 \delta N \ldots_3 \rangle,$$

we have $\delta = 1$ if and only if at least one of $\alpha, \beta = 1$.

2) $\cap$ **operator**

— For the confidentiality belief of the freshness identifier N,

$$\langle \alpha \ldots_2 N \ldots_3 \rangle \cap \langle \beta \ldots_2 N \ldots_3 \rangle \triangleq \langle \delta \ldots_2 N \ldots_3 \rangle,$$

we have $\delta = 1$ if and only if both $\alpha, \beta = 1$.

3) $+$ **operator**

— For the association belief of the freshness identifier N about a same principal,

$$\langle \ldots_1 \ldots_2 N\rho^a \rangle + \langle \ldots_1 \ldots_2 N\rho^b \rangle \triangleq \langle \ldots_1 \ldots_2 N\rho^d \rangle,$$

we have d=1 if and only if at least one of a, b=1. Here ρ^1 means the identity of the principal ρ ($\langle \ldots_1 \ldots_2 N\rho \rangle$), and ρ^0 means "...$_3$" ($\langle \ldots_1 \ldots_2 N \ldots_3 \rangle$), that is, the principal ρ is not associated with N, hence N may not be a new freshness identifier intended for the session related to the principal ρ.

— For the association belief of the freshness identifier N about various principals,

$$\langle \ldots_1 \ldots_2 NP_i \rangle + \langle \ldots_1 \ldots_2 NP_j \rangle \triangleq \langle \ldots_1 \ldots_2 NP_iP_j \rangle,$$

where P_i and P_j are various principals.

Definition 7.11 (belief multiset operation) Suppose ρ is an arbitrary principal, P_i and P_j are given principals, N_1 and N_2 are arbitrary freshness identifiers in a protocol run, $b_{\rho,\tau}$ is a belief multiset of ρ at time τ.

Suppose we have the belief multiset $b_{\rho,\tau} = \lfloor \langle ...N_1... \rangle, ..., \langle ... P_i \rangle \rfloor$, and the multiset elements $\langle ...N_2... \rangle$ and $\langle ... P_j \rangle$, we define

$$b_{\rho,\tau} + \langle ...N_2... \rangle + \langle ... P_j \rangle \triangleq \lfloor \langle ...N_1... \rangle, ..., \langle ... P_i \rangle, \langle ...N_2... \rangle, \langle ... P_j \rangle \rfloor$$

Example 7.4 Suppose P_i, P_j are principals, and N is a freshness identifier in a protocol run, then we have:

— $\lfloor \langle ...1N... \rangle, ..., \langle ...P_j \rangle \rfloor + \langle ...N... \rangle + \langle ...P_j \rangle = \lfloor \langle ...1N... \rangle, ..., \langle ...P_j \rangle \rfloor$

— $\lfloor\langle ...N...\rangle, ..., \langle 1P_j\rangle\rfloor + \langle ...N...\rangle) + \langle ...P_j\rangle = \lfloor\langle ...N...\rangle, ..., \langle 1P_j\rangle\rfloor$

— $\lfloor\langle 1...N...\rangle, ..., \langle ...P_j\rangle\rfloor + \langle 0...N...\rangle + \langle ...P_j\rangle = \lfloor\langle 0...N...\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle ...N...\rangle, ..., \langle ...P_j\rangle\rfloor + \langle 0...N...\rangle + \langle ...P_j\rangle = \lfloor\langle 0...N...\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle 1...N...\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...N...\rangle + \langle ...P_j\rangle = \lfloor\langle 1...N...\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle ...N...\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...NP_i\rangle + \langle ...P_j\rangle = \lfloor\langle ...NP_i\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle ...N...\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...NP_j\rangle + \langle ...P_j\rangle = \lfloor\langle ...NP_j\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle ...NP_i\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...NP_j\rangle + \langle ...P_j\rangle = \lfloor\langle ...NP_iP_j\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle ...NP_iP_j\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...N...\rangle) + \langle ...P_j\rangle = \lfloor\langle ...NP_iP_j\rangle, ..., \langle ...P_j\rangle\rfloor$

— $\lfloor\langle ...1NP_iP_j\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...N...\rangle + \langle 1P_j\rangle) = \lfloor\langle ...1NP_iP_j\rangle, ..., \langle 1P_j\rangle\rfloor$

— $\lfloor\langle ...1NP_iP_j\rangle, ..., \langle ...P_j\rangle\rfloor + \langle ...N'...\rangle + \langle 1P_i\rangle = \lfloor\langle ...1NP_iP_j\rangle, \langle ...N'...\rangle, ..., \langle 1P_i\rangle, \langle ...P_j\rangle\rfloor$

7.4.2.2 Inference rule

Inference rules, as described below, enable the derivation of new beliefs from current beliefs and incoming fresh messages. We believe that the security of a cryptographic protocol can only be derived from a trusted freshness identifier in trusted freshness analysis. The inference rules tend to be general statements of classical propositional calculus, and statements of the hypotheses underlying shared-key and public-key cryptographic communication protocols, and they are intuitively clear, so we just give a brief explanation here.

As an illustrated example of the hypotheses underlying shared-key and public-key cryptographic communication protocols, a signature could provide the evidence that a key is possessed by some party who signed it, and the trusted freshness identifier assures that the signature is a recent one, hence the recent signed signature could show the existence of this entity.

Recall that each security belief in the trusted freshness approach is only owned by a certain principal.

Let P_i and P_j range over principals, ρ be an arbitrary principal ranging over principals, and N and N' range over nonce.

Fragment rules

Fragment rules are rules about the liveness property of a principal, the freshness property and association property of a new freshness identifier N', which assert the binding of a new freshness identifier N' with a trusted freshness identifier N.

In general, if P_i asserts that the new freshness identifier N' is bound to a trusted freshness identifier N via a one-way transformation including N' and N, then P_i has the fragment $\sim \{...N, N'...\}_k$.

A1 (Fragment Rule)

The fragment rules A1(a) to A1(h) relate to the fragment held by the principal P_i who asserts that the new freshness identifier N' is bound to the

trusted freshness identifier N via a one-way transformation including N' and N.

A1(a) $-\{...N, N'...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11K_{P_iP_j}P_iP_j\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$
$\Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_iP_j}})$

The fragment rule A1(a) states: the principal P_i receives a term $\{...N, N'...\}_{K_{P_iP_j}}$ including the trusted freshness N (i.e., $B_{P_i,t}(\langle ...1N...\rangle)$), since P_i believes that $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, hence P_i asserts that it must be P_j who has sent this one-way transformation. Therefore, P_i believes that N' is generated for the same protocol run as N does, which is between P_i and P_j. That is, P_i asserts that N' and N are bound to the same protocol run.

A1(b) $-\{...N, N', P_j...\}_{K_{P_iS}} \wedge B_{P_i,t}(\langle 11K_{P_iS}P_iS\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$
$\Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_iS}})$

The fragment rule A1(b) states: the principal P_i receives a term $\{...\, N, N', P_j... \}_{K_{P_iS}}$ including the trusted freshness N, since P_i believes that K_{P_iS} is the shared long-term key between P_i and the trusted third party S, hence P_i asserts that it must be S who has sent this one-way transformation and has indicated that N' and N are bound to the same protocol run related to P_i and P_j (the explicit indication of the identity P_j).

A1(c) $-\{...N, N'...\}_{K_{P_i}} \wedge B_{P_i,t}(\langle 01K_{P_i}\rho\rangle) \wedge B_{P_i,t}(\langle 11K_{P_i}^{-1}P_i\rangle) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_i}})$

The fragment rule A1(c) states: the principal P_i receives a term $\{...N, N'...\}_{K_{P_i}}$ including the trusted freshness N, since P_i believes that $K_{P_i}^{-1}$ is the private key of P_i, hence P_i asserts that only P_i can decrypt this one-way transformation $\{...\, N, N'...\, \}_{K_{P_i}}$. Therefore, P_i believes that N' is generated for the same protocol run as N does, which is related to P_i. That is, P_i asserts that N' and N are bound to the same protocol run related to P_i.

A1(d) $-\{...N, N'...\}_{K_{P_j}^{-1}} \wedge B_{P_i,t}(\langle 01K_{P_j}\rho\rangle) \wedge B_{P_i,t}(\langle 11K_{P_j}^{-1}P_j\rangle) \wedge$
$B_{P_i,t}(\langle ...1NP_i...\rangle) \Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_j}^{-1}})$

The fragment rule A1(d) states: the principal P_i receives a term $\{...N, N'...\}_{K_{P_j}^{-1}}$ including the trusted freshness N, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, hence P_i asserts that it must be P_j who has generated this one-way transformation for the principal P_j. Therefore, P_i believes that N' is generated for the same protocol run as N does, which is related to P_j. That is, P_i asserts that N' and N are bound to the same protocol run related to P_j. Since P_i has $B_{P_i,t}(\langle ...1NP_i...\rangle)$, then P_i believes that N' is also related to the principal P_i. Hence, N' and N are bound to the same protocol run related to P_j and also P_i.

A1(e) $+\{...N, N'...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11K_{P_iP_j}P_iP_j\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$
$\Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_iP_j}})$

The fragment rule A1(e) states: the principal P_i sends out a term $\{...N, N'...\}_{K_{P_iP_j}}$ including the trusted freshness N, since P_i believes that $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, hence P_i asserts that only P_j can decrypt this one-way transformation. Therefore, P_i believes that N' is

generated for the same protocol run as N does, which is between P_i and P_j. That is, P_i asserts that N' and N are bound to the same protocol run.

A1(f) $+\{...N, N', P_j...\}_{K_{P_iS}} \wedge B_{P_i,t}(\langle 11 K_{P_iS} P_i S\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$
$\Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_iS}})$

The fragment rule A1(f) states: the principal P_i sends out a term $\{...\ N, N', P_j...\ \}_{K_{P_iS}}$ including the trusted freshness N, since P_i believes that K_{P_iS} is the shared long-term key between P_i and the trusted third party S, hence P_i asserts that only S can decrypt this one-way transformation and S will learn that N' and N are bound to the same protocol run related to P_i from K_{P_iS}, and P_j from the explicitly indicated identity of P_j.

A1(g) $+\{...N, N'...\}_{K_{P_j}} \wedge B_{P_i,t}(\langle 01 K_{P_j}\rho\rangle) \wedge B_{P_i,t}(\langle 11 K_{P_j}^{-1} P_j\rangle) \wedge$
$B_{P_i,t}(\langle ...1N P_i...\rangle) \Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_j}})$

The fragment rule A1(g) states: the principal P_i sends out a term $\{...N, N'...\}_{K_{P_j}}$ including the trusted freshness N, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, hence P_i asserts that only P_j can decrypt this one-way transformation $\{...\ N, N'...\ \}_{K_{P_j}}$. Therefore, P_i believes that N' is generated for the same protocol run as N is related to P_j. From the supposition $B_{P_i,t}(\langle ...1N P_i...\rangle)$, we know that N is also related to P_i, so N is related to both P_i and P_j. Hence, P_i asserts that N' and N are bound to the same protocol run between P_i and P_j.

A1(h) $+\{...N, N'...\}_{K_{P_i}^{-1}} \wedge B_{P_i,t}(\langle 01 K_{P_i}\rho\rangle) \wedge B_{P_i,t}(\langle 11 K_{P_i}^{-1} P_i\rangle) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_i}^{-1}})$

The fragment rule A1(h) states: the principal P_i sends out a term $\{...N, N'...\}_{K_{P_i}^{-1}}$ including the trusted freshness N, since every principal including P_i has the public-key K_{P_i} to check that N' and N are bound to the same protocol run as P_i expects, hence P_i asserts that N' is generated for the same protocol run as N does and the protocol run is related to P_i.

Expectation rules

Expectation rules are rules about the liveness property of a principal, the association property of a freshness identifier N, which assert that only the partner P_j can obtain the freshness identifier N from a one-way transformation that is sent by P_i.

The freshness identifier N is sent via a one-way transformation originated from P_i, and P_i expects that only the intended opposite partner P_j with the corresponding decryption key K_{P_j} can obtain N.

The expectation rules A2(a)-A2(c) refer to the expectation held by P_i who expects that only the intended opposite partner P_j with the relevant key can obtain the freshness identifier N.

A2 (Expectation Rule)

Suppose P_i believes that N is confidential and fresh, and $K_{P_iP_j}$ is the shared long-term key between P_i and P_j.

A2(a) $+\{...N...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11 K_{P_iP_j} P_i P_j\rangle) \wedge B_{P_i,t}(\langle 11N...\rangle)$

$\Rightarrow B_{P_i,t}(\prec \{N, P_i, P_j\})$

The expectation rule A2(a) states: the principal P_i believes that N is confidential and fresh, P_i encrypts the message $\{...N...\}$ using the shared long-term key $K_{P_iP_j}$ and then P_i sends $\{...N...\}_{K_{P_iP_j}}$ out. As P_i believes that only the intended opposite partner P_j has the corresponding decryption key $K_{P_iP_j}$ of this one way transformation $\{...N...\}_{K_{P_iP_j}}$, hence P_i expects that only the intended opposite partner P_j can obtain the freshness identifier N from the encryption, where N is related to a session of P_i.

Supose P_i believes that N is confidential and fresh, and $K_{P_j}^{-1}$ is P_j's private key.

A2(b) $+\{...N...\}_{K_{P_j}} \wedge B_{P_i,t}(\langle 11K_{P_j}^{-1}P_j\rangle) \wedge B_{P_i,t}(\langle 11N...\rangle)$
$\Rightarrow B_{P_i,t}(\prec \{N, P_j\})$

The expectation rule A2(b) states: the principal P_i believes that N is confidential and fresh, P_i encrypts the message $\{...N...\}$ using the public-key K_{P_j} of P_j and then P_i sends $\{...N...\}_{K_{P_j}}$ out. As P_i believes that only the intended opposite partner P_j has the corresponding decryption key $K_{P_j}^{-1}$ of this one way transformation $\{...N...\}_{K_{P_j}}$, hence P_i expects that only the intended opposite partner P_j can obtain the freshness identifier N from the encryption.

Supose P_i believes that N is confidential and fresh, K_{P_j} and $K_{P_j}^{-1}$ are key pair of P_j in public-key case.

A2(c) $+\{...P_i, N...\}_{K_{P_j}} \wedge B_{P_i,t}(\langle 11K_{P_j}^{-1}P_j\rangle) \wedge B_{P_i,t}(\langle 11N...\rangle)$
$\Rightarrow B_{P_i,t}(\prec \{N, P_i, P_j\})$

The expectation rule A2(c) states: the principal P_i believes that N is confidential and fresh, P_i encrypts the message $\{...P_i, N...\}$ using the public-key K_{P_j} of P_j and then P_i sends $\{...P_i, N...\}_{K_{P_j}}$ out. As P_i believes that only the intended opposite partner P_j has the corresponding decryption key $K_{P_j}^{-1}$ of this one way transformation $\{...P_i, N...\}_{K_{P_j}}$, hence P_i expects that only the intended opposite partner P_j can obtain the freshness identifier N from the encryption, and P_i knows that N is related to a session of P_i from the explicit indication of P_i in $\{...P_i, N...\}_{K_{P_j}}$.

Inference rules about the confidentiality of a freshness identifier

Confidentiality rule is a rule about the confidentiality property of a freshness identifier, which asserts that the freshness identifier is secret if it is transmitted in the form of an encryption that may not be decrypted by the attacker.

Recall the confidentiality lemma (Lemma 4.2), the confidentiality of a freshness identifier can be achieved by a participant P_i if the identifier is transmitted in the form of an encryption that cannot be decrypted by the attacker; if the freshness identifier is transmitted in the form of a plaintext or an encryption that may be decrypted by the attacker, then the freshness identifier is open, that is, it is known by the attacker.

Once a freshness identifier is open, it could not be confidential again in the subsequent protocol run.

A3 (Confidentiality Rule)

Suppose all freshness identifiers are confidential at the beginning of the protocol run, that is, $B_{P_i,t_0}(\langle 1...N...\rangle)$. The freshness identifier will become open if one of the following cases is met:

(1) The freshness identifier is transmitted in plaintext.

A3(a) $-\{...N...\} \Rightarrow B_{P_i,t}(\langle 0...N...\rangle)$

The confidentiality rule A3(a) states: if P_i receives a plaintext $\{...N...\}$ including a freshness identifier N, then P_i believes that N is open, that is, N is known by the attacker.

A3(b) $+\{...N...\} \Rightarrow B_{P_i,t}(\langle 0...N...\rangle)$

The confidentiality rule A3(b) states: if P_i sends a plaintext $\{...N...\}$ including a freshness identifier N, then P_i believes that N is open.

(2) The freshness identifier is transmitted in encryption, but the decryption key is known by adversary I.

Suppose that the corresponding decryption key k^{-1} of the encryption $\{...N...\}_k$ is known by adversary I.

A3(c) $-\{...N...\}_k \wedge B_{P_i,t}(Key(I, k^{-1})) \Rightarrow B_{P_i,t}(\langle 0...N...\rangle)$

The confidentiality rule A3(c) states: if P_i believes that the corresponding decryption key k^{-1} of the receiving encryption $\{...N...\}_k$ is known by an attacker, then P_i believes that the freshness identifier N could not be confidential since the attacker could get N from the encryption $\{...N...\}_k$ using k^{-1}.

A3(d) $+\{...N...\}_k \wedge B_{P_i,t}(Key(I, k^{-1})) \Rightarrow B_{P_i,t}(\langle 0...N...\rangle)$

The confidentiality rule A3(d) states: if P_i believes that the corresponding decryption key k^{-1} of the sending encryption $\{...N...\}_k$ is known by an attacker, then P_i believes that the freshness identifier N could not be confidential since the attacker could get N from the encryption $\{...N...\}_k$ using k^{-1}.

Inference rules about the liveness of a principal

1) General liveness rules.

Liveness rules are rules about the liveness property of a principal. Recall the liveness lemma (Lemma 4.1), the liveness of a principal P_j can be achieved by a participant P_i via a sent or received one-way transformation that includes a trusted freshness identifier of P_i, where the one-way transformation can only be accomplished by the principal P_j. P_i may have corroborative evidence that P_j is in lively correspondence with P_i if one of the following cases is met:

A4 (Liveness Rule)

(1) P_i has corroborative evidence that the received message must have been generated by P_j.

P_i receives an encryption including the trusted freshness N, and P_i believes that this encryption is encrypted by P_j using the shared long-term key

$K_{P_iP_j}$ between P_i and P_j.

A4(a) $-\{...N...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11K_{P_iP_j}P_iP_j\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$
$\Rightarrow B_{P_i,t}(\langle 1P_j\rangle)$

If P_i believes that N is fresh, $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, and P_i receives a one-way transformation $\{...N...\}_{K_{P_iP_j}}$, then P_i is entitled to believe that P_j is in lively correspondence with P_i in this session.

The liveness rule A4(a) states: principal P_i receives a term $\{...N...\}_{K_{P_iP_j}}$ including a trusted freshness identifier N, since P_i believes that $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, hence P_i asserts that it must be P_j who has encrypted the message including the trusted freshness identifier N using the key $K_{P_iP_j}$. Therefore, P_i believes that P_j is in lively correspondence with P_i in this session.

P_i receives a signature including the trusted freshness N, and P_i believes that this signature is signed by P_j using P_j's private key $K_{P_j}^{-1}$.

A4(b) $-\{...N...\}_{K_{P_j}^{-1}} \wedge B_{P_i,t}(\langle 01K_{P_j}\rho\rangle) \wedge B_{P_i,t}(\langle 11K_{P_j}^{-1}P_j\rangle) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\langle 1P_j\rangle)$

If P_i believes that N is fresh and $K_{P_j}^{-1}$ is the private key of P_j, and P_i receives a one-way transformation $\{...N...\}_{K_{P_j}^{-1}}$, then P_i is entitled to believe that P_j is in lively correspondence with P_i in this session.

The liveness rule A4(b) states: principal P_i receives a term $\{...N...\}_{K_{P_j}^{-1}}$ including the trusted freshness identifier N, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, hence P_i asserts that it must be P_j who has encrypted the message including the trusted freshness identifier N using the key $K_{P_j}^{-1}$. Therefore, P_i believes that P_j is in lively correspondence with P_i in this session.

(2) P_i has corroborative evidence that the sent fresh message has been processed by P_j.

Suppose P_i believes that N is fresh and P_i expects that only the intended opposite partner P_j can obtain the freshness identifier N. Subsequently, if P_i has received the term $\{...N...\}$ or $\{...N...\}_k$ and the correctness of N has been checked, then P_i is entitled to believe that P_j is in lively correspondence with P_i in this session.

A4(c) $-\{...N...\}_k \wedge B_{P_i,t}(\prec \{N, P_j\}) \wedge B_{P_i,t}(Key(P_i, k^{-1})) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\langle 1P_j\rangle)$

and

A4(c′) $-\{...N...\}_k \wedge B_{P_i,t}(\prec \{N, P_i, P_j\}) \wedge B_{P_i,t}(Key(P_i, k^{-1})) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\langle 1P_j\rangle)$

The liveness rule A4(c) states: principal P_i has the expectation $B_{P_i,t}(\prec \{N, P_j\})$ (or $B_{P_i,t}(\prec \{N, P_i, P_j\})$) which implies that only the intended opposite participant P_j can read the freshness identifier N. When P_i receives the term $\{...N...\}$ or $\{...N...\}_k$ including the trusted freshness identifier N, P_i asserts that it must be P_j who has sent the message including N, so P_i believes that P_j is in lively correspondence with P_i in this session.

More liveness rules could be given only if the cryptographic operation could provide a corroborative evidence that a principal is in lively correspondence with a challenge – a trusted freshness identifier N owned by the principal who wants to establish the liveness belief about its opponent participant.

2) Liveness rules with origin

P_i receives an encryption including the trusted freshness N, and P_i believes that this encryption is encrypted by P_j using the shared long-term key $K_{P_iP_j}$ between P_i and P_j.

A4(d) $-\{...N...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11 K_{P_iP_j} P_i P_j\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$
$\Rightarrow B_{P_i,t}(\langle P_i\ 1P_j\rangle)$

If P_i believes that N is fresh, $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, and P_i receives a one-way transformation $\{...N...\}_{K_{P_iP_j}}$, then P_i is entitled to believe that P_j is in lively correspondence with P_i in this session, and P_i has corroborative evidence that P_j is in lively correspondence specially with this origin participant P_i.

The liveness rule A4(d) states: principal P_i receives a term $\{...N...\}_{K_{P_iP_j}}$ including a trusted freshness identifier N, since P_i believes that $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, hence P_i asserts that it must be P_j who has encrypted the message including the trusted freshness identifier N using the key $K_{P_iP_j}$, and this is a special correspondence with P_i. Therefore, P_i believes that P_j is in lively correspondence specially with this origin participant P_i in this session.

P_i receives a signature including the trusted freshness N, and P_i believes that this signature is signed by P_j using P_j's private key $K_{P_j}^{-1}$.

A4(e) $-\{...P_iN...\}_{K_{P_j}^{-1}} \wedge B_{P_i,t}(\langle 01 K_{P_j} \rho\rangle) \wedge B_{P_i,t}(\langle 11 K_{P_j}^{-1} P_j\rangle) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\langle P_i\ 1P_j\rangle)$

If P_i believes that N is fresh and $K_{P_j}^{-1}$ is the private key of P_j, and P_i receives a one-way transformation $\{...P_iN...\}_{K_{P_j}^{-1}}$, then P_i is entitled to believe that P_j is in lively correspondence with P_i in this session, and P_i has corroborative evidence (the explicit indication of the identity P_i in $\{...P_iN...\}_{K_{P_j}^{-1}}$) that P_j is in lively correspondence specially with this origin participant P_i.

The liveness rule A4(e) states: principal P_i receives a term $\{...P_iN...\}_{K_{P_j}^{-1}}$ including the trusted freshness identifier N, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, hence P_i asserts that it must be P_j who has encrypted the message including the trusted freshness identifier N using the key $K_{P_j}^{-1}$, and this is a special correspondence with P_i. Therefore, P_i believes that P_j is in lively correspondence specially with this origin participant P_i in this session.

A4(f) $-\{...N...\}_k \wedge B_{P_i,t}(\prec \{N, P_i, P_j\}) \wedge B_{P_i,t}(Key(P_i, k^{-1})) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\langle P_i\ 1P_j\rangle)$

The liveness rule A4(f) states: principal P_i has the expectation $B_{P_i,t}(\prec \{N, P_i, P_j\})$ which implies that only the intended opposite participant P_j

can read the freshness identifier N related to P_i. When P_i receives the term $\{...N...\}$ or $\{...N...\}_k$ including the trusted freshness identifier N, P_i asserts that it must be P_j who has sent the message including N, and this is a special correspondence with P_i, so P_i believes that P_j is in lively correspondence specially with this origin participant P_i in this session.

Inference rules about freshness of the freshness identifier

Freshness rule is a rule about the freshness property of a freshness identifier, which asserts that the freshness identifier is new for this particular protocol run.

Recall the freshness lemma (Lemma 4.3), the freshness of a new freshness identifier could be achieved by a participant P_i when P_i has corroborative evidence that the identifier is new for this particular protocol run if one of the following cases is met:

1) The freshness identifier is a nonce new generated for this protocol run by the principal P_i itself or a timestamp falls in the acceptance time window;

2) The new freshness identifier is bound together with P_i's another trusted freshness identifier in a one-way transformation, where the one-way transformation can only be accomplished by the opponent participant P_j.

A5 (Freshness Rule)

(1) The freshness identifier is a timestamp T.

Suppose the timestamp difference between the initiator and the receiver is within the acceptance window, e.g., one second.

A5(a) $\pm\{...T...\} \Rightarrow B_{P_i,t}(\langle ...1T... \rangle)$

The generation rule (or the freshness rule) A5(a) states: if P_i exchanges a message $\{...T...\}$ including a freshness identifier T which is a timestamp, and the difference of T with the local clock is within the acceptance window, then P_i believes the freshness of the freshness identifier T.

(2) The freshness identifier N is a nonce generated by the participant P_i itself.

A5(b) $+\{...N_{P_i}...\} \Rightarrow B_{P_i,t}(\langle ...1N_{P_i}... \rangle)$

The generation rule (or the freshness rule) A5(b) states: if a freshness identifier N_{P_i} is invented by P_i for this particular protocol run, then P_i believes the freshness of the freshness identifier N_{P_i}.

(3) The new freshness identifier is bound to a trusted freshness identifier.

P_i has corroborative evidence that N' is fresh. Here the corroborative evidence may be a signature, a MAC or other one-way transformations including the new freshness identifier and the trusted freshness identifier.

If one of the above fragment rules is met, then P_i will have corroborative evidence that N' is fresh since N' and N are bound to the same protocol run.

A5(c) $B_{P_i,t}(\sim \{...N, N'...\}_k) \wedge B_{P_i,t}(\langle ...1N... \rangle) \Rightarrow B_{P_i,t}(\langle ...1N'... \rangle)$

The rule A5(c) states: since N is fresh in a particular run, and the principal P_i believes that the freshness identifier N' is bound to N in the same protocol run, then P_i has corroborative evidence that N' is also fresh.

Inference rules about association of the freshness identifier

Association rules are rules about the association property of a freshness identifier, which assert that the freshness identifier is related to some principals of a particular protocol run.

Recall the association lemma (Lemma 4.4), the association of a freshness identifier could be achieved by a participant P_i when P_i has corroborative evidence that the identifier is related to a particular protocol run related to P_i and/or P_j if one of the following cases is met:

A6 (Association Rule)

(1) Shared key case.

If P_i believes that $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, and P_i has exchanged a one-way transformation $\{...N...\}_{K_{P_iP_j}}$, then P_i is entitled to believe that the trusted freshness identifier N is associated with the protocol run between P_i and P_j.

A6(a) $\pm\{...N...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11 K_{P_iP_j} P_i P_j\rangle) \Rightarrow B_{P_i,t}(\langle ...N P_i P_j\rangle)$

The association rule A6(a) states: principal P_i exchanges a term $\{...N...\}_{K_{P_iP_j}}$ including a freshness identifier N, since P_i believes that $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, P_i believes that this protocol run related to N is associated with P_i and P_j.

P_i sends or receives a message including the identity of P_j, which is encrypted under the shared long-term key between P_i and the trusted third party S, then the freshness identifier N in this message is related to P_i and P_j.

A6(b) $\pm\{...N, P_j...\}_{K_{P_iS}} \wedge B_{P_i,t}(\langle 11 K_{P_iS} P_i S\rangle) \Rightarrow B_{P_i,t}(\langle ...N P_i P_j\rangle)$

The association rule A6(b) states: principal P_i exchanges a term $\{...N, P_j...\}_{K_{P_iS}}$ including a freshness identifier N, since P_i believes that K_{P_iS} is the shared long-term key between P_i and the trusted third party S, hence P_i believes that this protocol run related to N is associated with P_i. Since the identity of P_j is explicitly indicated in the term $\{...N, P_j...\}_{K_{P_iS}}$, P_i believes that N is also associated with the protocol run related to P_j.

(2) public-key case.

P_i receives a signature of P_j, then the freshness identifier N in this signature is related to P_j.

A6(c) $-\{...N...\}_{K_{P_j}^{-1}} \wedge B_{P_i,t}(\langle 01 K_{P_j} \rho\rangle) \wedge B_{P_i,t}(\langle 11 K_{P_j}^{-1} P_j\rangle)$
$\Rightarrow B_{P_i,t}(\langle ...N P_j\rangle)$

The association rule A6(c) states: principal P_i receives a term $\{...N...\}_{K_{P_j}^{-1}}$ including the freshness identifier N, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, P_i asserts that it must be P_j who has generated this one-way transformation for the protocol run related to P_j. Therefore, P_i believes that N is associated with P_j.

P_i receives a signature of P_j including the identity of P_i, then the freshness identifier N in this signature is also related to P_i from P_i's point of view.

A6(d) $-\{...N, P_i...\}_{K_{P_j}^{-1}} \wedge B_{P_i,t}(\langle 01 K_{P_j} \rho\rangle) \wedge B_{P_i,t}(\langle 11 K_{P_j}^{-1} P_j\rangle)$

$\Rightarrow B_{P_i,t}(\langle ...NP_iP_j\rangle)$

The association rule A6(d) states: principal P_i receives a term $\{...N, P_i...\}_{K_{P_j}^{-1}}$ including the identity of P_i, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, P_i asserts that it must be P_j who has generated this one-way transformation for the protocol run related to P_j. Since the identity of P_i is explicitly indicated in the term $\{...N, P_i...\}_{K_{P_j}^{-1}}$, P_i believes that N is also associated with P_i.

P_i receives an encryption under the public-key of P_i, then the freshness identifier N in this encryption is related to P_i.

A6(e) $-\{...N...\}_{K_{P_i}} \wedge B_{P_i,t}(\langle 01K_{P_i}\rho\rangle) \wedge B_{P_i,t}(\langle 11K_{P_i}^{-1}P_i\rangle)$
$\Rightarrow B_{P_i,t}(\langle ...NP_i\rangle)$

The association rule A6(e) states: principal P_i receives a term $\{...N...\}_{K_{P_i}}$ including the freshness identifier N, since P_i believes that $K_{P_i}^{-1}$ is the private key of P_i, hence P_i asserts that only P_i can decrypt this one-way transformation $\{...N...\}_{K_{P_i}}$. Therefore, P_i believes that N is associated with the protocol run related to P_i.

P_i sends an encryption including the identity of P_i using the public-key of the opponent P_j, then the freshness identifier N in this encryption is related to P_i.

A6(f) $+\{...P_i, N...\}_{K_{P_j}} \wedge B_{P_i,t}(\langle 01K_{P_j}\rho\rangle) \wedge B_{P_i,t}(\langle 11K_{P_j}^{-1}P_j\rangle)$
$\Rightarrow B_{P_i,t}(\langle ...NP_i\rangle)$

The association rule A6(f) states: principal P_i sends a term $\{...P_i, N...\}_{K_{P_j}}$ including the identity of P_i, since P_i believes that $K_{P_j}^{-1}$ is the private key of P_j, hence P_i asserts that only P_j can decrypt this one-way transformation $\{...P_i, N...\}_{K_{P_j}}$, but a legal attacker P_j may re-encrypt N and send it to some other hones participant, hence P_i isn't sure that N is associated with the protocol run related to P_j. Since the identity of P_i is explicitly indicated in the term $\{...P_i, N...\}_{K_{P_j}}$, P_i believes that N is associated with the protocol run related to P_j.

(3) Expectation association rule.

If P_i believes that N is fresh and it can only be obtained by the intended opposite partner P_j, and P_i receives a one-way transformation $\{...N...\}_k$ including N, then P_i is entitled to believe that N is associated with the protocol run related to P_j.

A6(g) $-\{...N...\}_k \wedge B_{P_i,t}(\prec \{N, P_i, P_j\}) \wedge B_{P_i,t}(Key(P_i, k^{-1})) \wedge$
$B_{P_i,t}(\langle ...1N...\rangle) \Rightarrow B_{P_i,t}(\langle ...1NP_j\rangle)$

The rule A6(g) states: principal P_i has the expectation $B_{P_{i,t}}(\prec\{N, P_i, P_j\})$ which implies that only the intended opposite participant P_j can read the freshness identifier N. When P_i receives the term $\{...N...\}_k$ including a trusted freshness identifier N, P_i asserts that N is related to a particular protocol run related to P_j. Therefore, P_i believes that N is associated with the principal P_j.

(4) Fragment association rule.

If P_i believes that N and N' are bound to the same protocol run, and

that the trusted freshness identifier N is bound to a protocol run between P_i and P_j, then P_i is entitled to believe that N' is bound to the same protocol run between P_i and P_j.

A6(h) $B_{P_i,t}(\sim \{...N, N'...\}_k) \wedge B_{P_i,t}(\langle ...1NP_iP_j\rangle) \Rightarrow B_{P_i,t}(\langle ...1N'P_iP_j\rangle)$

The rule A6(h) states: principal P_i believes that the new freshness identifier N' is bound to a trusted freshness identifier N, since N is fresh in a particular run between P_i and P_j, then N' is also fresh in the same particular run between P_i and P_j.

V. Other cryptographic operations which can provide evidence of P_j's (and/or P_i's) association with N.

7.5 Applications of belief multiset formalism

To analyze a cryptographic protocol, all one needs to do is to simply prove the security of a protocol via the manual security analysis method based on trusted freshness or the belief multiset formalism, then one gets to know whether a cryptographic protocol is secure or not in a realistic adversary-controlled network. The efficiency, rigorousness, automation possibility of the belief multiset formalism will be illustrated in this section via the analysis of the Needham-Schroeder public-key protocol.

The stepwise analysis based on belief multisets is refined as follows. Suppose there exists a protocol Π between A and B, the security goal of Π is to establish a new session key k between A and B to build a secure channel in an insecure network.

Specify the security goals to be reached based on the belief multisets

The sufficient and necessary conditions to guarantee the security of a cryptographic protocol are specified in Chapter 4, and they can be expressed as $b_{A,t_\$} = \lfloor \langle 11kAB\rangle, \langle 1B\rangle \rfloor$ and $b_{B,t_\$} = \lfloor \langle 11kAB\rangle, \langle 1A\rangle \rfloor$ in belief multiset formalism.

Specify the premises before the start of the cryptographic protocol

Recall that each participant has his own private key and all other parties' public-keys (respectively, the shared long-term key between co-operative principals or trusted third party) in public-key case (respectively, in shared key case). ρ is an arbitrary principal which ranges over participants of the protocol run including the attacker I.

1) public-key case:

$B_{A,t_0}(\langle 11K_A^{-1}A\rangle), B_{A,t_0}(\langle 11K_B^{-1}B\rangle), B_{A,t_0}(\langle 01K_A\rho\rangle), B_{A,t_0}(\langle 01K_B\rho\rangle)$

and $B_{B,t_0}(\langle 11K_B^{-1}B\rangle), B_{B,t_0}(\langle 11K_A^{-1}A\rangle), B_{B,t_0}(\langle 01K_A\rho\rangle), B_{B,t_0}(\langle 01K_B\rho\rangle)$.

2) Shared key case without trusted third party:

$B_{A,t_0}(\langle 11K_{AB}AB\rangle)$ and $B_{B,t_0}(\langle 11K_{AB}AB\rangle)$.

3) Shared key case with trusted third party:

$B_{A,t_0}(\langle 11 K_{AS} AS \rangle)$ and $B_{B,t_0}(\langle 11 K_{BS} BS \rangle)$.

Besides the above premises, key-establishment protocol has these assumptions:

$$b_{A,t_0} = \lfloor \langle 1...k... \rangle, \langle ...B \rangle \rfloor \text{ and } b_{B,t_0} = \lfloor \langle 1...k... \rangle, \langle ...A \rangle \rfloor.$$

Establish the security properties

Establish the security properties of a cryptographic protocol based on inference rules from current beliefs and incoming fresh messages that include a trusted freshness identifier.

Compare with the security goals established in step 1

The analysis results can either establish the correctness of the protocol when it is in fact correct, or identify the absence of the security properties and the structure to construct attacks based on the absence.

1) Absence of the liveness of a principal like A: impersonating A to launch an attack, e.g., the Otway-Rees protocol[24], revised Woo-Lam protocol[25];

2) Absence of the freshness of a freshness identifier: launching an attack by replaying the recorded one-way transformation with a compromised session key, e.g., the Needham-Schroeder shared key protocol[7, 26];

3) Absence of the association of a freshness identifier: launch an attack by making a legitimate principal like B confuse a key k' between I and A (or B) with the key between A and B, e.g., the Needham-Schroeder public-key protocol[7].

7.5.1 Analysis of Needham-Schroeder public-key protocol

The analysis of the original Needham-Schroeder public-key protocol and the analysis of the revised Needham-Schroeder public-key protocol are given in Subsections 7.5.1.1 and 7.5.1.2 respectively.

7.5.1.1 Analysis of the Original Needham-Schroeder public-key protocol

The Needham-Schroeder public-key protocol (say, Needham-Schroeder protocol)[7] is a well-known cryptographic protocol, whose intended goal is to establish a secret shared key k between two principals Alice (say, A) and Bob (say, B) via the shared parts N_A and N_B. N_A and N_B are nonce invented by A and B respectively. Let ρ be an arbitrary principal. Figure 7.1 illustrates the protocol part related to key establishment.

$$\begin{aligned} &\text{Message1} \quad A \rightarrow B: \quad \{N_A, A\}_{K_B} \\ &\text{Message2} \quad B \rightarrow A: \quad \{N_A, N_b\}_{K_A} \\ &\text{Message3} \quad A \rightarrow B: \quad \{N_B\}_{K_B} \end{aligned}$$

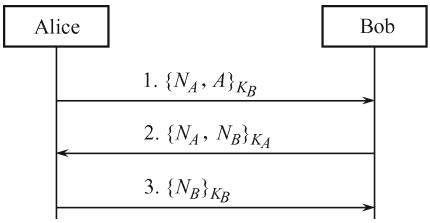


Fig. 7.1 The Needham-Schroeder public-key protocol.

Specify the security goals

$$b_{A,t_\$} = \lfloor\langle 11N_AAB\rangle, \langle 11N_BAB\rangle, \langle 1B\rangle\rfloor,$$
$$b_{B,t_\$} = \lfloor\langle 11N_AAB\rangle, \langle 11N_BAB\rangle, \langle 1A\rangle\rfloor.$$

Specify the premise before the start of the protocol

$$b_{A,t_0} = \lfloor\langle 11K_A^{-1}A\rangle, \langle 11K_B^{-1}B\rangle, \langle 01K_A\rho\rangle, \langle 01K_B\rho\rangle, \langle 1...\ N_A...\ \rangle, \langle 1...\ N_B...\ \rangle, \langle ...\ B\rangle\rfloor,$$
$$b_{B,t_0} = \lfloor\langle 11K_B^{-1}B\rangle, \langle 11K_A^{-1}A\rangle, \langle 01K_B\rho\rangle, \langle 01K_A\rho\rangle, \langle 1...\ N_A...\ \rangle, \langle 1...\ N_B...\ \rangle, \langle ...\ A\rangle\rfloor.$$

Establish the security properties of the Needham-Schroeder protocol

Step 1. Upon receiving Message 1 $A \rightarrow B : \{N_A, A\}_{K_B}$

No.	Security properties	Term	Applied
(1)	$+\{...N_A...\} \Rightarrow B_{A,t_1}(\langle ...1N_A...\rangle)$	$+\{...N_A...\}$	Rule A5(b)
(2)	$B_{A,t_0}(\langle 11K_B^{-1}B\rangle) \Rightarrow B_{A,t_0}(\langle 11K_B^{-1}...\rangle)$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(3)	$B_{A,t_0}(\langle 11K_B^{-1}B\rangle) \Rightarrow B_{A,t_0}(\neg Key(I, K_B^{-1}))$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(4)	$+\{...N_A...\}_{K_B} \wedge B_{A,t_0}(\neg Key(I, K_B^{-1})) \Rightarrow B_{A,t_1}(\langle 1...N_A...\rangle)$	$+\{...N_A...\}_{K_B}$	(2),(3), A3(d)
(5)	$+\{...A, N_A...\}_{K_B} \wedge B_{A,t_0}(\langle 01K_B\rho\rangle) \wedge B_{A,t_0}(\langle 11K_B^{-1}B\rangle) \Rightarrow B_{A,t_1}(\langle ...N_AA\rangle)$	$+\{...A, N_A...\}_{K_B}$	Premise, (1),A6(f)
(6)	$+\{...\ A, N_A...\ \}_{K_B} \wedge B_{A,t_0}(\langle 11K_B^{-1}B\rangle) \wedge B_{A,t_1}(\langle 11N_A...\rangle) \Rightarrow B_{A,t_1}(\prec \{N_A, A, B\})$	$+\{...A, N_A...\}_{K_B}$	Premise, (1),(4),A2(c)
(7)	$B_{B,t_0}(\langle 11K_B^{-1}B\rangle) \Rightarrow B_{B,t_0}(\langle 11K_B^{-1}...\rangle)$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(8)	$B_{B,t_0}(\langle 11K_B^{-1}B\rangle) \Rightarrow B_{B,t_0}(\neg Key(I, K_B^{-1}))$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(9)	$-\{...N_A...\}_{K_B} \wedge B_{B,t_0}(\neg Key(I, K_B^{-1})) \Rightarrow B_{B,t_1}(\langle 1...N_A...\rangle)$	$-\{...N_A...\}_{K_B}$	(7),(8), A3(c)

Step 2. Upon receiving Message 2 $B \rightarrow A : \{N_A, N_B\}_{K_A}$

No.	Security properties	Term	Applied
(10)	$+\{...N_B...\} \Rightarrow B_{B,t_2}(\langle ...1N_B... \rangle)$	$+\{...N_B...\}$	Rule A5(b)
(11)	$+\{...N_B...\}_{K_A} \wedge B_{B,t_0}(\neg Key(I, K_B^{-1}))$ $\Rightarrow B_{B,t_2}(\langle 1 \ldots N_B... \rangle)$	$+\{...N_B...\}_{K_A}$	(7),(8),A3(d)
(12)	$+\{...N_B...\}_{K_A} \wedge B_{B,t_0}(\langle 11K_A^{-1}A \rangle)$ $\wedge B_{B,t_2}(\langle 11N_B... \rangle) \Rightarrow B_{B,t_2}(\prec \{N_B, B, A\})$	$+\{...N_B...\}_{K_A}$	Premise,(10), (11), A2(b)
(13)	$B_{A,t_0}(\langle 11K_A^{-1}A \rangle) \Rightarrow B_{A,t_0}(\langle 11K_A^{-1}... \rangle)$	Definition of $\langle ..._1..._2N..._3 \rangle$	
(14)	$B_{A,t_0}(\langle 11K_A^{-1}A \rangle) \Rightarrow B_{A,t_0}(\neg Key(I, K_A^{-1}))$	Definition of $\langle ..._1..._2N..._3 \rangle$	
(15)	$-\{...N_B...\}_{K_A} \wedge B_{A,t_0}(\neg Key(I, K_A^{-1}))$ $\Rightarrow B_{A,t_2}(\langle 1...N_B... \rangle)$	$-\{...N_B...\}_{K_A}$	(13), (14), A3(c)
(16)	$-\{... N_A, N_B...\}_{K_A} \wedge B_{A,t_0}(\langle 01K_A\rho \rangle)$ $\wedge B_{A,t_0}(\langle 11K_A^{-1}A \rangle) \wedge B_{A,t_1}(\langle ...1N_A... \rangle)$ $\Rightarrow B_{A,t_2}(\sim \{...N_A, N_B...\}_{K_A})$	$-\{... N_A, N_B... \}_{K_A}$	Premise, (1), A1(c)
(17)	$B_{A,t_0}(\langle 11K_A^{-1}A \rangle) \Rightarrow B_{A,t_0}(Key(A, K_A^{-1}))$	Definition of $\langle ..._1..._2N..._3 \rangle$	
(18)	$-\{... N_A...\}_{K_A} \wedge B_{A,t_1}(\prec \{N_A, B\})$ $\wedge B_{A,t_0}(Key(A, K_A^{-1})) \wedge B_{A,t_1}(\langle ...1N_A... \rangle)$ $\Rightarrow B_{A,t_2}(\langle 1B \rangle)$	$-\{...N_A...\}_{K_A}$	(1),(6), (17),A4(c)
(19)	$-\{... N_A... \}_{K_A} \wedge B_{A,t_1}(\prec \{N_A, A, B\})$ $\wedge B_{A,t_0}(Key(A, K_A^{-1})) \wedge B_{A,t_1}(\langle ...1N_A... \rangle)$ $\Rightarrow B_{A,t_2}(\langle ...N_AB \rangle)$	$-\{...N_A...\}_{K_A}$	(1),(6), (17), A6(g)
(20)	$B_{A,t_2}(\langle 11N_AAB \rangle)$	(1),(4),(5),(19),R3,R4,R5	
(21)	$B_{A,t_2}(\sim \{...N_A, N_B...\}_{K_A})$ $\wedge B_{A,t_2}(\langle ...1N_AAB \rangle) \Rightarrow B_{A,t_2}(\langle ...1N_BAB \rangle)$	(16),(20),A6(h)	
(22)	$B_{A,t_2}(\langle 11N_BAB \rangle)$	(15),(21),R5	

Step 3. Upon receiving Message 3 $A \rightarrow B : \{N_B\}_{K_B}$

No.	Security properties	Term	Applied
(23)	$B_{B,t_0}(\langle 11K_B^{-1}B \rangle) \Rightarrow B_{B,t_0}(Key(B, K_B^{-1}))$	Definition of $\langle ..._1..._2N..._3 \rangle$	
(24)	$-\{... N_B... \}_{K_B} \wedge B_{B,t_2}(\prec \{N_B, B, A\})$ $\wedge B_{B,t_0}(Key(B, K_B^{-1})) \wedge B_{B,t_2}(\langle ...1N_B... \rangle)$ $\Rightarrow B_{B,t_3}(\langle 1A \rangle)$	$-\{...N_B...\}_{K_B}$	(12),(23), (10),A4(c)
(25)	$-\{... N_B... \}_{K_B} \wedge B_{B,t_2}(\prec \{N_B, B, A\})$ $\wedge B_{B,t_0}(Key(B, K_B^{-1})) \wedge B_{B,t_2}(\langle ...1N_B... \rangle)$ $\Rightarrow B_{B,t_3}(\langle ...1N_BA \rangle)$	$-\{...N_B...\}_{K_B}$	(12),(23), (10),A6(g)
(26)	$-\{...N_B...\}_{K_B} \wedge B_{B,t_0}(\langle 11K_B^{-1}B \rangle)$ $\wedge B_{B,t_0}(\langle 01K_B\rho \rangle) \Rightarrow B_{B,t_3}(\langle ...N_BB \rangle)$	$-\{...N_B...\}_{K_B}$	Premise, A6(e)
(27)	$B_{B,t_3}(\langle 11N_BB \rangle)$	(10),(11),(25),(26),R3,R4,R5	
(28)	$b_{A,t_\$} = \lfloor \langle 11N_AAB \rangle, \langle 11N_BAB \rangle, \langle 1B \rangle \rfloor$	(20),(22),(18)	
(29)	$b_{A,t_\$} = \lfloor \langle 1...N_A... \rangle, \langle 11N_BAB \rangle, \langle 1A \rangle \rfloor$	(9),(27),(24)	

Table 7.2 shows the analyzing result of the Needham-Schroeder public-key protocol based on belief multisets. Upon terminationd, A believes that Needham-Schroeder public-key protocol is secure, but B could not be assure the freshness of N_A, and also the association of the session key shared part N_A with the principal A and the principal B.

Table 7.2 Analysis of the Needham-Schroeder public-key protocol based on belief multisets

Step	A's beliefs			B's beliefs		
Message 1		$\langle 11 N_A A\rangle$			$\langle 1 ... N_A ...\rangle$	
Message 2	$\langle 1B\rangle$	$\langle 11 N_A AB\rangle$	$\langle 11 N_B AB\rangle$			$\langle 11 N_B ...\rangle$
Message 3				$\langle 1A\rangle$		$\langle 11 N_B AB\rangle$
At the End	$\langle 1B\rangle$	$\langle 11 N_A AB\rangle$	$\langle 11 N_B AB\rangle$	$\langle 1A\rangle$	$\langle 1 ... N_A ...\rangle$	$\langle 11 N_B AB\rangle$

Compare with the security goals established in step 1

1) Absence of the association of freshness identifier N_A with A and B for the principal B

→ Cheat B by confusing N'_A (between the attacker I and A) with N_A (between A and B).

2) The liveness of the principal A under the condition 1)

→ Suggest that A must be alive to act as an oracle in this attack, so it is a concurrent run attack.

From the above analysis, we can get the attack structure, as shown in Fig. 7.2: first, the adversary can only cheat B; second, it must be an interleave attack; last, the adversary can confuse N_A to cheat B. This constructed not just discovered attack is just the same as the well-known flaw discovered by Lowe using FDR[12].

$$
\begin{aligned}
&\text{Message1} \quad A \to I: \quad \{N_A, A\}_{K_I}\\
&\text{Message1}' \quad I(A) \to B: \quad \{N_A, A\}_{K_B}\\
&\text{Message2}' \quad B \to I(A): \quad \{N_A, N_B\}_{K_A}\\
&\text{Message2} \quad I \to A: \quad \{N_A, N_B\}_{K_A}\\
&\text{Message3} \quad A \to I: \quad \{N_B\}_{K_I}\\
&\text{Message3}' \quad I(A) \to B: \quad \{N_B\}_{K_B}
\end{aligned}
$$

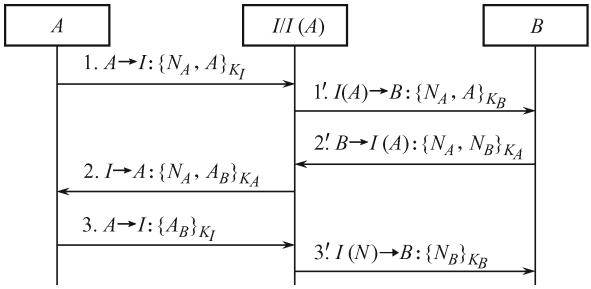


Fig. 7.2 Attack on the Needham-Schroeder public-key protocol.

The most important characteristic of the security analysis approach based on the trusted freshness is that the security analysis result suggests the struc-

ture to construct the attacks directly, and we may construct several attacks instantly from the security absence.

7.5.1.2 Analysis of the N-S-L public-key protocol

In [12], Lowe gave a correction of the original Needham-Schroeder public-key protocol, said N-S-L protocol, in Message 2:

$$B \to A: \quad \{B, N_A, N_B\}_{K_A}$$

Figure 7.2 illustrates the N-S-L protocol.

$$\begin{aligned} &\text{Message1} \quad A \to B: \quad \{N_A, A\}_{K_B} \\ &\text{Message2} \quad B \to A: \quad \{B, N_A, N_B\}_{K_A} \\ &\text{Message3} \quad A \to B: \quad \{N_B\}_{K_B} \end{aligned}$$

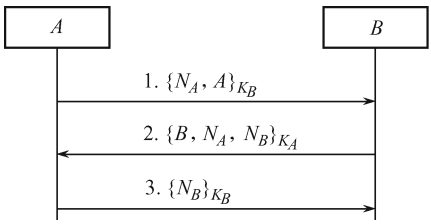


Fig. 7.3 N-S-L public-key protocol

Lowe proves the correctness of the revised protocol using FDR. A different correctness proof of N-S-L protocol with the belief multiset approach is given below. The similar analysis process in Subsection 7.5.1.1 is omitted.

7.5.1.3 Establish the security properties of N-S-L protocol

Up to now, both principals A and B have achieved the security goals of the protocol, hence N-S-L protocol is MK-secure, that is, N-S-L protocol is secure for building an secure channel between A and B in an insecure network. Table 7.3 shows the analyzing result of the N-S-L public-key protocol based on belief multisets.

Step 2. Upon receiving Message 2 $B \to A: \{B, N_A, N_B\}_{K_A}$

No.	Security properties	Term	Applied
(29)	$+\{...\ B, N_B...\}_{K_A} \wedge B_{B,t_0}(\langle 11K_A^{-1}A\rangle)$ $\wedge B_{B,t_2}(\langle 11N_B...\rangle) \Rightarrow B_{B,t_2}(\prec \{N_B, B, A\})$	$+\{...B, N_B...\}_{K_A}$	Premise, (10), (11),A2(c)
(30)	$+\{...B, N_B...\}_{K_A} \wedge B_{B,t_0}(\langle 01K_A\rho\rangle)$ $\wedge B_{B,t_0}(\langle 11K_A^{-1}A\rangle) \Rightarrow B_{B,t_2}(\langle ...N_B B\rangle)$	$+\{...B, N_B...\}_{K_A}$	Premise, A6(f)
(31)	$+\{...\ N_A, N_B...\ \}_{K_A} \wedge B_{B,t_0}(\langle 01K_A\rho\rangle)$ $\wedge B_{B,t_0}(\langle 11K_A^{-1}A\rangle) \wedge B_{B,t_2}(\langle ...1N_B B...\rangle)$ $\Rightarrow B_{B,t_2}(\sim \{...N_A, N_B...\}_{K_A})$	$+\{...\ N_A, N_B...\ \}_{K_A}$	Premise,(10), (30), A1(g)

Step 3. Upon receiving Message 3 $B \rightarrow A : \{N_B\}_{K_B}$

No.	Security properties	Term	Applied
(32)	$-\{...N_B...\}_{K_B} \wedge B_{B,t_2}(\prec \{N_B, B, A\})$ $\wedge B_{B,t_0}(Key(B, K_B^{-1})) \wedge B_{B,t_2}(\langle ...1N_B... \rangle)$ $\Rightarrow B_{B,t_3}(\langle ...1N_B A \rangle)$	$-\{...N_B...\}_{K_B}$	(10),(23),(29), A6(g)
(33)	$B_{B,t_3}(\langle 11N_B AB \rangle)$	(11),(26),(32)	
(34)	$B_{B,t_2}(\sim \{...N_A, N_B...\}_{K_A})$ $\wedge B_{B,t_3}(\langle ...1N_B AB \rangle) \Rightarrow B_{B,t_3}(\langle ...1N_A AB \rangle)$	(31),(33),A6(h)	
(35)	$B_{B,t_3}(\langle 11N_A AB \rangle)$	(9),(34)	
(36)	$b_{B,t_\$} = \lfloor \langle 11N_A AB \rangle, \langle 11N_B AB \rangle, \langle 1A \rangle \rfloor$	(24),(33),(35)	

Table 7.3 Analysis of the N-S-L public-key protocol based on belief multisets

Step	A's beliefs			B's beliefs		
Message 1		$\langle 11N_A A \rangle$			$\langle 1...N_A... \rangle$	
Message 2	$\langle 1B \rangle$	$\langle 11N_A AB \rangle$	$\langle 11N_B AB \rangle$			$\langle 11N_B B... \rangle$
Message 3				$\langle 1A \rangle$	$\langle 11N_A AB \rangle$	$\langle 11N_B AB \rangle$
At the End	$\langle 1B \rangle$	$\langle 11N_A AB \rangle$	$\langle 11N_B AB \rangle$	$\langle 1A \rangle$	$\langle 11N_A AB \rangle$	$\langle 11N_B AB \rangle$

7.5.2 Analysis of Kerberos pair-key agreement in DSNs

The Kerberos pair-key agreement approach in *Distributed Sensor Networks* (DSNs) uses a *Key Distribution Center* (KDC) to establish a secret shared key k_{AB} between two sensor nodes A and B. The four-pass protocol intends to provide mutual entity authentication, key confirmation and a key freshness guarantee between the sensor nodes. ID_A,ID_B and ID_I are the unique identifiers of node A, B and the adversary I respectively. N_A is a nonce invented by A. Each sensor node shares a secret key K_{A,KDC_j}, K_{B,KDC_j} with KDC_j (the jth KDC) respectively. T_A is a timestamp taken by node A. Define $ticket_B$ as $\{k_{AB}, A, L\}_{K_{B,KDC_j}}$, and L is the life of the key k_{AB}.

Figure 7.4 illustrates the Kerberos version 5 protocol in DSNs with some fields removed for clarity.

1) $A \rightarrow KDC_j : \ KDC_j, ID_A, ID_B, N_A$
2) $KDC_j \rightarrow A : \ ID_A, KDC_j, ticket_B, N_A, \{k_{AB}, ID_B, N_A, L\}_{K_{A,KDC_j}}$
3) $A \rightarrow B : \quad ID_B, ID_A, ticket_B, \{ID_B, T_A\}_{k_{AB}}$
4) $B \rightarrow A : \quad ID_A, ID_B, \{T_A\}_{k_{AB}}$

Specifying the security goals

$b_{A,t_\$} = \lfloor \langle 11k_{AB}AB \rangle, \langle 1B \rangle \rfloor$ and $b_{B,t_\$} = \lfloor \langle 11k_{AB}AB \rangle, \langle 1A \rangle \rfloor$.

Specifying the premise before the start of the protocol

$b_{A,t_0} = \lfloor \langle 11K_{A,KDC_j}A \ KDC_j \rangle, \langle 1...k_{AB}... \rangle, \langle ...B \rangle \rfloor$ and

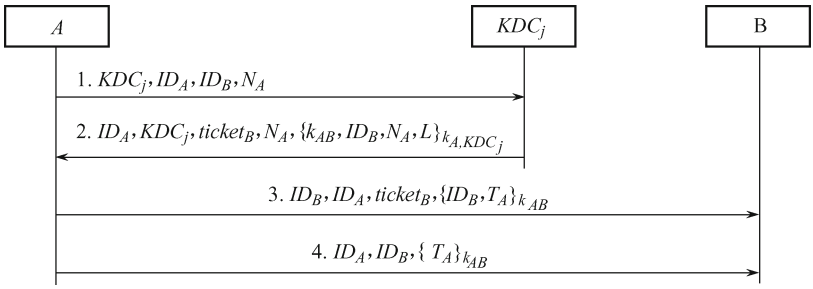


Fig. 7.4 Kerberos pair-key agreement approach in DSNs.

$b_{B,t_0} = \lfloor \langle 11 K_{B,KDC_j} B\ KDC_j \rangle, \langle 1 ... k_{AB} ... \rangle, \langle ... A \rangle \rfloor$.

Establishing the security properties

Step 1. Message 1 $A \rightarrow KDC_j : KDC_j, ID_A, ID_B, N_A$

No.	Security properties	Term	Applied
(1)	$+\{...N_A...\} \Rightarrow B_{A,t_1}(\langle ...1 N_A ... \rangle)$	$+\{...N_A...\}$	Rule A5(b)

Step 2. Message 2 $KDC_j \rightarrow A : ID_A, KDC_j, ticket_B, N_A, \{k_{AB}, ID_B, N_A, L\}_{K_{A,KDC_j}}$

No.	Security properties	Term	Applied
(2)	$B_{A,t_0}(\langle 11 K_{A,KDC_j} AKDC_j \rangle)$ $\Rightarrow B_{A,t_0}(\langle 11 K_{A,KDC_j} ... \rangle)$	Definition of $\langle ..._1 ..._2 N ..._3 \rangle$	
(3)	$B_{A,t_0}(\langle 11 K_{A,KDC_j} AKDC_j \rangle)$ $\Rightarrow B_{A,t_0}(\neg Key(I, K_{A,KDC_j}))$	Definition of $\langle ..._1 ..._2 N ..._3 \rangle$	
(4)	$-\{...k_{AB}...\}_{K_{A,KDC_j}}$ $\wedge B_{A,t_0}(\neg Key(I, K_{A,KDC_j}))$ $\Rightarrow B_{A,t_2}(\langle 1 ... k_{AB} ... \rangle)$	$-\{...k_{AB}...\}_{K_{A,KDC_j}}$	Premise, R4, (2),(3), A3(c)
(5)	$-\{...k_{AB}, N_A...\}_{K_{A,KDC_j}}$ $\wedge B_{A,t_0}(\langle 11 K_{A,KDC_j} AKDC_j \rangle)$ $\wedge B_{A,t_1}(\langle ...1 N_A ... \rangle)$ $\Rightarrow B_{A,t_2}(\sim \{...k_{AB}, N_A...\}_{K_A})$	$-\{...\ k_{AB}, N_A ...\ \}_{K_{A,KDC_j}}$	Premise, (1),A1(a)
(6)	$-\{...\ ID_B, N_A...\}_{K_{A,KDC_j}}$ $\wedge B_{A,t_0}(\langle 11 K_{A,KDC_j} AKDC_j \rangle)$ $\Rightarrow B_{A,t_2}(\langle ... N_A AB \rangle)$	$-\{...\ ID_B, N_A...\}_{K_{A,KDC_j}}$	Premise, (1),A6(b)
(7)	$B_{A,t_2}(\sim \{...k_{AB}, N_A...\}_{K_A})$ $\wedge B_{A,t_2}(\langle ...1 N_A AB \rangle)$ $\Rightarrow B_{A,t_2}(\langle ...1 k_{AB} AB \rangle)$	(1),(5),(6),A6(h)	
(8)	$B_{A,t_2}(\langle 11 k_{AB} AB \rangle)$	(4),(7)	

Step 3. Message 3 $A \rightarrow B : ID_B, ID_A, ticket_B, \{ID_B, T_A\}_{k_{AB}}$

No.	Security properties	Term	Applied
(9)	$+\{...T_A...\} \Rightarrow B_{A,t_3}(\langle ...1 T_A ... \rangle)$	$+\{...T_A...\}$	A5(a)
(10)	$-\{...T_A...\} \Rightarrow B_{A,t_3}(\langle ...1 T_A ... \rangle)$	$-\{...T_A...\}$	A5(a)

Step 4. Message 4 $B \rightarrow A : ID_A, ID_B, \{T_A\}_{k_{AB}}$

No.	Security properties	Term	Applied
(11)	$-\{...T_A...\}_{k_{AB}} \wedge B_{A,t_2}(\langle 11 k_{AB} AB\rangle)$ $\wedge B_{A,t_3}(\langle ...1T_A...\rangle) \Rightarrow B_{A,t_4}(\langle 1B\rangle)$	$-\{...T_A...\}_{k_{AB}}$	Premise, (8),(9),A4(a)
(12)	$b_{A,t_\$} = \lfloor \langle 11 k_{AB} AB\rangle, \langle 1B\rangle \rfloor$	(8),(11)	
(13)	$b_{B,t_\$} = \lfloor \langle .....k_{AB}...\rangle, \langle ...A\rangle \rfloor$		

Table 7.4 indicates the analyzing result of the Kerberos pair-key in DSNs based on belief multisets.

Table 7.4 Analysis of the Kerberos pair-key in DSNs based on belief multisets

Step	A's beliefs			B's beliefs		
Message 1		$\langle ...1N_A...\rangle$				
Message 2		$\langle 11 N_A AB\rangle$	$\langle 11 k_{AB} AB\rangle$			
Message 3		$\langle ...1T_A...\rangle$			$\langle ...1T_A...\rangle$	
Message 4	$\langle 1B\rangle$					
At the End	$\langle 1B\rangle$		$\langle 11 k_{AB} AB\rangle$	$\langle ...A\rangle$		$\langle .....k_{AB}...\rangle$

Comparing with the security goals

After the protocol execution, B cannot authenticate the liveness of A and the association of the session key k_{AB} with A and B. The absence of security properties suggests the structure of the attack directly. Suppose that an adversary has recorded $ticket'_B$ in Round 2 of a past normal run and the session key k'_{AB} is compromised before. Now the adversary I can launch an attack by impersonating A, and confusing B to regard an old key k'_{AB} (the absence of k_{AB}'s freshness) as a new session key k_{AB} between A and B (Similar to the attack discovered in [26] on Needham-Schroeder shared key protocol).

Figure 7.5 illustrates the attack on the Kerberos pair-key agreement approach in DSNs:

1) $I(A) \rightarrow KDC_j$: Skip
2) $KDC_j \rightarrow I(A)$: Skip
3) $I(A) \rightarrow B$: $ID_B, ID_A, ticket'_B, \{ID_B, T_I\}_{k'_{AB}}$
4) $B \rightarrow I(A)$: $ID_A, ID_B, \{T_I\}_{k'_{AB}}$

Now B thinks that B is communicating with A and sharing key k'_{AB} with A, while in fact A knows nothing about it.

Since an old $ticket_B$ can be replayed, a variation ($ticket_B \triangleq \{k_{AB}, N_A, L\}_{K_{B,KDC_j}}$) has been made to remedy the flaw. On the contrary, this variation just causes even worse flaw than the original protocol does, for the absence of association property of k_{AB} (with sensor nodes A and B) has not been solved yet. Suppose I is a legal participant, then I shares a secret key K_{IKDC_j} with KDC_j. N_I is a nonce invented by I, and T_I is a timestamp taken by node I.

Here is new attack on a revised Kerberos pair-key agreement approach in

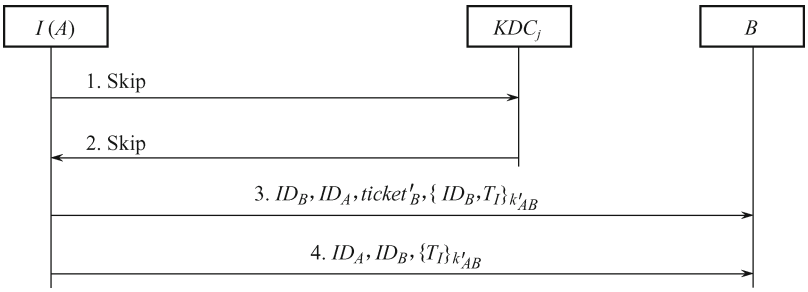


Fig. 7.5 An attack on the Kerberos pair-key agreement approach in DSNs.

DSNs as shown in Fig. 7.6.

1) $I \rightarrow KDC_j: \quad KDC_j, ID_I, ID_B, N_I$
2) $KDC_j \rightarrow I: \quad ID_I, KDC_j, ticket_B, N_I, \{k_{AB}, ID_B, N_I, L\}_{K_{IKDC_j}}$
3) $I(A) \rightarrow B: \quad ID_B, ID_A, ticket_B, \{ID_B, T_I\}_{k_{AB}}$
4) $B \rightarrow I(A): \quad ID_A, ID_B, \{T_I\}_{k_{AB}}$

This is a perfect attack. Upon receipt of the message in Round 3, B will check whether $ticket_B = \{k_{AB}, N_I, L\}_{K_{B,KDC_j}}$ and k_{AB} are correct or not. B could not find any abnormality, so B will continue the protocol run. Therefore, upon termination of this attack, B accepts the run with A even if A has not participated in the run with B at all. And B believes that B shares a new session key k_{AB} with A, but in deed B shares this key k_{AB} with the attacker I.

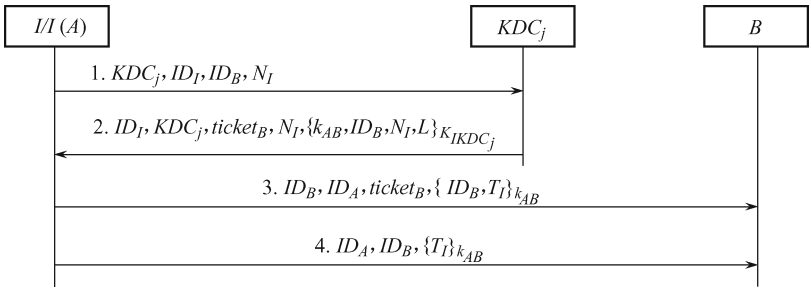


Fig. 7.6 Attack on a revised Kerberos pair-key agreement approach in DSNs.

7.5.3 Analysis of authentication in IEEE 802.11i

The future of networking lies within the wireless realm for its convenience and economy to use and for its ease to deploy. The IEEE 802.11 standard for Wireless Local Area Networks (WLAN) is one of the most widely adopted

standards for wireless Internet access[28]. But the open access to the radio interface in wireless networks exposes the content of communication to anyone who knows how to intercept radio waves at the proper frequencies.

The key problems of WLAN security are authentication and key management. In wireless networks, the key establishment protocol usually confirms the new session key while it is established, and it is very important whether a protocol could defense Denial of Service (DoS) attack.

7.5.3.1 Extension of the belief multiset formalism

Extended Notation

The belief multiset formalism is extended to meet these requirements as indicated in Table 7.5.

Table 7.5 Extended Statements

Fact	Description
$B_{P_i,\tau}(\langle P_i ..._1 ..._2 \rangle)$	beliefs owned by P_i about the DoS-tolerant property and the consistency property, the default is $B_{P_i,\tau}(\langle P_i ..... \rangle)$ or $B_{P_i,\tau}(\langle P_i ... \rangle)$. "$..._1$" states the DoS-tolerant property about the principal P_i itself. If P_i could defense DoS attack from the point of view of P_i, then P_i is DoS-tolerant and has $B_{P_i,t}(\langle P_i 1 ... \rangle)$; otherwise, P_i is not DoS-tolerant and has $B_{P_i,t}(\langle P_i 0 ... \rangle)$. Suppose each principal is DoS-tolerant at the beginning of the protocol analysis, that is $B_{\rho,t}(\langle \rho 1 ... \rangle)$. "$..._2$" states the consistency property about the protocol. If P_i believes that both sides have shown their possessions of the same session key, then P_i has $B_{P_i,t}(\langle P_i ... 1 \rangle)$; otherwise, P_i has $B_{P_i,t}(\langle P_i ... 0 \rangle)$.

Extended Rules

A7 (DoS-tolerant Rule)

From the point of view of a principal, if each principal of a protocol could defense DoS attack, then we say the cryptographic protocol could defense DoS attack, otherwise, the protocol is easy to be attacked by DoS.

Suppose each principal is DoS-tolerant at the beginning of the protocol analysis. If a received message is a plaintext or can be forged, then the receiver P_i could not defense the DoS attack.

A7(a) $-\{m\} \Rightarrow B_{P_i,t}(\langle P_i 0 ... \rangle)$

The DoS-tolerant rule A7(a) states: the principal P_i receives a plain text m, since even the adversary could construct the plaintext m, and P_i doesn't have the ability to distinguish whether m is from a legitimate participant or not, hence P_i could not defense the DoS attack.

A7(b) $-\{m\}_k \wedge B_{P_i,t}(Key(I,k)) \Rightarrow B_{P_i,t}(\langle P_i 0 ... \rangle)$

The DoS-tolerant rule A7(b) states: the principal P_i receives an encryption $\{m\}_k$ whose encryption key k is known by the attacker I. So I could construct the encryption $\{m\}_k$, and P_i doesn't have the ability to distinguish whether $\{m\}_k$ is from a legitimate participant or not, hence P_i could not defense the DoS attack.

A8 (Confirm Rule)

A8(a) $-\{...m...\}_k \wedge B_{P_i,t}(\langle 11kP_iP_j\rangle) \Rightarrow B_{P_i,t}(B_{P_j,t}(\langle 11kP_iP_j\rangle))$

The confirmed rule A8(a) states: if P_i believes that k is a new session key between P_i and P_j, and P_i receives an encryption of a plaintext m (or this message m could be recognized by the principal P_i) under the key k, then P_i is entitled to believe that the intended partner P_j knows the new session key k between P_i and P_j.

A8(b) $-\{...m...\}_k \wedge B_{P_i,t}(\prec \{k, P_i, P_j\}) \wedge B_{P_i,t}(\langle 11kP_iP_j\rangle)$
$\Rightarrow B_{P_i,t}(B_{P_j,t}(B_{P_i,t}(\langle 11kP_iP_j\rangle)))$

The confirmed rule A8(b) states: if P_i believes that k is a new session key between P_i and P_j, and only P_j can obtain the session key k for the protocol run related to P_i, when P_i receives a one-way transformation $\{...m...\}_k$ including a plaintext m (or this message m could be recognized by the principal P_i), then P_i believes that P_j knows that P_i has the same new session key k between principals P_i and P_j.

A9 (Consistency Rule):

$B_{P_i,t}(\langle 11kP_iP_j\rangle) \wedge B_{P_i,t}(B_{P_j,t}(\langle 11kP_iP_j\rangle)) \wedge B_{P_i,t}(B_{P_j,t}(B_{P_i,t}(\langle 11kP_iP_j\rangle)))$
$\Rightarrow B_{P_i,t}(\langle P_i...1\rangle)$

The consistency rule A9 states: if P_i believes that k is a new session key between P_i and P_j, and P_i believes that P_j knows that the key k is a new session key between P_i and P_j $(B_{P_i,t}(B_{P_j,t}(\langle 11kP_iP_j\rangle)))$, also P_i believes that P_j believes that P_i has the key k between P_i and P_j $(B_{P_i,t}(B_{P_j,t}(B_{P_i,t}(\langle 11kP_iP_j\rangle)))$, then the consistency of k on both sides is confirmed.

7.5.3.2 Analysis of the 4-way handshake in IEEE 802.11i

In this subsection, a formal security analysis of the enhanced authentication protocols in IEEE 802.11i including the 4-Way Handshake and the Group Key Handshake will be given. Analysis based on the belief multiset formalism is independent of the concrete formalization of attackers' possible behavior contrary to the analysis in [29, 30].

The first wireless security solution Wired Equivalent Privacy (WEP) protocol intends to provide confidentiality, access control and data integrity for IEEE 802.11-based networks[28]. However, the protocol has been proved to be vulnerable, including lack of legitimate key management and efficient authentication[31]. In order to enhance security of IEEE 802.11, a new standard called IEEE 802.11i is ratified in June 2004 to provide confidentiality, integrity, and mutual authentication[32].

IEEE 802.11i defines a Robust Security Network Association (RSNA) based on IEEE 802.1X authentication to enhance Medium Access Control (MAC) security[33,34]. RSNA includes a novel 4-Way Handshake to provide robust session key management.

There are three main entities in the IEEE 802.1X authentication system: supplicant (station, STA), authenticator (also called Access Point, AP), and Authentication Server (AS, Remote Authentication Dial In User Service

server). In IEEE 802.11, the AS might be physically integrated into an AP. Generally, a successful authentication means that STA and AP verify each other's liveness and generate a fresh session key for subsequent secure data transmissions.

A typical RSNA establishment procedure starts by executing an Extensible Authentication Protocol (EAP) authentication between the STA and the AS via the AP and STA's uncontrolled ports. If the STA and the AS authenticate each other successfully, they will both output the same Pairwise Master Key (PMK) independently. The AS then transmits the PMK to the AP through a secure channel (e.g., IPsec or TLS). The PMK may also be derived directly from a PreShared Key (PSK). With the PMK in place, the AP initiates a 4-Way Handshake with the STA, to confirm that the PMK at the peer is the same and current, and to derive and verify a Pairwise Transient Key (PTK) via PMK. Therefore, a secure communication channel between the authenticator AP and the supplicant STA can be constructed via the fresh key PTK.

The message exchanges of the 4-Way Handshake in IEEE 802.11i, as shown in Fig. 7.7.

$$
\begin{array}{lll}
\text{Message 1} & AP \rightarrow STA: & sn, ack, N_A \\
\text{Message 2} & STA \rightarrow AP: & sn, N_S, \{sn, N_S\}_{PTK} \\
\text{Message 3} & AP \rightarrow STA: & sn+1, ack, N_A, \{sn+1, ack, N_A\}_{PTK} \\
\text{Message 4} & STA \rightarrow AP: & sn+1, \{sn+1\}_{PTK}
\end{array}
$$

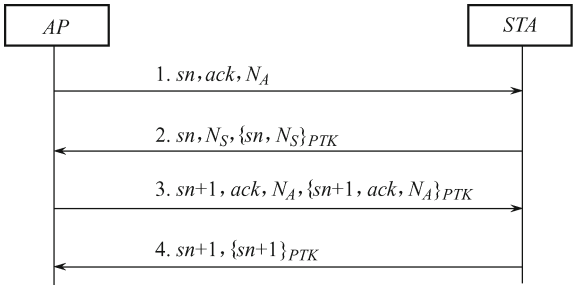


Fig. 7.7 The 4-Way Handshake in IEEE 802.11i.

We omit the lower level details of messages that do not contribute to the logical properties of authentication to be proved in belief multiset formalism. S and A represent STA and AP respectively; sn is a sequence number; ack means that a response to this message is required; N_A and N_S represent fresh nonce chosen by the subscript party A or S which are also called Anonce and Snonce; PTK (Pairwise Transient Key) is the new session key for this protocol; SPA and AA represent MAC address of STA and AP; the Min and Max operations for SPA and AA are with the address converted to a positive integer treating the first transmitted octet as the most significant octet of the

integer. $[m]_k$ is a message m encrypted under a key k with Message Integrity Code(MIC) protection.

Here, $PTK = PRF\text{ - }X$ (PMK, "Pairwise key expansion", Min(AA, SPA)||Max(AA, SPA)||Min(N_A, N_S)||Max(N_A, N_S)), and PTK should be a random number. $PRF\text{ - }X$ is a Pseudo Random Function with output length X.

The fresh PTK is split up into KCK (Key Confirmation Key), KEK (Key Encryption Key) and TK (Temporary Key). Note that MIC is actually calculated with KCK, which is only part of PTK. However, we do not distinguish KCK, KEK and TK from PTK here, for it doesn't effect the logical security properties of authentication to be proved.

The AP and STA normally derive a PTK only once per association. But the AP can refresh the PTK by running another 4-Way Handshake with the same PMK. The AP and STA silently discard any received message that has a used sequence number or an invalid MIC. The AP accepts only the expected reply within the configured time intervals, and the AP will deauthenticate the STA if the AP does not receive a valid response after several retries.

When IEEE 802.1X completes successfully, IEEE 802.11i assumes that the PMK is current and known only by STA and AP (AS does not expose the PMK or masquerade as STA or AP). Otherwise, IEEE 802.11i will fail to provide any security guarantees. If a PMK is not current, then it might be a compromised one and known by the attacker. If a PMK is known not only by STA and AP, then the attacker could compute PTK from PMK and the plaintext AA, SPA, N_A, N_S. Hence, the 4-Way Handshake relies heavily on the assumptions that PMK is current and known only by STA and AP.

Let's analyze the security of the 4-Way Handshake based on the above assumptions and the belief multiset formalism.

Specifying the security goals

The intended goal of the 4-Way Handshake is to establish secure communication between the authenticator AP (denoted A for short) and the supplicant STA (denoted S for short) via a new generated session key PTK. In the belief multiset formalism, the security goals of the 4-Way Handshake can be accurately expressed as

$$b_{A,t_\$} = \lfloor \langle 1S \rangle, \langle A11 \rangle, \langle 11PTK\ A\ S \rangle \rfloor,$$
$$b_{S,t_\$} = \lfloor \langle 1A \rangle, \langle S11 \rangle, \langle 11PTK\ A\ S \rangle \rfloor.$$

Specifying the premise sets

Recall the assumptions of PMK in the belief multiset formalism, and they can be stated as

$b_{A,t_0} = \lfloor \langle 11PMK\ A\ S \rangle, \langle ...S \rangle, \langle A1... \rangle, \langle 1...PTK... \rangle \rfloor,$
$b_{S,t_0} = \lfloor \langle 11PMK\ A\ S \rangle, \langle ...A \rangle, \langle S1... \rangle, \langle 1...PTK... \rangle \rfloor.$

Establishing the security properties

Establishing the security properties of the 4-Way Handshake based on the premise sets, the inference rules, and he fresh messages that include a trusted freshness identifier

Step 1. Message 1 $AP \rightarrow STA : sn, ack, N_A$

No.	Security properties	Term	Applied
(1)	$+\{...N_A...\} \Rightarrow B_{A,t_1}(\langle ...1N_A...\rangle)$	$+\{...N_A...\}$	A5(b)
(2)	$+\{...N_A...\} \Rightarrow B_{A,t_1}(\langle 0...N_A...\rangle)$	$+\{...N_A...\}$	A3(b)
(3)	$-\{...N_A...\} \Rightarrow B_{S,t_1}(\langle 0...N_A...\rangle)$	$-\{...N_A...\}$	A3(a)
(4)	$-\{sn, ack, N_A\} \Rightarrow B_{S,t_1}(\langle S0...\rangle)$	$-\{sn, ack, N_A\}$	A7(a)

Step 2. Message 2 $STA \rightarrow AP :$ $sn, N_S, \{sn, N_S\}_{PTK}$ [①]

No.	Security properties	Term	Applied
(5)	$+\{...N_S...\} \Rightarrow B_{S,t_2}(\langle ...1N_S...\rangle)$	$+\{...N_S...\}$	A5(b)
(6)	$+\{...N_S...\} \Rightarrow B_{S,t_2}(\langle 0...N_S...\rangle)$	$+\{...N_S...\}$	A3(b)
(7)	$+\{...PTK, N_S...\}_{PMK}$ $\wedge B_{S,t_0}(\langle 11\ PMK\ AS\rangle)$ $\wedge B_{S,t_2}(\langle ...1N_S...\rangle)$ $\Rightarrow B_{S,t_2}(\sim \{...PTK, N_S...\}_{PMK})$	$\{...PTK, N_S...\}_{PMK}$	Premise, (5), A1(a)
(8)	$B_{S,t_0}(\langle 11\ PMK\ AS\rangle)$ $\Rightarrow B_{S,t_0}(\neg Key(I, PMK))$	Definition of $\langle ..._1..._2N..._3\rangle$	
(9)	$B_{S,t_0}(\langle 11\ PMK\ AS\rangle)$ $\Rightarrow B_{S,t_0}(\langle 11\ PMK...\rangle)$	Definition of $\langle ..._1..._2N..._3\rangle$	
(10)	$+\{...PTK...\}_{PMK}$ $\wedge B_{S,t_0}(\neg Key(I,PMK))$ $\Rightarrow B_{S,t_2}(\langle 1...PTK...\rangle)$	$+\{...PTK...\}_{PMK}$	Premise,(8), (9), A3(d)
(11)	$+\{...N_S...\}_{PMK} \wedge B_{S,t_0}(\langle 11\ PMK\ AS\rangle)$ $\Rightarrow B_{S,t_2}(\langle ...N_S\ AS\rangle)$	$+\{...N_S...\}_{PMK}$	Premise, A6(a)
(12)	$B_{S,t_2}(\sim \{...PTK, N_S...\}_{PMK})$ $\wedge B_{S,t_2}(\langle ...1N_S\ AS\rangle)$ $\Rightarrow B_{S,t_2}(\langle ...1\ PTK\ AS\rangle)$		(5),(7), (11), A6(h)
(13)	$B_{S,t_2}(\langle 11\ PTK\ AS\rangle)$		(10),(12)
(14)	$+\{...sn...\}_{PTK} \wedge B_{S,t_2}(\langle 11\ PTK\ AS\rangle)$ $\wedge B_{S,t_2}(\langle 11\ PTK...\rangle)$ $\Rightarrow B_{S,t_2}(\prec \{PTK,S,A\})$	$+\{...sn...\}_{PTK}$	(13), A2(a)
(15)	$-\{...N_S...\} \Rightarrow B_{A,t_2}(\langle 0...N_S...\rangle)$	$-\{...N_S...\}$	A3(a)
(16)	$-\{...N_A...\}_{PMK} \wedge B_{A,t_0}(\langle 11\ PMK\ AS\rangle)$ $\Rightarrow B_{A,t_2}(\langle ...N_A\ AS\rangle)$	$-\{...N_A...\}_{PMK}$	Premise, A6(a)
(17)	$-\{...PTK, N_A...\}_{PMK}$ $\wedge B_{A,t_0}(\langle 11\ PMK\ AS\rangle)$ $\wedge B_{A,t_1}(\langle ...1N_A...\rangle)$ $\Rightarrow B_{A,t_2}(\sim \{...PTK, N_A...\}_{PMK})$	$-\{...\ PTK, N_A...\ \}_{PMK}$	Premise, (1), A1(a)

[①]PTK = PRF-X (PMK, "Pairwise key expansion", Min(AA, SPA)||Max(AA, SPA)||Min(N_A, N_S)||Max(N_A, N_S))

Continued

No.	Security properties	Term	Applied
(18)	$B_{A,t_2}(\sim\{...PTK, N_A...\}_{PMK})$ $\wedge B_{A,t_2}(\langle ...1N_A\ AS\rangle)$ $\Rightarrow B_{A,t_2}(\langle ...1\ PTK\ AS\rangle)$		(1), (16), (17), A6(h)
(19)	$B_{A,t_0}(\langle 11\ PMK AS\rangle)$ $\Rightarrow B_{A,t_0}(\neg Key(I, PMK))$	Definition of $\langle ..._1 ..._2 N ..._3\rangle$	
(20)	$-\{...PTK...\}_{PMK}$ $\wedge B_{A,t_0}(\neg Key(I,PMK)))$ $\Rightarrow B_{A,t_2}(\langle 1...PTK...\rangle)$	$-\{...PTK...\}_{PMK}$	(19), A3(c)
(21)	$B_{A,t_2}(\langle 11\ PTK\ AS\rangle)$		(18),(20)
(22)	$-\{...N_A...\}_{PMK} \wedge B_{A,t_0}(\langle 11\ PMK\ AS\rangle)$ $\wedge B_{A,t_1}(\langle ...1N_A...\rangle) \Rightarrow B_{A,t_2}(\langle 1S\rangle)$	$-\{...N_A...\}_{PMK}$	Premise, (1), A4(a),

Step 3. Message 3 $AP \rightarrow STA:\quad sn+1, ack, N_A, \{sn+1, ack, N_A\}_{PTK}$

No.	Security properties	Term	Applied
(23)	$-\{...N_S...\}_{PMK} \wedge B_{S,t_0}(\langle 11PMK\ AS\rangle)$ $\wedge B_{S,t_2}(\langle ...1N_S...\rangle) \Rightarrow B_{S,t_3}(\langle 1A\rangle)$	$-\{...N_S...\}_{PMK}$	Premise, (5), A4(a),
(24)	$-\{...sn+1...\}_{PTK} \wedge B_{S,t_2}(\langle 11\ PTK\ AS\rangle)$ $\Rightarrow B_{S,t_3}(B_{A,t}(\langle 11\ PTK\ AS\rangle))$	$-\{...sn+1...\}_{PTK}$	(13), A8(a)
(25)	$-\{...sn+1...\}_{PTK} \wedge B_{S,t_2}(\prec\{PTK,S,A\})$ $\wedge B_{S,t_2}(\langle 11PTK\ AS\rangle)$ $\Rightarrow B_{S,t_3}(B_{A,t}(B_{S,t}(\langle 11PTK\ AS\rangle)))$	$-\{...sn+1...\}_{PTK}$	Premise,(13), (14),A8(b)
(26)	$B_{S,t_2}(\langle 11PTK\ AS\rangle)$ $\wedge B_{S,t_3}(B_{A,t}(\langle 11PTK\ AS\rangle))$ $\wedge B_{S,t_3}(B_{A,t}(B_{S,t}(\langle 11PTK\ AS\rangle)))$ $\Rightarrow B_{S,t_3}(\langle S...1\rangle)$		(13),(24), (25), A9
(27)	$b_{S,t_3} = \lfloor\langle 1A\rangle, \langle S01\rangle, \langle 11\ PTK\ AS\rangle\rfloor$		(4),(13), (23),(26)
(28)	$+\{...sn...\}_{PTK} \wedge B_{A,t_2}(\langle 11\ PTK\ AS\rangle)$ $\wedge B_{A,t_2}(\langle 11PTK\ldots...\rangle)$ $\Rightarrow B_{A,t_3}(\prec\{PTK,A,S\}$	$+\{...sn...\}_{PTK}$	(21), A2(a)

Step 4. Message 4 $STA \rightarrow AP:\quad sn+1, \{sn+1\}_{PTK}$

No.	Security properties	Term	Applied
(29)	$-\{...sn+1...\}_{PTK} \wedge B_{A,t_2}(\langle 11PTK\ AS\rangle)$ $\Rightarrow B_{A,t_4}(B_{S,t}(\langle 11PTK\ AS\rangle))$	$-\{...sn+1...\}_{PTK}$	(21), A8
(30)	$-\{...sn+1...\}_{PTK} \wedge B_{A,t_3}(\prec\{PTK,A,S\}$ $\wedge B_{A,t_2}(\langle 11PTK\ AS\rangle)$ $\Rightarrow B_{A,t_4}(B_{S,t}(B_{A,t}(\langle 11PTK\ AS\rangle)))$	$-\{...sn+1...\}_{PTK}$	(21),(28), A8(b)
(31)	$B_{A,t_2}(\langle 11PTK\ AS\rangle)$ $\wedge B_{A,t_4}(B_{S,t}(\langle 11PTK\ AS\rangle))$ $B_{A,t_4}(B_{S,t}(B_{A,t}(\langle 11PTK\ AS\rangle)))$ $\Rightarrow B_{A,t_4}(\langle A...1\rangle)$		(21),(29), (30),A9
(32)	$b_{A,t_4} = \lfloor\langle 1S\rangle, \langle A11\rangle, \langle 11\ PTK\ AS\rangle\rfloor$		Premise,(21), (22),(31)

Table 7.6 shows the analyzing result of the 4-Way Handshake based on belief multisets.

Table 7.6 Analysis of the 4-Way Handshake based on belief multisets

Step	A's beliefs				S's beliefs			
Message 1			$\langle 01N_A ... \rangle$			$\langle S0 ... \rangle$	$\langle 0 ... N_A ... \rangle$	
Message 2	$\langle 1S \rangle$		$\langle 01N_A AS ... \rangle$	$\langle 11PTK\ AS \rangle$			$\langle 01N_S AS \rangle$	$\langle 11PTK\ AS \rangle$
Message 3					$\langle 1A \rangle$	$\langle S01 \rangle$		
Message 4		$\langle A11 \rangle$						
At the End	$\langle 1S \rangle$	$\langle A11 \rangle$		$\langle 11PTK\ AS \rangle$	$\langle 1A \rangle$	$\langle S01 \rangle$		$\langle 11PTK\ AS \rangle$

Comparing with the security goals established in step 1

Upon the termination of the protocol run, A has the belief with $b_{A,t_4} = \lfloor \langle 1S \rangle, \langle A11 \rangle, \langle 11PTK\ A\ S \rangle \rfloor$, so A has achieved all the security goals. That is, A believes that the PTK is special for this *Pairwise Transient Key Security Association* (PTKSA), so a secure communication channel between the authenticator $A(AP)$ and the supplicant $S(STA)$ can be constructed for subsequent data transmissions, based on the new session key PTK. The authenticator A also believes that A and S both have the same session key, and A can defense DoS attacks. Likewise, S has achieved the same security goals, as A has done, except that S may suffer from DoS attacks. That is, an attacker can launch DoS attacks upon S by impersonating A in a 4-Way Handshake.

From the analysis process, we can see that all the security properties are deduced from the belief $\langle 11PMK\ A\ S \rangle$ possessed by both A and S. That is to say, if the assumptions, that PMK is confidential and is only known by A and S, are broken, the 4-Way Handshake will fail to provide any security guarantees.

7.5.3.3 Analysis of the group key handshake in IEEE 802.11i

After the 4-way handshake protocol run, AP could make a Group Key Handshake protocol run to refresh a Group Temporal Key (GTK) to the STA by which multicast messages can be securely exchanged.

The Group Key Handshake is used only to issue a new GTK to peers with whom the local station (STA) has already formed security associations. The message exchanges of the Group Key Handshake protocol is shown in Fig. 7.8.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & sn+2, ack, \{sn+2, GTK\}_{PTK} \\ \text{Message 2} & S \rightarrow A: & sn+2, \{sn+2\}_{PTK} \end{array}$$

Specifying the security goals

The security goals of the Group Key Handshake are:

$$b_{A,t_\$} = \lfloor \langle 1S \rangle, \langle A1... \rangle, \langle 11\,GTK\ A\ S \rangle \rfloor \text{ and}$$
$$b_{S,t_\$} = \lfloor \langle 1A \rangle, \langle S1... \rangle, \langle 11\,GTK\ A\ S \rangle \rfloor.$$

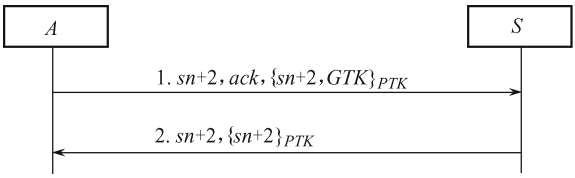


Fig. 7.8 Group Key Handshake in IEEE 802.11i.

Specifying the premise sets

The authenticator A couldn't initiate the Group Key Handshake until the 4-Way Handshake completes successfully, so the premise sets of the 4-Way Handshake protocol are

$$b_{A,t_0} = \lfloor \langle 11PTK\ A\ S\rangle, \langle ...S\rangle, \langle A1...\rangle \rfloor,$$
$$b_{S,t_0} = \lfloor \langle 11PTK\ A\ S\rangle, \langle ...A\rangle, \langle S1...\rangle \rfloor.$$

Recall that the authenticator A couldn't initiate the Group Key Handshake until the 4-Way Handshake completes successfully, so we can suppose that PTK is a fresh key, that is, the fresh property in the expression $B_{A,t_0}(\langle 11PTK\ A\ S\rangle)$ means that the key is fresh, and it is not a long-term property. Hence we have

$$B_{A,t_0}(\langle 11PTK\ A\ S\rangle) \Rightarrow B_{A,t_0}(\langle ...1PTK...\rangle).$$

Establishing the security properties of the Group Key Handshake

Message 1 $A \rightarrow S : sn+2, ack, \{sn+2, GTK\}_{PTK}$

No.	Security properties	Term	Applied
(1)	$+\{...GTK...\} \Rightarrow B_{A,t_1}(\langle ...1GTK...\rangle)$	$+\{...GTK...\}$	A5(b)
(2)	$B_{A,t_0}(\langle 11\ PTK\ A\ S\rangle) \Rightarrow B_{A,t_0}(\neg Key(I, PTK))$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(3)	$B_{A,t_0}(\langle 11\ PTK\ A\ S\rangle) \Rightarrow B_{A,t_0}(\langle ...1\ PTK...\rangle)$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(4)	$+\{...GTK...\}_{PTK} \wedge B_{A,t_0}(\neg Key(I, PTK))$ $\Rightarrow B_{A,t_1}(\langle 1...GTK...\rangle)$	$+\{...GTK...\}_{PTK}$	(2),A3(d)
(5)	$+\{...GTK...\}_{PTK} \wedge B_{A,t_0}(\langle 11\ PTK\ A\ S\rangle)$ $\Rightarrow B_{A,t_1}(\langle ...GTK\ A\ S\rangle)$	$+\{...GTK...\}_{PTK}$	Premise, A6(a)
(6)	$B_{A,t_1}(\langle 11\ GTK\ A\ S\rangle)$		(1),(4),(5)
(7)	$B_{S,t_0}(\langle 11\ PTK\ A\ S\rangle) \Rightarrow B_{S,t_0}(\neg Key(I, PTK))$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(8)	$B_{S,t_0}(\langle 11\ PTK\ A\ S\rangle) \Rightarrow B_{S,t_0}(\langle ...1\ PTK...\rangle)$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(9)	$-\{...GTK...\}_{PTK} \wedge B_{S,t_0}(\neg Key(I, PTK))$ $\Rightarrow B_{S,t_1}(\langle 1...GTK...\rangle)$	$-\{...GTK...\}_{PTK}$	(7),A3(c)
(10)	$-\{...GTK...\}_{PTK} \wedge B_{S,t_0}(\langle 11\ PTK\ A\ S\rangle)$ $\wedge B_{S,t_0}(\langle ...1\ PTK...\rangle)$ $\Rightarrow B_{S,t_1}(\sim \{...GTK...\}_{PTK})$	$-\{...\ GTK...\ \}_{PTK}$	Premise, (8),A1(a)
(11)	$B_{S,t_0}(\langle 11\ PTK\ A\ S\rangle) \Rightarrow B_{S,t_0}(\langle ...1\ PTK\ A\ S\rangle)$	Definition of $\langle ...{}_1...{}_2N...{}_3\rangle$	
(12)	$B_{S,t_1}(\sim \{...GTK...\}_{PTK})$ $\wedge B_{S,t_0}(\langle ...1\ PTK\ A\ S\rangle)$ $\Rightarrow B_{S,t_1}(\langle ...1\ GTK\ A\ S\rangle)$	(10),(11),A6(h)	
(13)	$B_{S,t_1}(\langle 11\ GTK\ A\ S\rangle)$		(9),(12)

Continued

No.	Security properties	Term	Applied
(14)	$-\{...GTK...\}_{PTK} \wedge B_{S,t_0}(\langle 11\ PTK\ A\ S\rangle)$ $\wedge B_{S,t_0}(\langle ...1\ PTK...\rangle) \Rightarrow B_{S,t_1}(\langle 1A\rangle)$	$-\{...GTK...\}_{PTK}$	Premise, (8), A4(a)
(15)	$b_{S,t_1}\lfloor\langle 1A\rangle, \langle 11\ GTK\ A\ S\rangle\rfloor$		(13),(14)

Message 2 $S \rightarrow A : sn + 2, \{sn + 2\}_{PTK}$

No.	Security properties	Term	Applied
(16)	$-\{...GTK...\}_{PTK} \wedge B_{A,t_0}(\langle 11\ PTK\ A\ S\rangle)$ $\wedge B_{A,t_0}(\langle ...1\ PTK...\rangle) \Rightarrow B_{A,t_2}(\langle 1S\rangle)$	$-\{...GTK...\}_{PTK}$	Premise, (3), A4(a)
(17)	$b_{A,t_2} = \lfloor\langle 1S\rangle, \langle A1...\rangle, \langle 11\ GTK\ A\ S\rangle\rfloor$		Premise, (6),(16)
(18)	$b_{S,t_2} = \lfloor\langle 1A\rangle, \langle S1...\rangle, \langle 11\ GTK\ A\ S\rangle\rfloor$		Premise, (15)

At last, S has $b_{S,t_\$} = \lfloor\langle 1A\rangle, \langle S1...\rangle, \langle 11GTK\ A\ S\rangle\rfloor$ and A has $b_{A,t_\$} = \lfloor\langle 1S\rangle, \langle A1...\rangle, \langle 11GTK\ A\ S\rangle\rfloor$. So, the Group Key Handshake has achieved its security goals to transfer GTK from the AP to the STA, and GTK will provide secure channel for multicast message transmissions if the assumptions (PTK is confidential, new and only known by A and S) are in place.

7.5.3.4 Discussion about authentication in IEEE 802.11i

The above security analysis shows that the 4-Way Handshake and the Group Key Handshake may provide satisfactory mutual authentication and key management under the assumptions that the particular Supplicant-Authenticator pair is authorized to know the PMK and to use it in the 4-Way Handshake. But if the assumptions are broken, then the 4-Way Handshake couldn't guarantee the mutual authentication, the confidentiality and association of PTK.

1. *PMK security*

Recall that a PMK may be derived from an EAP method or may be obtained directly from a preshared key PSK.

Although there is only one PTKSA with a Supplicant-Authenticator pair (STA-AP pair), there may be more than one Pairwise Master Key Security Association (PMKSA) with the same pair. Besides this, the PTKSA consists of MAC address AA and SPA, while the PMKSA consists of only AA.

If an STA wishes to roam to an AP for which it has cached one or more PMKSAs, it can include a PMKID in its (Re)Association Request frame. An AP whose authenticator has retained the PMK for one or more of the PMKIDs may skip IEEE 802.1X authentication and proceed with the 4-Way Handshake. If PMK is implicitly associated with an attacker and the supplicant (or the authenticator), a PTK for a spurious session between the supplicant and the authenticator could also be deduced by the attacker and the victim via PMK, the plaintext N_A, N_S, AA and SPA. Hence, we can only guarantee that IEEE 802.11i could potentially improve security services in

IEEE 802.11 WLAN.

2. DoS attack

The above analysis also shows that there exist DoS attacks upon the supplicant. Message 1 does not carry an MIC, as it is impossible for the supplicant to distinguish this message from a replay message. The supplicant must accept all Message 1's received in case of packet loss and retransmission, which makes the Message 1 attack unavoidable. Although the forgery of Message 1 may be detected in the failure of Message 3, the forged Message1s still cause the supplicant to be blocked for the inconsistency *Anonce* in Message 1 and Message 3. To improve this, the supplicant side may store every received *ANonce*, the responding *SNonce* and the derived *PTK*, but this may cause memory exhaustion for a WLAN device whose energy supply is limited. The supplicant may reuse the value of *SNonce* for the same PTKSA to eliminate the memory DoS attack at the cost of recomputing the final *PTK* upon receiving Message 3, so the supplicant side only needs to store one *SNonce* for this PTKSA[35,36]. In some cases, one entry of the derived *PTK* and the received *ANonce* can be stored to improve CPU consumption by verifying MIC in Message 3 directly.

3. Redudancy

It seems that the key replay counter is a redundancy in the 4-Way Handshake since replay protection has already been provided implicitly by *PTK* including *ANonce* and *SNonce*. However, the key replay counter is not redundant and it plays a useful role as a minor performance optimization in processing stale instances of messages. It is especially useful for these devices with limited computational power.

Ack bit helps to stop reflection attacks. Message 4 serves no cryptographic purpose, but it can ensure reliability. Message 4 is required to inform the authenticator that the supplicant knows that the authenticator has already installed *PTK*. Hence, 4-Way Handshake is a protocol with consistency which is an important virtue in open WLAN[31,36].

7.6 Comparison

A comparison of belief multiset formalism with other previously known formalisms is indicated in Table 7.7.

The central ingredient in the security analysis based on trusted freshness is the freshness principle: For each participant of a cryptographic protocol, the security of the protocol depends only on the sent or received "loose" one-way transformation of a message which includes a trusted freshness. The belief multiset formalism is established on the basis of the freshness principle, and the idea to analyze the security of cryptographic protocols based on trusted freshness is the fundamental difference from previous formalisms including

BAN logic.

Table 7.7 Comparison of Belief Multisets with Previous Formalisms

Formalism	Correctness Proving	Finding Flaws	Instant Construction Attacks	Independent of the Formalization of Attacker's Abilities	Independent of the Concurrent Runs of Protocols	Automation
BAN Logic[4]	✓			✓		✓
Formalism Based on Model Checking[5]		✓	✓			✓
Strand Space[6]	✓					✓
Authentication Test[37]	✓					✓
Random Oracle[38]	✓			✓	✓	
CK Model[39]	✓					
Belief Multisets[22]	✓	✓	✓	✓	✓	✓

A rigorous proof has been made to show whether a cryptographic protocol is secure under computational model, which implies that the security in the belief multisets will deduce the security in the computational model. Meanwhile, as we have seen, the security properties of cryptographic protocols can be expressed easily in belief multiset logic.

An important advantage of the belief multiset logic is that the security analysis based on trusted freshness can efficiently clarify whether a message is fresh or not based on already trusted freshness identifier. A freshness identifier generated by a principal itself is a trusted freshness, and other freshness identifiers involved in the protocol run are not inherently fresh. The trusted freshness of a freshness identifier can be attained by a one-way transformation of a message, which includes certain trusted freshness. All the beliefs attained in the reasoning process are established on the basis of trusted freshness.

Security analysis based on trusted freshness captures the exact security properties of a cryptographic protocol, which can establish the correctness of those protocols convincingly when they are in fact correct, or identify the absence of security properties definitely in those that are not correct. On the contrary, some formalisms like theorem proving, random oracle are correctness proving methods although a failure of a desired property by such a formalism may result in some insightful ideas to revelation of a hidden error[1–3].

The absence of security properties which is indicated in belief multisets suggests the way to construct attacks intuitively as we have demonstrated in. Chapter 4, Chapter 5 and also Chapter 7. Of course, a formalism based on

Model Checking may definitely indicate the path to reach an insecure state, but this attack is discovered by a global state space exploration and not by construction. So far as we know, security analysis based on trusted freshness is one of the most convenient and efficient approaches that help researchers to construct attacks directly.

Security analysis based on trusted freshness is independent of the idealization of a protocol, the concrete formalization of an attacker's possible behavior, and it is independent of the concurrent runs with any set of protocols. On the contrary, the analysis based on Model checking, strand space, CK model depends directly on the idealization of a protocol, the concrete formalization of an attacker's possible behavior, while the attacker's abilities are always developing, and a flaw is usually found by a new assumption of the attacker's abilities.

The proofs of security via belief multisets are simple and precise, which can be easily accomplished even by hand (Chapter 4 and Chapter 5), while at the same time the process is completely rigorous and amenable for automation (Chapter 7).

We must emphasize that the security analysis based on trusted freshness is foundationally different from BAN logic, although belief multiset formalism exemplified in this chapter is also a logic tool: the belief in the approach depends only on the sent or received one-way transformation that includes trusted freshness identifier while BAN logic deals with all messages. We distinguish between trusted fresheness and freshness, and all the beliefs attained in the reasoning process are established on the basis of trusted fresheness. This is the fundamental difference between the belief mulitset formalism and the BAN-like logic. The premise of the initial assumptions is simple and clear in the belief mulitset formalism (refer to Subsection 7.5); the idea behind operation rules and also inference rules is completely different from BAN logic, so the security properties achieved by a protocol based on these two logic methods are completely different. Moreover, BAN logic has problems in providing a correctness proving for its questionable idealization of a protocol, hence BAN logic fails to find the interleaving attacks on the Needham-Schroeder public-key protocol, although BAN logic is a simple and especially useful method[1].

The approach belief multiset formalism seems to offer important advantages. First, the approach can establish the correctness of a cryptographic protocol when it is in fact correct, or identify the absence of security properties definitely in that which is not correct, for the guarantee of security described in Chapter 6 is not only sufficient but also necessary. The CK model can establish the correctness of a cryptographic protocol via an authenticator while BAN logic can only be used as a finding fault tool for its informal idealization of the protocol and assumptions. Secondly, from the absence of security properties compared with the sufficient and necessary conditions, the belief multisets can not only discover attacks but also construct various attacks directly as we have illustrated. Finally, the initial assumptions re-

quired are explicit and the reasoning process based on the belief multisets is simple and perfectly performed. The only universal assumption required in the reasoning process is that each principal knows his own private key and the public-keys of other parties (including the adversary) in public-key case, or knows the shared long-term key with its partner or the trusted third party in shared key case.

References

[1] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[2] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[3] Goldreich O (2003) Foundations of Cryptography. Cambridge University Press, New York

[4] Burrows M, Abadi M, Needham R (1990) A Logic of Authentication. ACM Transactions on Computer Systems 8(1): 18 – 36

[5] Lowe G (1999) Towards a Completeness Result for Model Checking of Security Protocols. Journal of Computer Security 7(2 – 3): 89 – 146

[6] Fabrega FJT, Herzog JC, Guttman JD (1998) Strand Spaces: Why is a Security Protocol Correct? In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, 3 – 6 May 1998

[7] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[8] Zhang YQ (2000) Study on Analysis of Security Protocols of Computer Communication Network. PhD Dissertation (in Chinese), XIDIAN University

[9] Gong L, Needham R, Yahalom R (1990) Reasoning About Belief in Cryptographic Protocols. In Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, Oakland, 7 – 9 May 1990

[10] Abadi M, Tuttle MR (1991) A Semantics for a Logic of Authentication. In: Proceedings of the 10th ACM Symposium on Principles of Distributed Computing, Montreal, 19 – 21 Aug 1991

[11] Syverson PF, Oorschot PCV (1994) On Unifying Some Cryptographic Protocol Logics. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Oakland, 16 – 18 May 1994

[12] Lowe G (1996) Breaking and Fixing the Needham-Schroeder Public-key Protocol Using FDR. In: TACAS'96 Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Passau, 27 – 29 Mar 1996. Lecture Notes in Computer Science (Lecture Notes in Software Configuration Management), vol 1055. Springer, Heidelberg, pp 147 – 166

[13] Millen JK, Clark SC, Freedman SB (1987) The Interrogator: Protocol Security Analysis. IEEE Trans. Software Eng. 13(2): 274 – 288

[14] Mitchell JC, Mitchell M, Stern U (1997) Automated Analysis of Cryptographic Protocols Using MurΦ. In: Proceedings of 1997 IEEE Symposium on Security and Privacy, Oakland, 4 – 7 May 1997

[15] Meadows C (1994) A Model of Computation for the NRL Protocol Analyzer. In: Proceedings of the 1994 Computer Security Foundations Workshop, Franconia, 14 – 16 June 1994

[16] Meadows C (1996) The NRL Protocol Analyzer: an Overview. Journal of Logic Programming 26(2): 113 – 131

[17] Meadows C (1999) Analysis of the Internet key Exchange Protocol Using the NRL Protocol Analyzer. In: Proceedings of 1999 IEEE Symposium on Security and Privacy, Oakland, 9 – 12 May 1999

[18] Fabrega FJT, Herzog JC, Guttman JD (1999) Mixed Strand Spaces. In: Proceedings of the 12th IEEE Computer Security Foundations Workshop, Mordano, 28 – 30 June 1999

[19] Song D, Berezin S, Perrig A (2001) Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. Journal of Computer Security 9(1 – 2): 47 – 74

[20] Dong L (2008) Cryptographic Protocol Engineering and Protocol Security Based on Trusted Freshness. PhD Dissertation (in Chinese), Shanghai Jiaotong University

[21] Chen K, Dong L, Lai X (2008) Security Analysis of Cryptographic Protocols Based on Trusted Freshness. Journal of Korea Institute of Information Security and Cryptology, 18(6B): 1 – 13

[22] Dong L, Chen K, Lai X (2009) Belief Multisets for Cryptographic Protocol Analysis. Journal of Software 20(11): 3060 – 3076 (in Chinese)

[23] Dong L, Chen K, Lai X, Wen M (2009) When is a Key Establishment Protocol Correct? Security and Communication Networks, 2(6): 567 – 579

[24] Otway D, Rees O (1987) Efficient and Timely Mutual Authentication. Operating Systems Review 21(1): 8 – 10

[25] Abadi M, Needham R (1996) Prudent Engineering Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering 22(1): 6 – 15

[26] Denning DE, Sacco GM (1981) Timestamps in Key Distribution Protocols. Communication of the ACM 24(8): 533 – 536

[27] Lowe G (1995) An Attack on the Needham-Schroeder Public Key Authentication Protocol. Information Processing Letters 56(3): 131 – 133

[28] ANSI/IEEE Std 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Sept 1999

[29] Furqan Z, Muhammad S, Guha RK (2006) Formal Verification of 802.11i Using Strand Space Formalism. In: IEEE Proceedings of ICNICONSMCL'2006, Morne, 23 – 29 Apr 2006. IEEE Press, pp 140 – 140

[30] Sithirasenan E, Zafar S, Muthukkumarasamy V (2006) Formal Verification of the IEEE 802.11i WLAN Security Protocol. In: IEEE Proceedings of ASWEC'2006, Sydney, 18 – 21 Apr 2006. IEEE Press, pp 181 – 190

[31] Brown B (2003) 802.11: The Security Differences Between b and i. IEEE Potentials 22(4): 23 – 27

[32] IEEE Std 802.11i-2004. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Security Enhancements. July 2004

[33] IEEE Std 802.1X. Port-based Network Access Control. NewYork: IEEE Press, 2001

[34] IEEE Std EAP-2004. Extensible Authentication Protocol (EAP). New York: IEEE Press, June 2004

[35] He C, Mitchell JC (2004) Analysis of the 802.11i 4-Way Handshake. In: Proceedings of the 3rd ACM Workshop on Wireless security (Wise'04), Philadelphia, 1 Oct 2004. pp 43 – 50

[36] Chen JC, Jiang MC, Liu YW (2005) Wireless Lan Security and IEEE 802.11i. IEEE Wireless Communications 12(1): 27 – 36

[37] Guttman JD, Thayer F (2000) Authentication Tests. In: Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, 14 – 17 May 2000

[38] Bellare M, Rogaway P (1993) Entity Authentication and Key Distribution. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 773. Springer-Verlag, pp 232 – 249

[39] Canetti R, Krawczy H (2001) Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In: EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045. Springer-Verlag, pp 453 – 474

8 Design of Cryptographic Protocols Based on Trusted Freshness

Abstract Informal design principle research and formal design method research are the two main parts of cryptographic protocol design research. We have presented ten cryptographic protocol engineering principles for protocol design in Chapter 4, and a belief multiset design model based on trusted freshness is put forward in this chapter. Moreover, the efficiency of the key establishment protocol is also discussed.

The research of the cryptographic protocol design includes two main parts: informal design principle research and formal design method research. In the former field, we have introduced the notion of cryptographic protocol engineering which is derived from the notion of software engineering, and then we have put forward ten cryptographic protocol principles for protocol design, that is, the cryptographic protocol engineering principles. In this chapter, we will develop a belief multiset design model based on trusted freshness, with detailed specifications of security goals to meet the security property definitions under the Bellare-Rogaway's computational model, for designing cryptographic protocols.

The design of cryptographic protocol is much related to the analysis of cryptographic protocol. BAN logic[1] is one of the most advanced work in this area which presents a proof-based approach and focuses on partner's beliefs. Some others are model-based approaches which focus on partner interaction[2, 3]. The first complexity-theoretic treatment of the notion of security for key establishment protocols is given by Bellare and Rogaway[4]. A central aspect of this definition is the notion of key indistinguishability, which states that an adversary cannot distinguish between the real key and a nonce chosen at random. This model was refined and extended by Canetti and Krawczyk[5] via combination of previous definitional approaches[2, 4, 6]. Here, security of a real protocol is asserted by comparing it with an ideal protocol that is secure by construction. [5] proves an equivalence between single session UC-security of key exchange protocols and the indistinguishability-based notion introduced in [4]. But they offer a limited form of composition guarantees where primitives do not shared state or shared state under some very specific conditions[7, 8]. Moreover, these formalizations are not always

easy to work with for not providing detailed specifications for key establishment protocol design. For example, when will a key establishment protocol be secure? What security properties should the protocol have? How can the protocol achieve these security properties? and so on.

The belief multiset design model will answer these questions. The belief multiset design model is established based on the notation of the trusted freshness[9, 10]. Recall the important freshness principle presented in Chapter 4: for each participant of a cryptographic protocol, the security of the protocol depends only on the sent or received "loose" one-way transformation of a message which includes a trusted freshness. This methodology is also the central ingredient in our protocol design. Moreover, we will discuss the issue of protocol efficiency, where there are only scattered published discussions, on protocol design based on this belief multiset model.

8.1 Previously known methods for protocol design

We will very briefly summarize the ideas behind the previously known methods for protocol design. For details please refer to [7, 11 – 15].

8.1.1 A simple logic for authentication protocol design

The simple logic proposed by Buttyan, Staamann and Wilhelm in 1998 considers a distributed system to be a set of principals and channels[11]. The principals can interact with each other according to the rules of some predefined protocols in order to accomplish a common task. The channels are generalizations of communication links with various security properties. The channel can represent a physical link, as well as a cryptographically secured logical connection between principals. A channel is characterized by its set of readers and its set of writers (i.e., the set of principals that can receive messages via the channel and the set of principals that can send messages via the channel). There are six basic types of channels: Public Channel is a channel iff anybody in the system can write and read it; Authentic Channel is a channel iff anybody can read it but only one principal P can write it; Confidential Channel is a channel iff anybody can write it, but only one principal can read it; Dedicated Channel is a channel iff one principal can read it and another principal Q can write it; Closed Group Channel is a channel iff a set of principals can write it and the same set of principals can read it; Conventional Secret Channel is a channel iff it can be used only by two determined principals. In this simple logic, the authentication protocol is treated at a higher abstraction level without having to be concerned with the problems of the actual implementation such as dealing with encryption

or decryption operations.

The simple logic belongs to the BAN logic family. The simple logic gives several types of channels, a set of synthetic rules that can be used by protocol designers to establish channels, to construct a protocol in a systematic way. First, the designer must identify the goals of the protocol and describe them with the language of the simple logic. Then, by using the given synthetic rules, the designer can generate the whole protocol and the required assumptions in a systematic way. The result of this process is a formal description of the protocol in the simple logic language.

8.1.2 Fail-stop protocol design

A protocol is fail-stop if any attack interfering with a message sent in one step will cause all causally-after messages in the next step or later not to be sent. The fail-stop cryptographic protocol is introduced by Gong and Syverson in 1995[12]. As Gong and Syverson show, fail-stop protocols possess a very useful security property, namely: active attacks cannot cause the release of secrets within the run of a fail-stop protocol. The fail-stop property lets a protocol designer restrict his concerns to passive (eavesdropping) attacks with the cost that the protocol must terminate when active attacks occur, rather than attempt to continue. In fail-stop protocol design, it is believed that reliable termination is greatly preferred to unknown and insecure behavior in the face of active attacks on security.

In fail-stop protocol, a distributed system is considered as a collection of processes which are spatially separated and communicate with each other by exchanging messages. A protocol is a specification for the format and relative timing of the messages exchanged. A cryptographic protocol uses cryptographic mechanisms such as encryption and decryption algorithms to guarantee the integrity, the secrecy, the origin, the destination, the order, the timeliness, and ultimately the meaning of the messages. Using Lamport's definition of causality[13], the messages of a protocol are organized into an acyclic directed graph where each arc represents a message and each directed path represents a sequence of messages. In a fail-stop protocol, if a message actually sent is in any way inconsistent with the protocol specification, then all those messages that are causally after the altered message on some path in the graph will not be sent.

8.1.3 Authentication test

Authentication test is a heuristic method for finding attacks against incorrect protocols, which was proposed by Guttman and Thayer in 2000[16]. Authen-

tication test is also useful in protocol design. Suppose a principal in a cryptographic protocol creates and transmits a message containing a new value v, which it later receives back in cryptographically altered form. This kind of component may be regarded as an authentication test. Therefore, it can be concluded that some principal possessing the relevant key has transformed the message containing v. In some circumstances, this must be a regular participant of the protocol, not the attacker, who has transformed the message. The authentication tests give sufficient conditions for transforming edges being the work of regular principals.

There are two main kinds of authentication tests. A transforming edge is the action of changing the cryptographic form in which such a value v is sent out and later received in a new component, and the notion of a transforming edge, in which a value is received and later sent out in a new component. An outgoing test is the one in which the new value v is transmitted in encrypted form, and only a regular participant can extract v from that form. An incoming test is the one in which v is received back in encrypted form, and only a regular participant can put it in that form. An unsolicited test is a combination of these two tests with a supplementary idea and a related method for checking that certain values remain secret. Together, they determine what authentication properties are achieved by a wide range of cryptographic protocols. Authentication test is expressed in the strand space formalism.

The outgoing, incoming, and unsolicited tests, and the authentication results that apply to them suggest a protocol design process. At the level of abstraction in authentication test, authentication protocol design is largely a matter of selecting authentication tests, and constructing a unique regular transforming edge to satisfy. It is important to start by deciding the goals to be achieved. Then from the goal it follows that each regular participant of the protocol selects authentication tests and constructs the corresponding protocol messages. At last the protocol is formed by these constructed messages. We have noticed that the outgoing test, incoming test and unsolicited test in authentication test have considered the freshness property of new value v but have neglected the association property of v with determined regular participants of the protocol, hence the constructed protocol via authentication test may not defense the interleaving attack.

8.1.4 Canetti-Krawczyk model

Based on the modular approach [2], Canetti and Krawczyk present a formalism (Canetti-Krawczyk model, CK model for short) to construct a new provably secure key-exchange protocol: one can design and prove security of key-exchange protocols in an idealized model AM (authenticated-link adversarial model) where the communication links are perfectly authenticated,

and then translate them using general tools (authenticator) to obtain security in a realistic setting UM (unauthenticated-link adversarial model) of adversary-controlled links[5].

CK model combines previous definitional approaches and results in a definition of security that enjoys the important analytical benefit: any key-exchange protocol that satisfies the security definition can be composed with symmetric encryption and authentication functions to provide provably secure communication channels. AM model is authenticated-links model, which is defined in a way that the attacker is restricted to only deliver messages truly generated by the parties without any change or addition to them. That is, the attacker could launch a run of a protocol, impersonate a legitimate participant of a protocol, and discover the session key being used, but the attacker could not forge or replay the message of an uncorrupted party. UM model considers a probabilistic polynomial-time attacker that has full control of the communication links: it can listen to all the transmitted information, decide what messages will reach their destination and when to change these messages at will or inject its own generated messages. The attacker also controls the scheduling of all protocol events including the initiation of protocols and message delivery. Besides these abilities, the attacker could also obtain secret information stored in the parties' memories via explicit attacks, e.g., Session-state reveal, Session-key query, Party corruption. One important additional element in CK model is the notion of session expiration, and this takes the form of a protocol action that when activated causes the erasure of the named session key (and any related session state) from that party's memory. Authenticator is a protocol translation tool (or "compiler") which "automatically" transforms the constructed protocols in AM model into equivalent (or "as secure as") protocols in the realistic scenario of fully adversary-controlled communication UM model. This process is the notion of "emulation" which is introduced in order to capture the equivalence of functionality between protocols in different adversarial models, particularly between the UM model and AM model. However, CK model depends directly on the idealization of a protocol, the concrete formalization of an attacker's possible behavior, while the attacker's abilities are always developing, and a flaw is usually found by a new assumption of the attacker's abilities.

8.1.5 Models for secure protocol design and their compositions

In 1996, Heintze and Tygar proposed a model for secure protocols and their Compositions design[3]. Heintze-Tygar model splits the notion of security into two parts: the secret-security and the time-security of protocols. By secret-security, messages that are believed to be secret are never revealed. By time security, stale messages can not be replayed. Heintze-Tygar model uses a limited form of belief to capture part of the state of an agent. These beliefs

are either about the freshness of nonce, or about the security of messages (particularly, which secrets are shared with whom). Beliefs used to describe agent state in the Heintze-Tygar model bear little relationship to the actual protocol properties that can be established. Heintze-Tygar model proposes a very general treatment of time (particularly, there are no assumptions about global synchronization of time). Heintze-Tygar model also states sufficient conditions on two secure protocols guaranteeing the security of a new composite protocol.

In 2003, Datta et al. introduced a basic framework for deriving security protocols from an accepted set of standard concepts such as Diffie-Hellman key exchange, nonce to avoid replay, certificates from an accepted authority, and encrypted or signed messages[14]. As initial steps toward associating logical derivations with protocol derivations, the framework extends a previous security protocol logic with preconditions and temporal assertions, and derives a family of key exchange protocols, including Station-To-Station (STS), ISO-9798-3, Just Fast Keying (JFK), IKE and related protocol, from two basic protocols using a small set of refinements and protocol transformations.

In 2006, Datta et al. proposed a compositional method for proving cryptographically sound security properties of key exchange protocols, based on a symbolic logic that is interpreted over conventional runs of a protocol against a probabilistic polynomial-time attacker[7]. This logic is with axioms capturing properties of signatures, symmetric encryption, and message authentication codes, as well as with the Decisional Diffie-Hellman assumption. Since commonly used reasoning principles are codified in the proof system, protocol security proofs can be carried out at a high-level of abstraction without worrying about probability and complexity. Protocol proofs in this logic are compositional, that is, proofs of compound protocols can be constructed from proofs of their parts. The axioms used in a proof identify specific properties of cryptographic primitives that are sufficient to guarantee the desired protocol properties. The proof system is used to establish security of a standard protocol in the computational model. However, the procedure of protocol idealization can be an error-prone process, for instance, formulae, Fresh and Honest about the messages transmitted between principals.

In 2007, Datta et al. further proposed the Protocol Composition Logic (PCL) which is designed around a process calculus with actions for possible protocol steps including generating new random numbers, sending and receiving messages, and performing decryption and digital signature verification actions[17]. The proof system consists of axioms about individual protocol actions and inference rules that yield assertions about protocols composed of multiple steps. Each provable assertion involving a sequence of actions holds in any protocol run containing the given actions and arbitrary additional actions by a malicious adversary. The PCL supports compositional reasoning about complex security protocols and has been applied to a number of industry standards including SSL/TLS, IEEE 802.11i and Kerberos V5.

8.2 Security properties to achieve in protocol design

Security properties, the level of security attainable, are the security goals of the cryptographic protocols. In this section, we adopt most of the security notions in [18] with some alteration, the security property requirements are augmented to provide security guarantees following a natural language argument, similar to the one that a protocol designer might be used to convince himself of the correctness of a protocol. Then, we give the expressions of some security properties following the security definition in trusted freshness formalism, where the security properties could not be implemented via direct cryptographic algorithm application.

Recall some notations. P_i and P_j are principals indexed by subscript which range over participants of the protocol run. N and N' are freshness identifiers which can be nonce, timestamps, session keys or shared parts of a session key. k is a cryptographic key and k^{-1} is the inverse of k. m is an arbitrary message.

The most popular aspects of information security are confidentiality, data integrity, entity authentication, and data origin authentication. Other security properties include non-repudiation, availability, access control and fairness. Most often the cryptosystem is used in conjunction with primitives providing these security properties. One of the fundamental tools used in information security is the signature. It is a building block for many other services such as non-repudiation, data origin authentication, identification, and fairness, to mention a few. However, key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

8.2.1 Confidentiality

Confidentiality (or secrecy, or privacy) is a service used to keep the content of information from all but those authorized to have it. Methods for providing confidentiality range from physical protections to mathematical algorithms which render data unintelligible.

The typical mathematical algorithms for providing confidentiality service include the following:

— Share-key encryptions (block ciphers and stream ciphers).
— Public-key encryptions.
— Keyed hashes.

8.2.2 Data integrity

Data integrity is a service which addresses the unauthorized modifications of data since the time it was created, transmitted, or stored by an authorized source. Modifications include creating, writing, deleting, changing, changing status, and delaying or replaying of transmitted messages.

The typical mathematical algorithms for providing data integrity service include the following:

— Message authentication codes (MACs, such as block ciphers with MDC, keyed hashes).
— Digital signature schemes.
— Appending (prior to encryption) a secret authenticator value to encrypted text.

Note that encryption provides protection only from passive attacks since the attacker could not get the sensitive information from the eavesdropping encryption without the decryption key, while an active attacker may replay the message or construct a fraudulent representation where the data integrity security is needed.

Verification of data integrity requires that only a subset of all candidate data items satisfy particular criteria distinguishing the acceptable from the unacceptable. Criteria allowing recognizability of data integrity include appropriate redundancy or expectation with respect to format. Data integrity includes the notion that data items are complete. For items split into multiple blocks, the alterations, such as insertion of bits, deletion of bits, and re-ordering of bits or groups of bits, apply analogously with blocks envisioned as substrings of a contiguous data string[18].

8.2.3 Data origin authentication

All secret keys used for encryption or data origin authentication should remain secret as long as the data secured there (the protection lifetime).

Authentication is a service related to the identification of a message or an entity. Authentication is usually subdivided into two major classes: data origin authentication and entity authentication. The first concerns validating a claimed property of a message, such as origin, date of origin, data content, and time sent; the second ensures a claimed identity of a message sender, that is, the identity of the party entering into a communication is not false.

Authentication depends on context of usage; data origin authentication focuses on the identity of the source of data; entity authentication focuses on the identity of a party, and the liveness of the party at a given instant; (implicit) key authentication focuses on the identity of party which may possibly shared a key; key confirmation focuses on the evidence that a key is possessed by some party; (explicit) key authentication focuses on the evidence that an

identified party possesses a given key[18].

Data origin authentication (or message authentication, or message source identification) is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some time in the past.

For example, suppose A and B are usually not in direct communication, and A wants to send an electronic mail message (e-mail) to B. The message may travel through various network communication systems and be stored for B to retrieve at some later time. B would like some means to verify that the message received and purportedly created by A did originate from A. Data origin authentication can provide this security service. Furthermore, data origin authentication implicitly provides data integrity since A would no longer be the originator if the message was modified during transmission.

A common misconception is that encryption provides data origin authentication and data integrity, under the argument that if a message is decrypted with a key shared only with party A, and if the decrypted message is meaningful, then the receiver is assured that the received encryption must have originated from A. Here "meaningful" means the message contains sufficient redundancy or meets some other priori expectations. There exists an intuition that an attacker must know the secret key in order to manipulate messages, however, this is not always true. In some cases the attacker may choose the plaintext message, while in other cases the attacker may manipulate effectively[19].

Example 8.1 Figure 8.1 illustrates a protocol without data-origin authentication security service.

Message 1 $A \rightarrow B$: $\{I\ am\ Alice.\ Wait\ at\ 6pm\ at\ the\ garden\ gate.\}_{K_{AB}}$

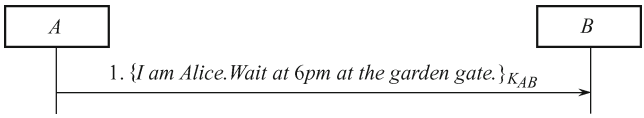


Fig. 8.1 A protocol illustration without data-origin authentication security service.

In the above example, the attacker recorded this message $\{I\ am\ Alice.\ Wait\ at\ 6pm\ at\ the\ garden\ gate.\}_{K_{AB}}$, as shown in Fig. 8.2, at a previous run of this protocol and then replayed this message to Bob by impersonating Alice. That is:

Message 1 $I(A) \rightarrow B$: $\{I\ am\ Alice.\ Wait\ at\ 6pm\ at\ the\ garden\ gate.\}_{K_{AB}}$

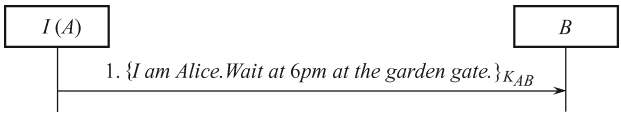


Fig. 8.2 An attack on the Example 8.1 without data-origin authentication security.

Data origin authentication necessarily involves identifying the source of a

message and data integrity, but no uniqueness and timeliness guarantees.

Although MACs and digital signatures may be used to establish that data was generated by a specified party at some time in the past, these techniques cannot alone detect message re-use or replay, which is necessary in environments where messages may have renewed effect on second or subsequent use. Hence, data-origin authentication necessarily involves establishing freshness of a message, while again, data integrity could be a piece of stale data which has perfect data integrity. Requiring that a message be fresh follows a common sense that a fresh message implies a good correspondence between the communication principals, and this may further imply less likelihood that, e.g., the communication principals, apparatus, systems, or the message itself may have been sabotaged. To obtain an effective data-origin authentication service, a message receiver should verify whether or not the message has been sent sufficiently recently (that is, the time interval between the message issuance and its receipt is sufficiently small)[19]. Hence, a message authentication code, a digital signature, etc. should include a trusted freshness to guarantee the message is new generated for a particular protocol run.

By definition, data origin authentication includes data integrity, and they are two very different notions. Data-origin authentication is a security service for a message receiver to verify whether a message is from a purported source and it necessarily involves communications. While data integrity security service can be provided on stored data, hence data integrity security service can be provided without message source identification. Note that a piece of stale data can have perfect data integrity. Besides identifying the source of a message, data-origin authentication should also convince the receiver that this message is sent by the intended partner for this particular protocol run but not a replay one, that is, this message is fresh. Otherwise, the attacker may record a message authentication code, a digital signature, etc. and replay this record to the receiver to impersonate the intended partner. A message, which is deemed by the receiver to have been issued sufficiently recently, is often referred to as a fresh message.

A fresh message without message source identification is not secure since the source of this message is the attacker but not the reputable principal source if a message has been modified in a malicious way. An example is illustrated as follow:

Example 8.2 Figure 8.3 shows a constructed cryptographic protocol without data-origin authentication security, whose intended goal is to establish a secret shared key k between two principals A and B via the shared parts N_A and N_B. N_A and N_B are nonce invented by A and B respectively. $SequenceNo$ is a sequence number. K_A, K_B are A's and B's public-keys respectively. The protocol part related to key establishment is

Message 1 $A \rightarrow B$: $\{SequenceNo, N_A\}_{K_B}$

Message 2 $B \rightarrow A$: $\{N_A, N_B\}_{K_A}$

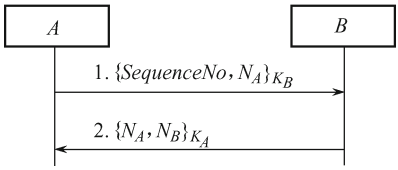


Fig. 8.3 A protocol without data-origin authentication security.

This protocol is insecure without data-origin authentication security of both A and B, hence we can construct attacks by cheating A and B respectively.

1. Attack 1 on the illustration protocol by cheating A

Message 1 $A \rightarrow I(B)$: $\{SequenceNo, N_A\}_{K_B}$
Message 1′ $I \rightarrow B$: $\{SequenceNo, N_A\}_{K_B}$
Message 2′ $B \rightarrow I$: $\{N_A, N_B\}_{K_I}$
Message 2 $I(B) \rightarrow A$: $\{N_A, N_B\}_{K_A}$

Attack 1 is a perfect attack, as shown in Fig. 8.4. In attack 1, the adversary I knows the shared parts of N_A and N_B of this protocol run to establish the new session key k, hence I will know the subsequent communication content between A and B.

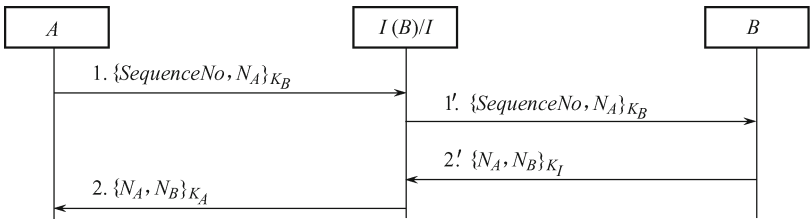


Fig. 8.4 Attack on the protocol without data-origin authentication security.

2. Attack 2 on the illustration protocol by cheating B

Message 1 $I(A) \rightarrow B$: $\{SequenceNo, N_A\}_{K_B}$
Message 2 $B \rightarrow I(A)$: $\{N_A, N_B\}_{K_A}$

In attack 2 as shown in Fig. 8.5, the principal B believes that B is sharing the shared parts N_A and N_B with A to establish the new session key k, while A knows nothing about this protocol run.

As we have illustrated above, this protocol is insecure without data-origin authentication security. Hence, a message authentication code, a digital signature, etc. should include a trusted freshness and provide the corroborative data-origin evidence to show that this is a message for a particular protocol

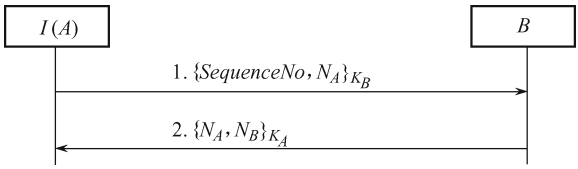


Fig. 8.5 Another attack on the protocol without data-origin authentication.

run.

The typical mathematical algorithms for providing data origin authentication service include the following:

1) Message authentication codes (MACs, such as block ciphers with MDC, keyed hashes) with a trusted freshness.

2) Digital signature schemes with a trusted freshness.

3) Appending (prior to encryption) a secret authenticator value including a trusted freshness to encrypted text.

Note that data origin authentication provided by a digital signature is valid only when the secrecy of the signer's private key is maintained. A threat which may be addressed is that a signer intentionally discloses his private key, and thereafter claims that a previously valid signature was forged. Similar problems exist with credit cards and other methods of authorization.

8.2.4 Entity authentication

Entity authentication considers techniques designed to allow one party (the verifier) to gain assurances that the identity of another (the claimant) is as declared, thereby preventing impersonation. Entity authentication involves corroboration of a claimant's identity through actual communications with an associated verifier during execution of the protocol itself (i.e., in real-time, while the verifying entity awaits). The most common technique is by the verifier checking the correctness of a message (possibly in response to an earlier message) which demonstrates that the claimant is in possession of a secret associated with the genuine party. That is, a message is provided to B along with additional information so that B can determine the identity of the entity A that originates the message.

Entity authentication doesn't necessarily provide guarantees of timeliness and uniqueness. For example, entity authentications are typically provided by fixed-password schemes and by certificates from an accepted authority like CA, and those forms of entity authentication provide no guarantee of timeliness, but they are useful in situations where the timeliness guarantee may not be the most important thing, such as in a passive attackers' environment, or one of the parties is not active in the communication.

Entity authentication (or identification of source) 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 was active at the time the evidence was created or acquired. The entity authentication can be broadly subdivided into unilateral entity authentication, mutual entity authentication, that is, either one or both parties may corroborate their identities to the other, providing, respectively, unilateral or mutual identification.

Unilateral entity authentication technique is assured of a corroborated identity of a second party and that this party is active at the protocol run.

Mutual entity authentication technique is assured of corroborated identities of both protocol parties and that they are active at the protocol run. Mutual authentication may also be obtained by running any of the unilateral authentication mechanisms twice.

An entity authentication protocol (also called identification protocol) typically involves no meaningful messages other than the claim of being a particular entity and an actual communication to show possession of a secret associated with the claimed genuine party. Here are some examples:

Example 8.3 An entity authentication protocol via fixed-password scheme, as shown in Fig. 8.6.

Message 1 $A \rightarrow B$: *I'm A.*
Message 2 $B \rightarrow A$: *Your Password:*
Message 3 $B \rightarrow A$: *%&*A356(*

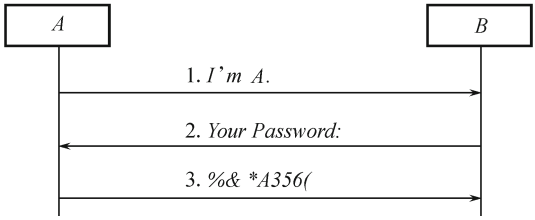


Fig. 8.6 An illustration of an identification protocol to show possession of a password.

Example 8.4 Figure 8.7 illustrates an identification protocol. The principal A wants to authenticate itself to B via possession of A's private key K_A^{-1}.

Message 1 $A \rightarrow B: \{I'm\ A.\}_{K_A^{-1}}$

Recall that entity authentication doesn't necessarily provide guarantees of timeliness and uniqueness. The timeliness guarantee is typically provided by appropriate use of time-variant parameters (TVPs). The TVPs include random numbers, sequence numbers and timestamps in cryptographic protocols.

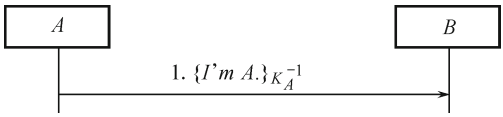


Fig. 8.7 An illustration of an identification protocol to show possession of A's certificate.

Example 8.5 Figure 8.8 illustrates a protocol, in which the principal A wants to authenticate itself to B via possession of A's private key K_A^{-1} with a timestamp T_A to assure the timeliness.

Message 1 $A \rightarrow B: \{I'm\ A,\ T_A.\}_{K_A^{-1}}$

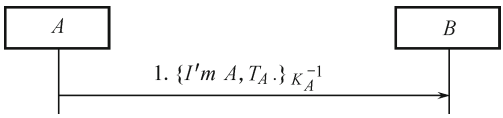


Fig. 8.8 An illustration to show possession of A's certificate with timeliness.

The distinction between entity authentication and data origin authentication is that the former acquires an actual communication to show possession of a secret associated with the claimed genuine party, whereas the latter need not acquire an actual communication but an evidence to show the (original) source of the specified data created at some time in the past. Entity authentication involves identifying the opponent participant's identity of a protocol run with timeliness guarantee or without, while data origin authentication itself provides no timeliness guarantees with respect to when a message was created.

If an entity authentication protocol identifies the opponent participant's identity via the fixed-password scheme, then it may be attacked by offline password-guessing through a classical password dictionary or a dictionary based on rainbow table.

If an entity authentication protocol identifies the opponent participant's identity via possession of a secret associated with the claimed genuine party, then this protocol may be attacked by message replay. See Example 8.6.

Example 8.6 An attack on Example 8.4. $I(A)$ is the adversary I impersonating A, and $\{I'm\ A.\}_{K_A^{-1}}$ is an old message recorded by the adversary I. Fig. 8.9 illustrates an attack on above example.

Message 1 $I(A) \rightarrow B: \{I'm\ A.\}_{K_A^{-1}}$

In this attack, the adversary I may intercept the message $\{I'm\ A.\}_{K_A^{-1}}$ when A wants to authenticate itself to B, or record this message when A wants to authenticate itself to I, and then replay this message to B to authenticate itself to B by impersonating A.

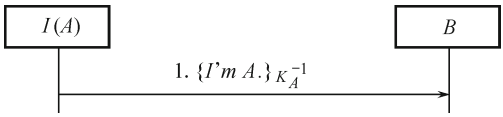


Fig. 8.9 An attack on Example 8.4.

The timeliness guarantee allows replay detection via a sequence number included within the data of messages authenticated by a MAC or digital signature algorithm, or via a random number, which is sent by the other party in the previous message, covered by a MAC or digital signature algorithm on each of the second and subsequent messages. For the random number schemes, the MAC values or signatures in the replayed messages would be incorrect for the disagreement of the random number in the replayed message and the most recent random number of the new protocol run. However, the timeliness guarantee still could not assure the detection of a replay message in a man in the middle attack. As for Example 8.5, the attack in Example 8.6 may still work.

Example 8.7 An attack on Example 8.5. I is the adversary, and $I(A)$ is the adversary I impersonating A. Figure 8.10 illustrates an attack on above example.

Message 1 $A \rightarrow I$: $\{I'm\ A,\ T_A.\}_{K_A^{-1}}$

Message 1′ $I(A) \rightarrow B$: $\{I'm\ A,\ T_A.\}_{K_A^{-1}}$

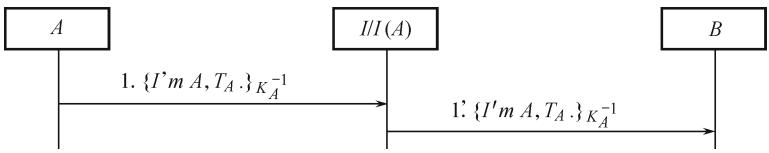


Fig. 8.10 An attack on Example 8.5.

In this attack, the adversary I intercepts the message $\{I'm\ A,\ T_A.\}_{K_A^{-1}}$ when A wants to authenticate itself to I, and then replays this message to B to authenticate itself to B by impersonating A.

Recall the UA-Secure and MA-Secure notion in Theorems 6.8 and 6.9, and we give the UA-Secure and MA-Secure rule under the belief multiset formalism.

Rule 8.1 (UA-secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. We say P_j has authenticated itself to the participant P_i if we have $b_{P_i,t_\$} = \lfloor \langle 1P_j \rangle \rfloor$.

Rule 8.2 (MA-secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full

control of the communication links. We say P_i has authenticated itself to the participant P_j and P_j has authenticated itself to the participant P_i if we have $b_{P_i,t_\$} = \lfloor\langle 1P_j\rangle\rfloor$ and $b_{P_j,t_\$} = \lfloor\langle 1P_i\rangle\rfloor$.

The typical mathematical algorithms for providing entity authentication service include the following:

— Message authentication codes (MACs, such as block ciphers with MDC, keyed hashes).
— Digital signature schemes.
— Public-key encryptions/decryptions.
— Appending (prior to encryption) a secret authenticator value to encrypted text.

Public-key encryption is usually combined with a digital signature, providing confidential key transport with source identity assurances. The intended recipient authenticates itself by returning some time-variant value which it alone may produce or recover. This may allow authentication of both the entity and a transferred key. Schemes using public-keys (transported by certificates) require signatures for verification, but signatures are not required within protocol messages.

8.2.5 Origin entity authentication

The uniqueness guarantee is necessary to assure the origin entity authentication. The uniqueness guarantee may be provided by the appropriate use of TVPs which should be associated with the participants of a particular protocol run. We can change Example 8.5:

Example 8.8 Figure 8.11 illustrates an entity authentication protocol via possession of A's private key K_A^{-1}. The timestamp T_A is associated with the principals A and B by the explicitly indication of the identities of both A and B.

Message 1 $A \rightarrow B$: $\{I'm\ A,\ B,\ T_A.\}_{K_A^{-1}}$

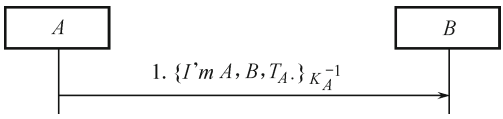


Fig. 8.11 An illustration of identification protocol with timeliness and uniqueness

Rule 8.3 (Origin UA-secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links, and N is a time-variant parameter. We say P_j has authenticated itself to the origin participant P_i if we have

$b_{P_i,t_\$} = \lfloor \langle P_i 1 P_j \rangle \rfloor$.

That is, P_i believes that the intended opposite participant P_j is specially in lively correspondence with this origin participant P_i in this protocol run.

Rule 8.4 (Origin MA-secure) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links, and N is a time-variant parameter. We say P_j has authenticated itself to the origin participant P_i and P_j has authenticated itself to the origin participant P_i if we have $b_{P_i,t_\$} = \lfloor \langle P_i 1 P_j \rangle \rfloor$ and $b_{P_j,t_\$} = \lfloor \langle P_j 1 P_i \rangle \rfloor$.

The typical mathematical algorithms for providing origin entity authentication service include the following:

— Message authentication codes with trusted freshness.
— Digital signature schemes with trusted freshness.
— Public-key encryptions /decryptions with trusted freshness.
— Appending (prior to encryption) a secret authenticator value to encrypted text with trusted freshness.

As for Example 8.8, the attack in Example 8.6 may not work.

8.2.6 Non-repudiation

Non-repudiation is a service which prevents an entity from denying previous commitments or actions. A procedure involving a trusted third party is needed to resolve the dispute where an entity may deny that certain commitments were made or certain actions were taken. Commonly used fairness security in electronic commerce protocols can be derived from non-repudiation.

Data origin authentication mechanisms based on shared secret keys (e.g., MACs) do not allow a distinction to be made between the parties sharing the key, and thus (as opposed to digital signatures) do not provide non-repudiation of data origin since either party can equally originate a message using the shared key. If resolution of subsequent disputes is a potential requirement, then either an on-line trusted third party in a notary role or asymmetric techniques may be used.

Rule 8.5 (Non-repudiation) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links, and N is a time-variant parameter. We say P_j could not deny that certain commitments were made or certain actions were taken to the participant P_i by P_j if we have $b_{P_i,t_\$} = \lfloor \langle 1 P_j \rangle \rfloor$.

The typical mathematical algorithms for providing non-repudiation service include the following:

1) Message authentication codes with an on-line trusted third party.

2) Digital signature schemes.

3) Appending (prior to encryption) a secret authenticator value to encrypted text.

Data origin authentication focuses on identity of the source of data, and this is typically provided by symmetric-key origin authentication mechanisms and signature mechanisms; non-repudiation focuses on being able to convince others at some time in the future that the non-repudiation data m was valid at some time in the past. Non-repudiation via symmetric-key origin authentication mechanisms may be accepted by a verifier as a form of authorization in an environment of mutual trust. Digital signatures are non-repudiable mechanisms in nature.

Example 8.9 As shown in Fig. 8.12 a party A has signed and sent a sensitive message m, such as a car order, to B, and B has corroborative evidence that the received message m is from A. Hence, A could not deny A's sending of message m.

Message 1 $A \rightarrow B: \{I \text{ order 2 cars.}\}_{K_A^{-1}}$

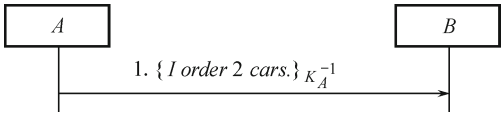


Fig. 8.12 An illustration of non-repudiation protocol.

This non-repudiation protocol is flawed for being without timeliness. The attacker I may record this message $\{I \text{ order 2 cars.}\}_{K_A^{-1}}$ and replay it to B. This protocol may be fixed by the appropriate use of TVPs, and it is like $\{T, I \text{ order 2 cars.}\}_{K_A^{-1}}$ where T is a TVP such as the date of order. However, this fixed protocol may still be attacked by an attack similar to the one in Example 8.6. The uniqueness guarantee is also necessary to assure the genuine non-repudiation. Similar to Origin entity authentication, the uniqueness guarantee may be provided by the appropriate use of TVPs which should be associated with the participants of a particular protocol run. Hence we have the origin non-repudiation security.

Rule 8.6 (Origin non-repudiation) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links, and N is a time-variant parameter. We say P_j could not deny that certain commitments were made or certain actions were taken to the origin participant P_i by P_j if we have $b_{P_i,t_\$} = \lfloor \langle P_i 1 P_j \rangle \rfloor$.

That is, P_i believes that P_j could not deny that certain commitments were specially made or certain actions were specially taken to the origin participant P_i.

The typical mathematical algorithms for providing origin non-repudiation

service include the following:

1) Message authentication codes including a trusted freshness with an on-line trusted third party.

2) Digital signature schemes with trusted freshness.

3) Appending (prior to encryption) a secret authenticator value to encrypted text with trusted freshness.

8.2.7 Access control

Access control is a service which addresses the authorization of a party to access some information resources. Only authorized parties may access the information resources of the target system.

Typical methods for providing access control service include the following:

— Fixed-password schemes with plaintext password file. The most obvious approach is for the system to store user passwords cleartext in a system password file, which is both read- and write-protected. Upon password entry by a user, the system compares the entered password to the password file entry for the corresponding user id; employing no secret keys or cryptographic primitives such as encryption, this is classified as a non-cryptographic technique.

— Fixed-password schemes with "Encrypted" password files. Rather than storing a cleartext user password in a plaintext password file, a one-way function of each user password is stored in place of the password itself. To verify a user-entered password, the system computes the one-way function of the entered password, and compares this to the stored entry for the stated user id.

— Fixed-password schemes with salting. Each password, upon initial entry, may be augmented with a t-bit random string called a salt before applying the one-way function. Both the hashed password and the salt are recorded in the password file. When the user subsequently enters a password, the system looks up the salt, as altered or augmented by the salt.

— Fixed-password schemes with time factor. Each password may be augmented with a t-bit random string of time before applying the one-way function. Upon access control entry by a user, the system looks up the password for the stated user id, gets the system time, applies the one-way function to the password and several timestamps (lie in the time window round the getting system time), then compares this to the user's entry. Time factor increases the timeliness and the complexity of a dictionary attack against a large set of passwords simultaneously, by requiring the dictionary to contain 2^t variations of each trial password.

— Two-factor authentication schemes. Two-factor authentication is a security process in which the user provides two means of identification, one

of which is typically a physical token, such as a card, and the other of which is typically something memorized, such as a security code. In this context, the two factors involved are sometimes spoken of as something you have and something you know. A common example of two-factor authentication is a bank card: the card itself is the physical item and the personal identification number (PIN) is the data that goes with it.

Some security procedures now require three-factor authentication, which involves possession of a physical token and a password, used in conjunction with biometric data, such as fingerscanning or a voiceprint.

One of the primary purposes of entity authentication is to facilitate access control to a resource, when an access privilege is linked to a particular identity (e.g., access to software applications; physical entry to restricted areas or border crossings). Furthermore, origin entity authentication is required to provide safe access control.

8.2.8 Key establishment

Key establishment is a service used to establish shared secrets, which are typically called or used to derive session keys for communication principals in an insecure channel. Key establishment protocol is any process whereby a shared secret key becomes available to two or more parties, for subsequent cryptographic use. Key establishment protocol can be broadly subdivided into key agreement protocol and key transport protocol.

Ideally, a session key is an ephemeral secret, i.e., the one whose use is restricted to a short time period such as a single telecommunications connection (or session), after which all trace of it is eliminated. While privacy of keying material is a requirement in key establishment protocols, source authentication is also typically needed. Key establishment is essentially entity authentication and message authentication where the message is the key.

Key transport protocol is a key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s) as a session key.

Key agreement protocol is a key establishment technique in which a session key is derived by two (or more) parties as a function of information contributed by or associated with each of these, (ideally) so that no party can predetermine the resulting value.

Authenticated key establishment protocol is to establish a shared secret with a party whose identity has been (or can be) corroborated.

Unilateral authenticated key establishment technique assures a corroborated identity of a second party and a corroborated shared of a secret key only with this active party at the protocol run.

Mutual authenticated key establishment technique assures corroborated identities of both protocol parties and a corroborated shared of a secret key

only between these two active parties at the protocol run.

The typical mathematical algorithms for providing key establishment service include the following:

— Message authentication codes with trusted freshness.
— Digital signature schemes with trusted freshness.
— Appending (prior to encryption) a secret authenticator value to encrypted text with trusted freshness.

Rule 8.7 (Unilateral authenticated key transport protocol) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links, and k is a temporary session key selected by the principal P_j. We say that the protocol Π is UK-secure if we have $b_{P_i,t_\$} = \lfloor \langle 11kP_iP_j \rangle, \langle 1P_j \rangle \rfloor$.

Rule 8.8 (Mutual authenticated key transport protocol) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links, and k is a temporary session key selected by the principal P_j. We say that the protocol Π is MK-secure if we have $b_{P_i,t_\$} = \lfloor \langle 11kP_iP_j \rangle, \langle 1P_j \rangle \rfloor$ and $b_{P_i,t_\$} = \lfloor \langle 11kP_iP_j \rangle, \langle 1P_j \rangle \rfloor$.

Rule 8.9 (Mutual authenticated key agreement protocol) Given a protocol Π between partners P_i and P_j in the presence of a probabilistic polynomial-time adversary I that has full control of the communication links. The temporary session key k is an output of a function of all protocol participants' random input N_{P_i} and N_{P_j}, which are randomly input by P_i and P_j respectively. We say that the protocol Π is MK-secure if we have $b_{P_i,t_\$} = \lfloor \langle 11N_{P_i}P_iP_j \rangle, \langle 11N_{P_j}P_iP_j \rangle, \langle 1P_j \rangle \rfloor$ and $b_{P_j,t_\$} = \lfloor \langle 11N_{P_i}P_iP_j \rangle, \langle 11N_{P_j}P_iP_j \rangle, \langle 1P_i \rangle \rfloor$.

8.2.9 Fairness

Fairness is a service to keep each honest protocol participant to have sufficient evidence (through acquisition of corroborative evidence) to solve the argumentation between or among parties, which may arise in or after a protocol run. It is the most important security service in an electronic commerce protocol.

The typical mathematical algorithms for providing fairness service include the following:

— Message authentication codes including a trusted freshness with an online trusted third party.
— Digital signature schemes with trusted freshness.
— Appending (prior to encryption) a secret authenticator value to encrypted text with trusted freshness.

Electronic commerce protocol is any process to provide secure electronic trades over network for two (or more) parties. The focuses of electronic commerce protocols are fairness and non-repudiation.

8.3 Protocol design based on trusted freshness

In this section, we will introduce the belief multiset design model for developing a cryptographic protocol with guarantee that the security of an authentication protocol is adequate based on security definition in Chapter 4.

This work is inspired by the probabilistic indistinguishability approach[4, 6], CK model[2, 5] and BAN-like logic[1, 20, 21]. CK model presents a definition, SK-security, for the security of a key establishment protocol, and it guarantees that any key establishment protocol that satisfies the security definition SK-security could be composed with symmetric encryption and authentication functions to provide a provably secure communication channel. In CK model, the security proof of a key establishment protocol is accomplished via an authenticator that translates a secure protocol in the Authenticated-links Adversarial Model (AM) into an equivalent protocol in the Unauthenticated-Links Adversarial Model (UM). However, for lack of authenticators, the proof method based on CK model is not so operational as the authors have expected.

Based on the freshness principle presented in Chapter 4, a simple and operational modal logic, the belief multiset design formalism, is given to express the security of a key establishment protocol and to reason about beliefs in the protocol. Proofs in the logic are therefore quite insightful.

The central idea behind the model is the observation that a participant's beliefs about security should depend only on a loose one-way transformation that binds together with some item whose freshness has been verified by the participant. Here loose one-way transformation means that for certain encrypted components of messages, only a regular protocol participant who has possession of a secret can change these encrypted components in a way that will be accepted by other regular participants. That is, the adversary cannot apply any non-trivial actions on this encrypted component, otherwise the message receiver will detect the alteration. These results allow us to achieve many authentication results without any further consideration of the dynamic execution of protocols, which could involve the activity of several principals. Instead, we need only consider the possible behavior of each principal independently.

The belief multiset language in belief multiset design model is almost the same as that in belief multiset formalism in Chapter 7. We assume that principals are not trustworthy, encrypted messages contain sufficient redundancy, and principals can recognize and ignore their own messages. In the belief multiset design model, a protocol is derived to achieve the security

objects as indicated in Section 8.2 step by step.

When designing a cryptographic protocol, the following two steps are essential[18]:

1) identify all assumptions in the protocol design;

2) for each assumption, determine the effect on the security objective if that assumption is violated.

In a protocol, a message is typically sent to one intended recipient who is designated implicitly, explicitly in the plain text or in encrypted parts of messages. However, due to the asynchronous nature of contemporary networks, a principal may receive any messages transmitted on the network, and the network-messages may be completely lost or even duplicated due to network failures and errors. Time is a key problem to keep a message distinct and fresh. The simplest way of introducing time is to assume the existence of a global clock. But unfortunately, in an insecure networked environment, such an assumption is best restrictive, and worst unrealistic. Furthermore, although in general the availability of synchronized clocks and the use of timestamps often can reduce the numbers of messages and rounds for cryptographic protocols, the efficiency of the protocols is not increased either, because all parties have the chance to exchange nonce in the whole protocol run[15].

We advocate that a cryptographic protocol be designed using encrypted messages that carry as much information as possible (the identities of the protocol run, the components to guarantee the timeliness and the uniqueness) regarding the current authentication run, to the extent that it becomes self-contained and uniquely identifiable as belonging to a particular authentication run.

8.3.1 Notations and descriptions

Similar to the belief multiset formalism in Chapter 7, we write a message exchange in a protocol run as $\{...N...\}_k$ where N is a freshness component trusted by the participant, and we omit messages or message parts that do not contribute to the security logical properties of the protocol to be achieved. We require that each exchanged message in a protocol should be dissimilar (see Definition 4.14) to avoid replay attack.

8.3.1.1 Events

The basic interactions among participants are modeled by events in the belief multiset design approach. An event is a message exchange operation where the message is distinct for being bound together by signature or encryption with some item whose freshness has been verified by the participant. Each event is principal identity indexed, that is to say, an event is related to a participant.

We use an abstract notion of distributed logic time to describe the order of events. This is based on an observation about the protocol message exchange: in distributed network environment, each participant has a local notion before and after that defines a total order on each partner's message exchanges, but there is no global absolute ordering of message exchanges. Hence, it is not necessary that time is global and synchronized in this circumstance because the lack of synchronization would not be observable and thus could not cause problem. We adopt the non-standard notion of Lamport clock[13]. Lamport pointed out that clock synchronization need not be absolute and that what usually matters is not that all partners agree on exactly what time it is, but rather, that they agree on the order in which events occur.

We introduce a notion causal consistency to enhance concurrency in protocol message exchanges. Message exchanges potentially causally related must be seen in the same order by all principals. Concurrent exchanges may be seen in a different order by different principals. Suppose event e_1 generates a message $\{...m_1...\}_k$, event e_2 reads $\{...m_1...\}_k$, and event e_3 sends back $\{...m_1...\}_{k'}$. Here the reading of $\{...m_1...\}_k$ and the writing of $\{...m_1...\}_{k'}$ are potentially causally related because the generation of $\{...m_1...\}_{k'}$ depends on the read of $\{...m_1...\}_k$ (by e_2) to get m_1.

Definition 8.1 An event is a term exchange operation among participants. Events are subscripted by principal identity to indicate different occurrence of the same message exchange for various participants. The collection of events for a particular participant is equipped with a total order, while the collection of events from all participants is not required to be equipped with a total order.

The same term exchange is not the same event for different participants. Suppose P_i has sent a message m, and the opposite principal P_j will receive this message m, then each principal has different events e_{P_i} and e_{P_j} respectively.

Each term appearing in the rules of belief multiset formalism in Subsection 7.4.2 could be an event. Here are some typical event illustrations:

1) $+\{...N_{P_i}...\}$ indicates that the participant P_i has sent the message $\{...N_{P_i}...\}$, and N_{P_i} is a freshness identifier generated by P_i.

This event could be applied to the freshness rule A5(b), and then P_i could get the freshness property of N_{P_i} from it.

2) $-\{...N|N'...\}_{K_{P_iP_j}}$ indicates that the participant P_i has received the message $\{...N|N'...\}_{K_{P_iP_j}}$, $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, N is a trusted freshness identifier of P_i, and N' is new a freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(a), P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N.

3) $-\{...N|N'|P_j...\}_{K_{P_iS}}$ indicates that the participant P_i has received the

message $\{...N|N'|P_j...\}_{K_{P_iS}}$, K_{P_iS} is the shared long-term key between P_i and the trusted third party S, N is a trusted freshness identifier of P_i, and N' is a new freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(b), then P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N.

4) $-\{...N|N'...\}_{K_{P_i}}$ indicates that the participant P_i has received the message $\{...N|N'...\}_{K_{P_i}}$, the decryption key $K_{P_i}^{-1}$ is the private key of P_i, N is a trusted freshness identifier of P_i, and N' is a new freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(c), then P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N.

5) $-\{...N|N'|P_i...\}_{K_{P_j}^{-1}}$ indicates that the participant P_i has received the message $\{...N|N'|P_i...\}_{K_{P_j}^{-1}}$, the encryption key $K_{P_j}^{-1}$ is the private key of P_j, the decryption key K_{P_j} is known by all participants. Note that N is a trusted freshness identifier owned by P_i, N' is a new freshness identifier whose freshness has not been verified by P_i, meanwhile the explicit identity of P_i indicates that N and N' are associated with a session related to P_i.

This event could be applied to the fragment rule A1(d), then P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N.

6) $+\{...N|N'...\}_{K_{P_iP_j}}$ indicates that the participant P_i has sent the message $\{...N|N'...\}_{K_{P_iP_j}}$, $K_{P_iP_j}$ is the shared long-term key between P_i and P_j, N is a trusted freshness identifier of P_i, N' is a new freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(e) or expectation rule A2(a), then P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N or could assert that only the partner P_j can obtain the freshness identifier N.

7) $+\{...N|N'|P_j...\}_{K_{P_iS}}$ indicates that the participant P_i has sent the message $\{...N|N'|P_j...\}_{K_{P_iS}}$, K_{P_iS} is the shared long-term key between P_i and the trusted third party S, N is a trusted freshness identifier of P_i, and N' is a new freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(f), then P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N.

8) $+\{...N|N'|P_i...\}_{K_{P_j}}$ indicates that the participant P_i has sent the message $\{...N|N'|P_i...\}_{K_{P_j}}$, K_{P_j} is the public-key of P_j, N is a trusted freshness identifier of P_i, and N' is a new freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(g) or the expectation rule A2(b), then P_i could assert that the new freshness identifier N' is bound

to the trusted freshness identifier N, or assert that only the partner P_j can obtain the freshness identifier N.

9) $+\{...N|N'...\}_{K_{P_i}^{-1}}$ indicates that the participant P_i has sent the message $\{...N|N'...\}_{K_{P_i}^{-1}}$, the encryption key $K_{P_i}^{-1}$ is the private key of P_i, N is a trusted freshness identifier of P_i, and N' is a new freshness identifier whose freshness has not been verified by P_i.

This event could be applied to the fragment rule A1(h), then P_i could assert that the new freshness identifier N' is bound to the trusted freshness identifier N associated with a session related to P_i.

8.3.1.2 Causal consistency

There exists a total order on the collection of events corresponding to a particular participant, hence each event of this participant could be superscript by the happen order of the event, e.g., $e^1_{P_i}, e^2_{P_i}$ and so on.

We use $<_{P_i}$ to denote the relation "happen before". If $e <_{P_i} e'$, then we say that the event e precedes the event e' for the participant P_i; if $e \approx_{P_i} e'$, then we say that the event e and the event e' are the same events.

Recall that the collection of events from all participants is not required to be equipped with a total order. For example, if P_i sends a message m while opposite principal P_j receives this message m, then each principal has different events e_{P_i} and e_{P_j} respectively. In this case, what counts is whether event e_{P_i} is older or newer than e_{P_j}, not their absolute creation time. We define causal consistency notion to ensure that the send event is preceded by the receive event for the same message exchange.

Definition 8.2 (Causal Consistency) We use E_{P_i} to denote the collection of events corresponding to principals P_i and E to denote the union of all of the E_{P_i}. We say E has the property of causal consistency if there is a partial order $<$ on the events of E so that

— if $e <_{P_i} e'$ then $e < e'$;
— the event sent is preceded by the event received for the same message exchange.

We emphasize that causal consistency is only a consistency requirement and it does not imply a global ordering of events, where concurrency in protocol message exchanges will not be expressed in terms of a fixed event ordering. Hence these concurrent protocol message exchanges can be implemented in the same round[22].

In belief multiset design model, events are related to term exchanges that include a trusted freshness component. Message exchanges in multiple sessions of a protocol or instances of multiple protocols, or potential interactions among them are subsequently considered as distinct events.

8.3.1.3 Beliefs and rules

Belief is the key element of protocol security in belief multiset design model and it differs greatly from that in BAN logic or in Heintze-Tydel model. The beliefs in this model can only be deduced from the received fresh and confidential messages and the beliefs already possessed by certain party while BAN deals with all messages; this approach is independent of the formalization of cryptographic protocols themselves, the concrete formalization of an adversary's possible behavior, concurrent run with any set of protocols while beliefs in Heintze-Tydel model are used to describe one aspect of a participant state at each point. In this model, a belief is pointwise valid, and it is asynchronous. Recall that instead of using a concrete notion of time, we use causal consistency to reason about time and to describe the beliefs at each point, and this results in no loss of generality. We need to state what it means for particular beliefs held in the model.

A belief is a trust about the liveness of legitimate participants, the confidentiality, freshness and association of a freshness component (including nonce, timestamp, session key or shared parts of a session key), and also the fragment beliefs, the expectation beliefs defined in belief multiset analysis approach.

A belief can only be deduced from a fresh message which is bound with certain trusted freshness identifier. A belief multiset is an unordered collection of beliefs owned by each of the legitimate participants.

Intuitively, initial beliefs are things that are assumed to be true before the start of the protocol in cryptographic literature. For example, in symmetric cryptosystems case, each participant knows the shared key with other parties or a trusted third party.

The liveness property of a legitimate participant, the freshness and association property of a freshness component are monotone increasing, while the belief about the confidentiality of a message is supposed to be true at the start of the protocol run, and it may become open at any point of the protocol run.

In the belief multiset design model, the fragment beliefs and the expectation beliefs are:

1) An expectation belief: An expectation $\prec \{\hat{m}, P_j\}$ owned by P_i who asserts that only the partner P_j can obtain the term $\hat{m}$ from a one-way transformation that is sent by P_i.

2) A fragment belief: A fragment $\sim\{...N, N'...\}_k$ owned by P_i who asserts the binding of a new freshness identifier N' with a trusted freshness identifier N.

Given a participant P_i accompanied with a set of beliefs and a set of events, a protocol specifies a set of legal transitions that a participant can take.

The events and beliefs in this model are fundamentally different from those in Heintze-Tydel model: the belief multiset beliefs are trusts about the liveness of legitimate participants, and trusts about the confidentiality, freshness and association of some freshness components while beliefs in Heintze-

Tydel model are used to capture one aspect of a principal state at each point; our events include only message exchange operations where the messages are distinct for being bound together by signature or encryption with a trusted freshness identifier while events in Heintze-Tydel model include all send operations, receive operations or even generate operations. Based on (1) what the participant currently believes (about principals and freshness components) and (2) an event (such as the receipt of a fresh message), each participant adds new beliefs to its set of current beliefs, and takes actions in accordance with the protocol such as sending a fresh message.

Rules are a set of legal transitions that reflect the underlying assumptions in cryptographic literature. The rules in the belief multiset design model are the same as those in the belief multiset analysis approach in Subsection 4. These rules are carefully constructed to correspond to certain intuitions about how participants interact and how a protocol unfolds. A more comprehensive response is that if the assumptions the rules based on are satisfied, the protocol will be correct.

8.3.2 Design of cryptographic protocols

With the hypothesis indicated in Subsection 6.3.6.2, the steps for protocol design based on belief multiset are refined as follows.

The security requirements of a cryptographic protocol

Step 1. The security requirements of a cryptographic protocol varies for different applications. Here we list some necessary chosen about the security design of a cryptographic protocol, such as the security goals, cryptographic mechanism.

1. The security goals to achieve

1) Unilateral Entity Authentication.
2) Origin Unilateral entity authentication.
3) Mutual Entity Authentication.
4) Origin Mutual entity authentication.
5) Unilateral authenticated key transport.
6) Mutual authenticated key transport.
7) Mutual authenticated key exchange (or key agreement).

2. The way to get freshness for a cryptographic protocol

1) Nonce, that is Challenge-Response protocol.
2) Timestamp.
3) Sequence number.

3. Cryptographic mechanism

1) Public-key algorithm.

2) Share-key algorithm,including cypher block and keyed hash.

4. With or without the trusted third party

1) With the trusted third party.
2) Without the trusted third party.

Initial states and premises of cryptographic protocols

Step 2. We must stipulate the cryptographic conditions under which the protocol will operate. Assume that each party has a shared long-term key with its participant in shared key case without the trusted third party; Assume that each party has a shared long-term key with the trusted third party in shared key case with the trusted third party. Assume that each party has an asymmetric key pair, and holds the other's public-key or can reliably obtain it via a one-time setup. Perhaps some public-key infrastructure is already in place.

Furthermore, the private key or the shared long-term key is commonly assumed to be too long to guess in a computationally feasible way. Therefore, the adversary does not know whether a guess is correct or not, and guessing attack is rendered impotent. Assume that the encryption or signature schemes in use can meet the security strength requirements for a particular protocol, such as secure against chosen ciphertext attacks and so on. The participants of a cryptographic protocol are definitely indicated and so does the role of each participant: the initiator, the responder, or the trusted third party.

Suppose P_i and P_j are participants, ρ is an arbitrary principal which ranges over participants of the protocol run including the attacker I, S is the trusted third party, k is a new session key randomly chosen by the participant P_i, P_j, or the trusted third party S, or generated from the random input by both P_i and P_j, then the initial assumptions related to the protocol participants are:

1) public-key case:
$B_{P_i,t_0}(\langle 11K_{P_i}^{-1}P_i\rangle)$, $B_{P_i,t_0}(\langle 11K_{P_j}^{-1}P_j\rangle)$, $B_{P_i,t_0}(\langle 01K_{P_i}\rho\rangle)$, $B_{P_i,t_0}(\langle 01K_{P_j}\rho\rangle)$ and
$B_{P_j,t_0}(\langle 11K_{P_j}^{-1}P_j\rangle)$, $B_{P_j,t_0}(\langle 11K_{P_i}^{-1}P_i\rangle)$, $B_{P_j,t_0}(\langle 01K_{P_j}\rho\rangle)$, $B_{P_j,t_0}(\langle 01K_{P_i}\rho\rangle)$

2) Shared key case without trusted third party:
$B_{P_i,t_0}(\langle 11K_{P_iP_j}^{-1}P_iP_j\rangle)$ and $B_{P_j,t_0}(\langle 11K_{P_iP_j}^{-1}P_iP_j\rangle)$

3) Shared key case with trusted third party:
$B_{P_i,t_0}(\langle 11K_{P_iS}^{-1}P_iS\rangle)$ and $B_{P_j,t_0}(\langle 11K_{P_jS}^{-1}P_jS\rangle)$.

Security goals based on the trusted freshness

Step 3. Security goals and premises can be formally and accurately presented based on the trusted freshness. The security goals of typical applications of authentication protocols are listed below:

1) Unilateral Entity Authentication: P_i wants to authenticate the identity of P_j. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle 1P_j\rangle\rfloor$ based on the trusted freshness.

2) Origin Unilateral Entity Authentication: P_i wants to authenticate the identity of P_j, and N (a nonce, a timestamp, or a sequence number) is a challenge generated by P_i for P_j to authenticate. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle P_i 1 P_j\rangle\rfloor$ based on the trusted freshness N.

3) Mutual Entity Authentication: P_i and P_j want to authenticate each other. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle 1 P_j\rangle\rfloor$ and $b_{P_j,t_\$} = \lfloor\langle 1 P_i\rangle\rfloor$ based on the trusted freshness.

4) Origin Mutual Entity Authentication: P_i and P_j want to authenticate each other, N_{P_i} and N_{P_j} are challenges generated by P_i and P_j respectively to authenticate the opponent participant. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle P_i 1 P_j\rangle\rfloor$ and $b_{P_j,t_\$} = \lfloor\langle P_j 1 P_i\rangle\rfloor$ based on the trusted freshness P_i and P_j respectively.

5) Unilateral Authenticated Key Transport: P_i wants to authenticate the identity of P_j, and P_i believes that the session key generated by the authenticated participant P_j or a TTP can provide a secure channel over an insecure network. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle 1 P_j\rangle, \langle 11 k P_i P_j\rangle\rfloor$ based on the trusted freshness.

6) Mutual authenticated key transport: P_i and P_j want to authenticate each other, and both P_i and P_j believe that the session key generated by one of the authenticated participants or a TTP can provide a secure channel over an insecure network. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle 1 P_j\rangle, \langle 11 k P_i P_j\rangle\rfloor$ and $b_{P_j,t_\$} = \lfloor\langle 1 P_i\rangle, \langle 11 k P_i P_j\rangle\rfloor$ based on the trusted freshness.

7) Mutual authenticated key exchange (or key agreement): P_i and P_j want to authenticate each other, both P_i and P_j believe that the new session key, which is the output of a function of all protocol participants' random input N_{P_i} and N_{P_j}, can provide a secure channel over an insecure network. The security goal could be given as $b_{P_i,t_\$} = \lfloor\langle 11 N_{P_i} P_i P_j\rangle, \langle 11 N_{P_j} P_i P_j\rangle, \langle 1 P_j\rangle\rfloor$ and $b_{P_j,t_\$} = \lfloor\langle 11 N_{P_i} P_i P_j\rangle, \langle 11 N_{P_j} P_i P_j\rangle, \langle 1 P_i\rangle\rfloor$ based on the trusted freshness.

Message design of a cryptographic protocol

Step 4. According the requirements of the protocol security, proper events should be selected from the event list in Subsection 8.3.1.1. From the initial states and the premises defined in Subsection 3, the security properties of a cryptographic protocol could be established based on inference rules in Subsection 7.4.2.2 and the selected events. Repeat step 4, and establish the security properties from current beliefs and the fresh messages, which are constructed from the selected events including a trusted freshness identifier. Repeat this step 4 until the security goals in Subsection 3 are met.

8.3.3 Lower bounds for SK-secure protocols

In this section, we discuss the efficiency of the constructed cryptographic pro-

tocol under the belief multiset design model. We first introduce two metrics, the number of messages and the number of rounds.

Definition 8.3 The number of messages is the total number of message exchanges required to complete the protocol.

Definition 8.4 A round consists of all messages that can be sent and received in parallel within one time unit. The number of rounds in a protocol is the total number of time units from the instant that the originator sends the first message till the instant that the last message is received, under the best execution scenario[22].

With the notation of the distributed logic time causal consistency, the concurrent messages of the cryptographic protocol are determined, then these concurrent messages can be transmitted in the same message round (sent out at the same time), to optimize the design of the cryptographic protocol. Other measures could also be applied to improve the performance of a protocol.

Let's analyze the number of messages and the number of rounds in a cryptographic protocol:

1. As for confidentiality, non-repudiation and data integrity properties

They needn't have a communication feature: they could be provided on stored data.

2. As for data origin authentication property

It necessarily involves communications: the lower bounds are one message and one round, that is, the receiver should convince that the receiving message is originally sent by the intended data originator. Data-origin authentication necessarily involves establishing the freshness of a message, otherwise it may be a replayed message by an attacker.

3. Common entity authentication

We refer common entity authentication to the "loose" entity authentication to distinguish it from origin entity authentication. As for "loose" entity authentication, the lower bounds are one message and one round if a timestamp is adopted, otherwise, the lower bounds are two messages and two rounds if nonce or sequence number is adopted in a challenge response protocol; As for origin entity authentication, the lower bounds are the same as those in the "loose" entity authentication. While in origin entity authentication, the freshness identifier (timestamp, nonce or sequence number) should be associated with the participants of the communication.

Here are the examples of "loose" entity authentication and origin entity authentication:

Example 8.10 Suppose A and B are the principals of the protocol run, T is a timestamp from A, key K_A^{-1} is the private key of A. A wants to authenticate the identity of A to the opponent participant, that is, the "loose" entity

authentication, as shown in Fig. 8.13.

Message 1 $A \rightarrow B: \{I\ am\ Alice,\ T.\}_{K_A^{-1}}$

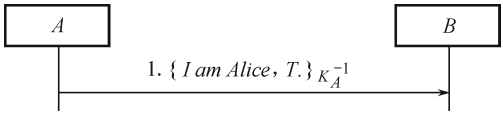


Fig. 8.13 An illustration of the "loose" entity authentication security service.

The "loose" entity authentication protocol is not secure since the principal A may be impersonated by an attacker I with the existence of A. Here is the attack on Example 8.10.

Example 8.11 Suppose A and B are the principals of the protocol run, I is an attacker, T is a timestamp from A, and the key K_A^{-1} is the private key of A. A wants to authenticate A to the principal I, meanwhile the attacker I may use this signature $\{I\ am\ Alice.\}_{K_A^{-1}}$ to authenticate $I(A)$ to the principal B by impersonating A. Figure 8.14 illustrates an attack on the protocol of the "loose" entity authentication security service.

Message 1 $A \rightarrow I: \{I\ am\ Alice,\ T\}_{K_A^{-1}}$

Message 1′ $I(A) \rightarrow B: \{I\ am\ Alice,\ T\}_{K_A^{-1}}$

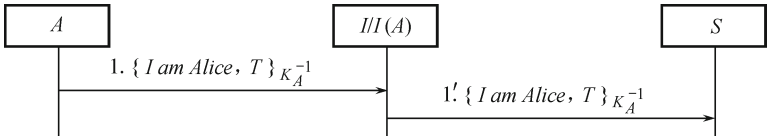


Fig. 8.14 An attack on the protocol in Example 8.10.

As for origin entity authentication, the evidence to show the existence of an entity should be associated with all of the protocol run principals to avoid the attack of replay like Example 8.11.

Example 8.12 Suppose A and B are the principals of the protocol run, T is a timestamp from A, key K_A^{-1} is the private key of A. As shown in Fig. 8.15, A wants to authenticate A itself to the particular principal B, that is, origin entity authentication.

Message 1 $A \rightarrow B: \{I\ am\ Alice,\ B,\ T\}_{K_A^{-1}}$

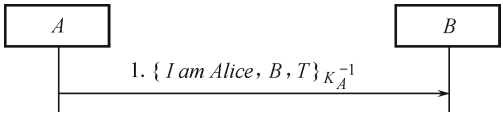


Fig. 8.15 A protocol illustration of the origin entity authentication security service.

In this example, the signature $\{I\ am\ Alice,\ B,\ T\}_{K_A^{-1}}$ is used as the explicit evidence of the existence of the entity A to the particular principal B in this particular protocol run. From this signature, according to the association rule A6(d) in the belief multiset formalism, the freshness identifier T is associated with both the principals A and B. Hence, B asserts that it is really A who has generated this message, and this signature is indeed sent to B to authenticate A for this particular protocol run between A and B in this moment T, that is, the origin entity authentication.

To achieve access control property, "loose" entity authentication is usually used in real world, hence there may exist the attack similar to that in the Example 8.11.

4. As for unilateral authenticated key establishment protocol

The key is generated by one of the participants and then transmitted to its opponent participant. The lower bound are one message and one round if a timestamp is adopted, while the lower bound are two messages and two rounds if a nonce or a sequence number is adopted.

Example 8.13 As shown in Fig. 8.16, we design a unilateral authenticated key transport protocol via the shared key algorithm and the timestamp. Suppose A and B are the principals of the protocol run, T is a timestamp from A, K_{AB} is the shared long-term key between A and B, and the new session key k_{AB} is generated by A and then transmitted to the principal B.

Message 1 $A \rightarrow B:$ $\{k_{AB}, T\}_{K_{AB}}$

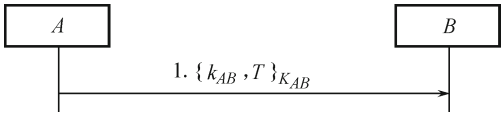


Fig. 8.16 An illustration of the unilateral authenticated key establishment protocol.

According to the freshness rule A5(a) and the association rule A6(a) in the belief multiset formalism, the principal B believes that the new session key k_{AB} is fresh and associated with both A and B from the trusted freshness identifier T. Hence, according to the security definition of UK-Secure and the conditions to guarantee the UK-Secure in Definition 4.5, the protocol in Example 8.13 is UK secure.

Example 8.14 As shown in Fig. 8.17, we adopt the public-key algorithm, and the timestamp to design a key transport protocol. Suppose A and B are the principals of the protocol run, T is a timestamp from A, K_A^{-1} is the private key of A, and the new session key k_{AB} is generated by A and then transmitted to the principal B.

Message 1 $A \rightarrow B:$ $\left\{\{k_{AB}, T\}_{K_A^{-1}}\right\}_{K_B}$

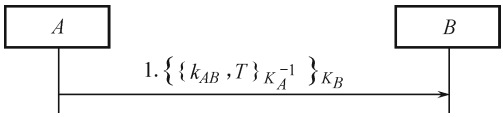


Fig. 8.17 An illustration of the key transport protocol using public-key algorithm.

According to the freshness rule A5(a) in the belief multiset formalism, the principal B believes that the new session key k_{AB} is fresh; from the association rule A6(e), B believes that the session key k_{AB} is associated with B, and it is the run of the moment T, but B could not assert that k_{AB} is associated with A. Hence, B could not assert that the protocol in Example 8.14 is UK secure. Here is an attack on the protocol in Example 8.14.

Example 8.15 Suppose A and B are the principals of the protocol run, T is a timestamp from A, I is an attacker and the key K_A^{-1} is the private key of A. As shown in Fig. 8.18, A wants to generate a new session key k_{AB} for principals A and I, and then transmit it to the principal I, in this time the attacker I may use this signature $\{k_{AB}, T\}_{K_A^{-1}}$ to authenticate $I(A)$ to the principal B by impersonating A.

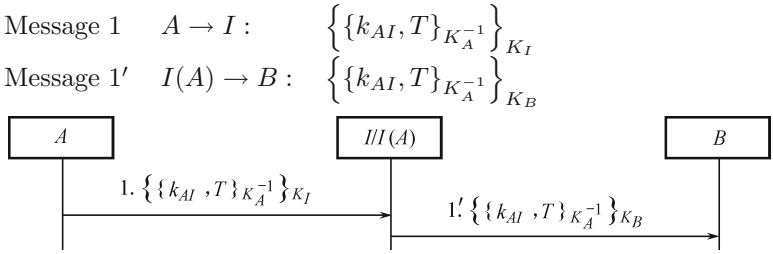


Fig. 8.18 An attack on the protocol in Example 8.14.

Upon the termination of the protocol run in Example 8.15, B believes that the session key k_{AI} is fresh and associated with both A and B while k_{AI} is indeed associated with I and B, and A knows nothing about the protocol run with B. From the attack in Example 8.15, we find that the opponent participant responder B could not believe the origin entity authentication of the initiator A. To achieve the security goals in the unilateral authenticated key establishment protocol, the message in Example 8.14 could be fixed as:

Example 8.16 Fig. 8.19 illustrates a revision of Example 8.14.

Message 1 $A \to B$: $\left\{\{k_{AB}, B, T\}_{K_A^{-1}}\right\}_{K_B}$

For the revision of the key transport protocol using public-key algorithm, the attack 8.53 will not work on the Example 8.54.

5. As for mutual authenticated key transport protocol

The key is generated by one of the participants and then transmitted to its opponent participant. The lower bound is two messages and two rounds if

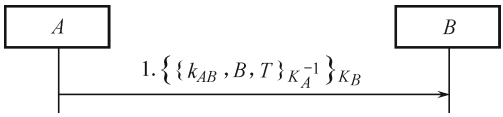


Fig. 8.19 A revision of the key transport protocol using public-key algorithm.

a timestamp is adopted, while the lower bound is three messages and three rounds if a nonce or a sequence number is adopted.

Example 8.17 In Fig. 8.20, we adopt the public-key algorithm, and the timestamp to design a mutual authenticated key transport protocol. Suppose A and B are the principals of the protocol run, T is a timestamp from A, K_A^{-1} and K_B^{-1} are the private key of A and B respectively. In the mutual authenticated key transport protocol, A and B want to authenticate each other, and the new session key k_{AB} is generated by A and then transmitted to the principal B.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \left\{\{k_{AB}, B, T\}_{K_A^{-1}}\right\}_{K_B} \\ \text{Message 2} & B \rightarrow A: & \left\{\{k_{AB}, T\}_{K_B^{-1}}\right\}_{K_A} \end{array}$$

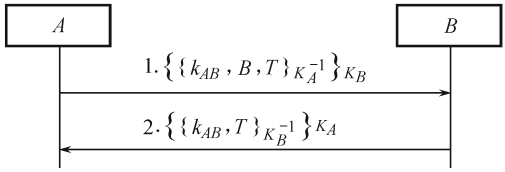


Fig. 8.20 An illustration of the mutual authenticated key transport protocol.

6. As for mutual authenticated key agreement protocol

The key is an output of a function of all protocol participants' random input, (ideally) so that no party can predetermine the resulting value. The lower bound is two messages and two rounds if a timestamp is adopted, while the lower bound is three messages and three rounds if a nonce or a sequence number is adopted.

As an illustration, let's analyze the number of messages and the number of rounds in a key agreement protocol using nonce. According to the security definition of SK-Security and the conditions to guarantee the SK-Security in Definition 6.3, both the originator and the responder should convince that the opposite participant is in lively correspondence in this protocol run. First, the originator notifies the opposite participant of starting the protocol with a trusted freshness identifier N. Next, the opposite participant sends this identifier N back to the originator. From the liveness rule A5, the originator could establish the belief about the responder's in lively correspondence with the originator and generates a fresh identifier N' by the originator itself. Thus

a lower bound is three messages. Since beliefs are achieved based on some items whose freshness is trusted by the participant, thus three rounds are a lower bound.

Theorem 8.1 (Lower Bounds) Let Π be an SK-secure challenge-response public-key key establishment protocol, then the lower bounds are three messages and three rounds.

Proof The main proof technique for the lower bounds on the number of messages and rounds is to identify a critical path – a causal chain of messages – that cannot be further shortened.

Suppose A, B are the principals of the protocol run, k_{AB} is the new session between A and B. The sufficient and necessary conditions to guarantee the security of a key establishment protocol are specified in Chapter 4, which can be expressed as $b_{A,t_\$} = \lfloor\langle 11 k_{AB} AB\rangle, \langle 1B\rangle\rfloor$ and $b_{B,t_\$} = \lfloor\langle 11 k_{AB} AB\rangle, \langle 1A\rangle\rfloor$ in belief multiset formalism.

According to the SK-Security definition, the originator A has to notify opposite participant B of starting the protocol with a trusted freshness identifier N_A as a shared part N_A of a new session key k, there should exists an event e_A^1 such as $e_A^1 : +\{N_A\}_{K_{AB}}$, or $e_A^1 : +\{N_A, B\}_{K_{AS}}$ or $e_A^1 : +\{N_A, A\}_{K_B}$; the responder B needs to send N_A back in security, the event e_A^2, so that A can establish its beliefs about the liveness of the principal B and the freshness and association of the shared part N_A of the new session key k. Obviously we have $e_A^1 <_A e_A^2$. On the contrary, the responder B needs to send a trusted freshness identifier N_B to A, the event $e_B^1 : +\{N_B\}_{K_{AB}}$, or $e_B^1 : +\{N_B, A\}_{K_{BS}}$ or $e_B^1 : +\{N_B, B\}_{K_A}$, and get the response from the originator A, the event e_B^2. Obviously we have $e_B^1 <_B e_B^2$. Since e_B^1 could not happen before B notifies the nonce of N_A by the originator, according to causal consistency, we have $e_A^1 < e_B^1$. Since there is not any causal logic between e_A^2 and e_B^1, these concurrent message exchanges could be implemented in the same message and in the same round, that is:

1) Round 1 and Message 1: e_A^1.

2) Round 2 and Message 2: e_A^2 and e_B^1, here the nonce N_A could be sent back to A by B in the same round with the new generated nonce N_B, and they could be sent back also in the same Message 2.

3) Round 3 and Message 3: e_B^2.

Therefore, three messages and three rounds are lower bounds.

8.4 Application of protocol design via trusted freshness

Design of a key establishment protocol is more complicated than that of other type cryptographic protocols. Hence we show the process to design a key establishment protocol under the belief multiset model as an illustration.

Key establishment is a foundational element for secure communications.

It concerns how to set up a new session key to protect communications during a subsequent session. Key establishment protocols are mechanisms to establish such common secret keys between pairs of parties over an adversary-controlled network.

Let's review the statement about key establishment in [18]: When designing or selecting a key establishment technique for use, it is important to consider what assurances and properties an intended application requires. Distinction should be made between functionality provided to a user, and technical characteristics which distinguish mechanisms at the implementation level (cost and performance variations). Characteristics which differentiate key establishment techniques include:

1) Nature of the authentication. Any combination of the following may be provided: entity authentication, key authentication, and key confirmation.

2) Reciprocity of authentication. When provided, entity authentication, key authentication, and key confirmation may separately be unilateral or mutual (provided to one or both parties).

3) Key freshness. A key is fresh (from the viewpoint of one party) if it can be guaranteed to be new, as opposed to possibly an old key being reused through actions of either an adversary or authorized party.

4) Key control. In some protocols (key transport), one party chooses a key value. In others (key agreement), the key is derived from joint information, and it may be desirable that neither party be able to control or predict the value of the key.

5) Efficiency. Considerations include:

(1) number of message exchanges (passes) required between parties;

(2) bandwidth required by messages (total number of bits transmitted);

(3) complexity of computations by each party (as it affects execution time), and

(4) possibility of precomputation to reduce on-line computational complexity.

6) Third party requirements. Considerations include:

(1) requirement of an on-line (real-time) party, off-line party, or no third party;

(2) degree of trust required in a third party (e.g., trusted to certify public-keys vs. trusted not to disclose long-term secret keys).

7) Type of certificate used, if any. More generally, one may consider the manner by which initial keying material is distributed, which may be related to third party requirements.

8) Non-repudiation. A protocol may provide some type of receipt that keying material has been exchanged.

In this section, the usability and the compositional adequacy of the belief multiset design model are demonstrated via the redesign of Needham-Schroeder key establishment protocol. We first introduce how to specify security properties in the model, then we detail the generation of a challenge-response public-key protocol for two-party authentication and key agreement,

at last the security of the protocol will be checked.

8.4.1 Construction of a two-party key establishment protocol

Before construction of two-party authenticated key agreement protocol in this model, we must first determine what the secrets are to be protected in a key establishment protocol: session keys or shared parts of a session key. This is typically implicit in the protocol specifications, and should be made explicit in the model.

The security requirements of a cryptographic protocol

It is important to start by deciding the goals to be achieved. Let us assume that we intend to construct a protocol in which the initiator A and the responder B each generates a fresh, secret value, N_A and N_B respectively. They want to shared these values between themselves without disclosing them to any other party. Each should learn that the other has proceeded far enough in the protocol to receive the values. The two principals intend to hash the two values together to produce a new session key for the future encrypted conversation. The intended goal of this challenge-response public-key protocol is to establish secure communication (SK-secure) between two principals A and B via the shared parts N_A and N_B. In the model, the security goals can be expressed as:

$$b_{A,t_\$} = \lfloor \langle 1B \rangle, \langle 11 N_A AB \rangle, \langle 11 N_B AB \rangle \rfloor, \text{and}$$
$$b_{B,t_\$} = \lfloor \langle 1A \rangle, \langle 11 N_A AB \rangle, \langle 11 N_B AB \rangle \rfloor.$$

The freshness guarantee is typically provided by an appropriate use of a fresh random number, nonce, in a challenge response protocol, and the public-key cryptographic mechanism without a trusted third party will be anticipant in the Needham-Schroeder key exchange protocol.

The initial state and the premise of a cryptographic protocol

Assume that each party has its private key as well as the public-keys of all parties before the start of key establishment protocol, the initial assumptions related to the protocol participants are: $b_{A,t_0} = \lfloor \langle 11 K_A^{-1} A \rangle, \langle 11 K_B^{-1} B \rangle, \langle 01 K_A \rho \rangle, \langle 01 K_B \rho \rangle \rfloor$ and $b_{B,t_0} = \lfloor \langle 11 K_B^{-1} B \rangle, \langle 11 K_A^{-1} A \rangle, \langle 01 K_A \rho \rangle, \langle 01 K_B \rho \rangle \rfloor$. ρ is an arbitrary principal including the adversary.

Message design of a cryptographic protocol

With the security goals and the premise set established, we could construct the challenge-response public-key protocol. In the belief multiset formalism design model, the participant only makes responses to the terms which include trusted freshness. Hence, it requires each participant to send a nonce before receiving a temporal session key, which is a basis security

requirement based on the challenge-response mechanism. That is, a participant without a synchronized clock could not accept any temporal session key before sending a nonce generated by itself.

Here are some tips for a cryptographic protocol design:

1) It is better to explicitly indicate the identities sharing the new session key in the message including this session key.

2) It is better to confirm the ownership of the new session key via an encryption of a known plaintext by the opponent partner under this key.

3) There should not exist similar maximum terms (see Definition 4.10). That is, the structure of the sending message including a trusted freshness identifier should not be similar to any receiving terms.

Step 1 Unilateral authentication.

1) At the start of the communication, A should first establish a trusted freshness identifier. According to the event definition of 8.1(1) and the freshness rule A5(b), the originator A has to notify B of starting the protocol, and generates a fresh nonce N_A where A has $b_{A,t_1} = \lfloor\langle ...1N_A...\rangle\rfloor$ before sending the notification message.

2) To establish the confidential property of the session key shared part N_A, the freshness identifier N_A should be transmitted in an encryption, and the attacker I doesn't have the corresponding decryption key k^{-1}. Hence, the freshness identifier N_A can be encrypted under the public-key K_B of the participant B. According to the event Definition 8.1, we have the event $e_A^1 : +\{...N_A...\}_{K_B}$ and the event $e_B^1 : -\{...N_A...\}_{K_B}$. Obviously, we have the relationship of causal consistency between the events e_A^1 and e_B^1, that is $e_A^1 < e_B^1$. According to the confidentiality rule A3(d) and the expectation rule A2(b), A has $B_{A,t_1}(\langle 11N_A...\rangle)$ and $B_{A,t_1}(\prec \{N_A, B\})$.

3) For A to achieve unilateral authentication with B in public-key case, according to the event Definition 8.1, the confidentiality rule A3(d) and expectation rule A2(b), B needs to send N_A back to A in security, expecting only A and B know the secret shared parts N_A of the session key, hence we have the event $e_B^2 : +\{...N_A...\}_{K_A}$ and the event $e_A^2 : -\{...N_A...\}_{K_A}$. Obviously, we have the relationship of causal consistency "before" between the events e_A^1 and e_A^2, that is $e_A^1 <_A e_A^2$. We also have the events e_B^1 and e_B^2, that is $e_B^1 <_B e_B^2$. According to the liveness rule A4(c), A has the belief $B_{A,t_2}(\langle 1B\rangle)$.

Hence, for A to achieve unilateral authentication with B in public-key case, the message exchanges will be of the forms:

Message 1 $A \to B$: $\{...N_A...\}_{K_B}$

Message 2 $B \to A$: $\{...N_A...\}_{K_A}$

Up to now, A has achieved the belief $b_{A,t} = \lfloor\langle 1B\rangle, \langle ...N_B...\rangle, \langle 11N_A...\rangle\rfloor$ via the above message exchanges.

Step 2 Mutual authentication.

In step 1, A has achieved the unilateral authentication with B. Similarly, for B to achieve unilateral authentication with A in public-key case, B has

the event $e_B^{1'} : +\{...N_B...\}_{K_A}$, A has the events $e_A^{1'} : -\{...N_B...\}_{K_A}$ and $e_A^{2'} : +\{...N_B...\}_{K_B}$, we have the relationship of "before" between the events $e_A^{1'}$ and $e_A^{2'}$, that is $e_A^{1'} <_A e_A^{2'}$, and causal consistency before $e_B^1 < e_B^{1'}$, $e_B^{1'} < e_A^{1'}$. Hence, for B to achieve unilateral authentication with A in public-key case, the message exchanges are:

Message 1 $B \rightarrow A$: $\{...N_B...\}_{K_A}$

Message 2 $A \rightarrow B$: $\{...N_B...\}_{K_B}$

Up to now, B has achieved the belief $b_{B,t} = \lfloor\langle 1A\rangle, \langle ...N_A...\rangle, \langle 11N_B...\rangle\rfloor$ via the above message exchanges.

Note that there doesn't exist the relationship of causal consistency "before" between e_B^2 and $e_B^{1'}$. e_B^2 and $e_B^{1'}$ are concurrent message exchanges, so they could be transmitted concurrently. Combining the message exchanges considering causal consistency, both participants A and B could still achieve the mutual authentication secure:

Message 1 $A \rightarrow B$: $\{...N_A...\}_{K_B}$

Message 2 $B \rightarrow A$: $\{...N_A, N_B...\}_{K_A}$

Message 3 $A \rightarrow B$: $\{...N_B...\}_{K_B}$

Step 3 Key agreement.

1) For principal A

According to association rule A6(e), upon receiving Message 2, A has the belief $b_{A,t_2} = \lfloor\langle ...N_A A\rangle\rfloor$. To achieve the belief $b_{A,t_2} = \lfloor\langle ...N_A B\rangle\rfloor$, according to association rule A6(g), A should have the expectation belief $B_{A,t_1}(\prec \{N_A, A, B\})$, hence according to expectation rule A2(c), Message 1 could be revised to

Message 1 $A \rightarrow B$: $\{...A, N_A...\}_{K_B}$

So we have the event $e_A^1 : +\{...A, N_A...\}_{K_B}$, then A has the belief $b_{A,t_2} = \lfloor\langle ...N_A B\rangle\rfloor$. According to the event definition of 8.1 (4), there exists event $-\{...N_A, N_B...\}_{K_A}$. Then, according to fragment rule A1(c) and association rule A6(h), A has the belief $b_{A,t_2} = \lfloor\langle 11N_B AB\rangle\rfloor$. Hence, upon receiving Message 2, A has achieved the security goals of the Needham-Schroeder protocol in this step by the above message exchanges.

2) For principal B

According to fragment rule A1(g), for B to achieve the fragment belief $B_{B,t_2}(\sim \{...N_A, N_B...\}_{K_A})$, B should have the belief $B_{B,t_2}(\langle ...1N_B B\rangle)$ at step 2, hence Message 2 could be revised to

Message 2 $B \rightarrow A$: $\{...B, N_A, N_B...\}_{K_A}$

So we have the event $e_B^{1'} : +\{...B, N_B...\}_{K_A}$. From this event $e_B^{1'} : +\{...B, N_B...\}_{K_A}$, according to association rule A6(f), upon sending Message 2, B has the belief $B_{B,t_2}(\langle ...N_B B\rangle)$. Hence, according to fragment rule A1(g), B has the fragment belief $B_{B,t_2}(\sim \{...N_A, N_B...\}_{K_A})$. Upon sending Message 2, according to expectation rule A2(c), B has the belief $B_{B,t_2}(\prec \{N_B, B, A\})$. Upon receiving Message 3, according to association rule A6(g), B has the belief $b_{B,t_3} = (\langle ...N_B A\rangle)$. Then B has the belief $b_{B,t_3} = (\langle 11N_B AB\rangle)$. Ac-

cording to association rule A6(h), B has the belief $b_{B,t_3} = (\langle 11 N_A AB \rangle)$. Hence, upon receiving Message 3, B has achieved the security goals of the Needham-Schroeder protocol in this step by the above message exchanges.

Up to now, we have gotten the constructed Needham-Schroeder protocol:

Message 1 $A \rightarrow B: \quad \{...A, N_A...\}_{K_B}$

Message 2 $B \rightarrow A: \quad \{...B, N_A, N_B...\}_{K_A}$

Message 3 $A \rightarrow B: \quad \{...N_B...\}_{K_B}$

As we have seen, the constructed key establishment protocol in the model achieves the lower bounds of three messages and three rounds. Note that although Gong's nonce-based protocols without handshakes achieve lower bounds of two messages and two rounds[23], they are indeed not MK-secure and there exists denial of service attacks. Three messages and three rounds are lower bounds proven for nonce-based public-key protocols, which has been proven in Subsection 8.3.3.

In this section, we have exemplified the usability and the efficiency of the belief multiset design model for designing cryptographic protocols, and how the detailed security goals are achieved step by step. This stepwise method makes it possible for protocol designers to select efficient solutions to cater to various security requirements, such as unilateral authentication secure, mutual authentication secure or session key secure. Furthermore, we indicate that the developed key establishment protocol has achieved SK-Secure with lower bounds.

References

[1] Burrows M, Abadi M, Needham R (1990) A Logic of Authentication. ACM Transactions on Computer Systems 8(1): 18 – 36

[2] Bellare M, Canetti R, Krawczyk H (1998) A Modular Approach to the Design and Analysis of Authentication and Key-exchange Protocols. In: Proceedings of the 30th STOC, Dallas, 23 – 26 May 1998

[3] Heintze N, Tygar J (1996) A Model for Secure Protocols and Their Compositions. IEEE Transactions on Software Engineering 22(1): 16 – 30

[4] Bellare M, Rogaway P (1993) Entity Authentication and Key Distribution. In: CRYPTO'93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 22 – 26 Aug 1993. Lecture Notes in Computer Science, vol 773, pp 232 – 249, Springer

[5] Canetti R, Krawczy H (2001) Analysis of Key-exchange Protocols and Their Use for Building Secure Channels. In: EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, 6 – 10 May 2001. Lecture Notes in Computer Science, vol 2045, pp 453 – 474, Springer

[6] Goldwasser S, Micali S (1984) Probabilistic Encryption. Journal of Computer and System Sciences 28(2): 270 – 299

[7] Datta A, Derek A, Mitchell JC, Warinschi B (2006) Computationally Sound Composi-tional Logic for Key Exchange Protocols. In: Proceedings of the

19th IEEE Computer Security Foundations Workshop, Venice, 5 – 7 July 2006

[8] Canetti R, Rabin T (2003) Universal Composition with Joint State. In: CRYPTO'03 Proceedings of the 23rd Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, 17 – 21 Aug 2003. Lecture Notes in Computer Science, vol 2729, pp 265 – 281, Springer

[9] Dong L (2008) Cryptographic Protocol Engineering and Protocol Security Based on Trusted Freshness. PhD Dissertation (in Chinese), Shanghai Jiaotong University

[10] Dong L, Chen K, Lai X (2009) Belief Multisets for Cryptographic Protocol Analysis. Journal of Software 20(11): 3060 – 3076 (in Chinese)

[11] Buttyan L, Staamann S, Wilhelm U (1998) A Simple Logic for Authentication Protocol Design. In: Proceedings of the 11th IEEE Computer Security Foundations Workshop, Rockport, 9 – 11 June 1998

[12] Gong L, Syverson P (1995) Fail-stop Protocols: An Approach to Designing Secure Protocols. In: Proceedings of IFIP DCCA-5, Illinois, 27 – 29 Sept 1995

[13] Lamport L (1978) Time, Clocks and the Ordering of Events in a Distributed System. Communication of the ACM 21(7): 558 – 565

[14] Datta A, Derek A, Mitchell JC, Pavlovic D (2003) A Derivation System for Security Protocols and its Logical Foundation. In: Proceedings of the 16th IEEE Computer Security Foundations Workshop, Pacific Grove, 30 June – 2 July 2003

[15] Gong L (1995) Optimal Authentication Protocols Resistant to Password Guessing Attacks. In: CSDW'95 Proceedings of the 8th IEEE Workshop on Computer Security Foundations, County Kerry, Ireland, 13 – 15 June 1995

[16] Guttman JD, Thayer F (2000) Authentication Tests. In: Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, 14 – 17 May 2000

[17] Datta A, Derek A, Mitchell JC, Roy A (2007) Protocol Composition Logic (PCL). Electronic Notes in Theoretical Computer Science 172: 311 – 358.

[18] Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, New York

[19] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[20] Gong L, Needham R, Yahalom R (1990) Reasoning About Belief in Cryptographic Protocols. In Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, Oakland, 7 – 9 May 1990

[21] Syverson PF, Oorschot PCV (1994) On Unifying Some Cryptographic Protocol Logics. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Oakland, 16 – 18 May 1994

[22] Gong L (1993) Lower Bounds on Messages and Rounds for Network Authentication Protocols. In: CCS'93 Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, 3 – 5 Nov 1993

[23] Gong L (1994) Efficient Network Authentication Protocols: Lower Bounds and Optimal Implementations. Technical Report SRI-CSL-94-15, Computer Science LaboratorySRI International, 1994

9 Automated Analysis of Cryptographic Protocols Based on Trusted Freshness

Abstract Logics can be systematically applied to reasoning about the working of protocols. However, the process of applying and reapplying the inference rules is often tedious and error-prone when carried out manually, while automation method will improve this problem. An automated logic-based analysis tool based on the freshness principle is introduced and developed, which uses the belief multiset formalism to analyze the security of cryptographic protocols.

The freshness principle presented in Chapter 4 has proved to be an efficient and easy idea in analyzing the security of cryptographic protocols from its capacity in distinguishing whether a message is fresh or not based on already trusted freshness identifier. The inherent appeal in using modal logic to instance the freshness principle stems from logic's simplicity and effectiveness for analyzing cryptographic protocols[1–4]. The modal logic of security analysis based on trusted freshness, the belief multiset formalism, has been presented in Chapter 7. Logic can be systematically applied to reasoning about the working of protocols, often helping to reveal missing assumptions, deficiencies or redundancies. This can then lead to the protocol, the assumptions or the original goals being re-evaluated, after which the inference rules can be reapplied to determining whether the goals are attainable after these modifications have been made. However, the process of applying and reapplying the inference rules is often tedious and error-prone when carried out manually. Another problem is that protocol has become so advanced and complex that we often cannot perform certain security analysis by hand, and we may accidentally miss conclusions drawn from inference rules, the freshness principle or the informal principles based on trusted freshness. Hence, specialized tool support for formal methods can significantly aid protocol engineers in creating and implementing cryptographic protocols which do not leak information, do achieve security goals and are immune to replay attacks[5].

In this chapter, we will give an automated logic-based analysis tool based on the freshness principle (see Chapter 4), and this Prolog-based analyzer uses the belief multiset formalism (see Chapter 7) to analyze protocols and

it is developed mainly based on the concepts introduced during the SPEAR II Framework[5, 6].

9.1 Previously known methods for automated analysis

The cryptographic protocol automated analysis tools are useful for keeping track of whether or not a cryptographic protocol is secure or not.

9.1.1 Automated analysis tool based on logic

The symbolic manipulation correctness analysis approaches can greatly simplify the analysis of protocols, which consist of theoretic computer scientists in formal method area, and the security properties are expressed as a set of abstract symbols which can be manipulated, sometimes by a formal logic system, sometimes by an mechanical tool called a theorem prover, toward a YES/NO result[7].

SPEAR I, the Security Protocol Engineering and Analysis Resource[8], was developed by J.P. Bekmann, P. de Goede and A. Hutchison in 1997 to aid in the design and analysis of cryptographic protocols. The two primary goals of SPEAR I are to enable secure and efficient protocol design and to support the generation of protocol source code. SPEAR I offers developers of cryptographic protocols an environment in which security protocols are designed, analyzed and generated. Protocols are specified using a graphical user interface in the style of Event Trace diagrams and the security analysis based on the BAN logic[9] is facilitated.

SPEAR II, the Security Protocol Engineering and Analysis Resource II[6], is a protocol engineering tool built on the foundation of previous experience garnered during the SPEAR I project. The goal of the SPEAR II tool is to facilitate cryptographic protocol engineering and to aid users in distilling the critical issues during an engineering session by presenting them with an appropriate level of detail and guiding them as much as possible. The SPEAR II tool consists of four components that have been integrated into one consistent and unified graphical interface: a protocol specification environment (GYPSIE), a GNY statement construction interface (Visual GNY), a Prolog-based GNY analysis engine (GYNGER) and a message rounds calculator. GYNGER is a Prolog-based analyzer that performs automated analysis of protocols by using the GNY modal logic. Recall that the GNY logic could find the reflection replay attack (Suppose the message sender could recognize the message sent by itself), but it is hard for the GNY logic to find the transfer replay attacks and the direct replay attacks. The analysis engine employs a forward chaining approach to mechanize the tedious application of GNY in-

ference rules, allowing all derivable GNY statements to be generated quickly, accurately and efficiently. To conduct an analysis with GYNGER a protocol engineer needs to specify a protocol's messages, initial assumptions and target goals in a Prolog-style GNY syntax. The GNY rule set is then imported and employed in the analysis, after which a proof is generated in an English-style GNY syntax for every successful goal that was specified. Visual GNY functions as a user-friendly interface to the GYNGER analyzer. SPEAR II provides a graphically based protocol security analysis environment, and it can find the security flaws in a lot of authentication protocols. The idea of SPEAR II to implement GNY logic could be referenced for automation of protocol security analysis based on the logic – belief multiset formalism.

9.1.2 Automated analysis tool based on model checking

9.1.2.1 FDR tool

Failures Divergences Refinement (FDR)[10−12] is a model checker tool that is tailored to check CSP processes for refinement relations, where CSP stands for Communication Sequential Process[13]. CSP is particularly suitable for modeling and describing the behavior of concurrency and communication systems, and this feature has inspired some researchers for its using for formal analysis of authentication protocols. FDR models a complex system, such as an authentication protocol, into a (finite) state system and the properties of a state system can be expressed by some state satisfaction relations. FDR allows the refinement relation to be checked mechanically for finite state processes and it can be used to clarify whether or not certain properties of a complex system will be satisfied. In 1995, Lowe applied the FDR model checker and successfully uncovered a previously unknown error in the Needham-Schroeder Public-key Authentication Protocol, where this flaw has not been discovered for seventeen years since the publication of this protocol in 1978[11]. In FDR, a principal (it may be an attacker) in a protocol is considered as a concurrent CSP process, and a variety of attacks (such as eavesdropping, imitation or replaying) could be applied by an attacker process. The security of a protocol is modeled as the sequences of principal's events, and the FDR is used to check whether or not certain sequence of a principal's events is satisfied. The FDR model checker for CSP has achieved the success in analyzing the Needham-Schroeder Public-key Authentication Protocol.

9.1.2.2 NRL protocol analyzer

The NRL protocol analyzer[14] is a PROLOG-based protocol model-checking tool developed by Meadows, where "NRL" stands for Naval Research Laboratory of the United States of America. The NRL Protocol Ana-

lyzer is also based on the Dolev and Yao threat model of communications[15]. In Dolev-Yao model, an adversary could observe all message traffic over the network, intercept, read, modify or destroy messages, perform transformation operations on the intercepted messages (such as encryption or decryption, as long as he has in his possession of the correct keys), and send his messages to other principals by masquerading as some principals. Since an adversary's computational capability is polynomially bounded in the Dolev-Yao model, after an execution of the protocol, the adversary could not learn any information of the secret messages or cryptographic keys for which a protocol is meant to protect.

In Model-checking techniques the analysis of the behavior of a system usually involves a state space exploration to check whether or not certain properties will be satisfied. The main algorithm used in the NRL Protocol Analyzer settles a state reachability problem. It is well known that such algorithms are not guaranteed to terminate. Therefore a limit is placed on the number of recursive calls allowed for some of the checking routines. Using the tool seems to require quite a high level of user expertise in accurately coding the transition rules for a protocol and in specifying insecure state. The tool also has an inherent limitation on being particularly applicable to protocols for key establishment[7].

The NRL protocol analyzer has been used to analyze a number of authentication protocols and has successfully found or demonstrated known flaws in some of them. These protocols include the Needham-Schroeder Public-key Authentication Protocol[16] (for the analysis of this protocol Meadows provided a comparison between the analysis using the NRL protocol analyzer and Lowe's analysis using the model checker FDR in [11]), the Internet Key Exchange protocol (IKE, a reflection attack is found in the signature-based "Phase 2" exchange protocol)[17,18] and the Secure Electronic Transaction protocols (SET)[19].

9.1.2.3 The Murφ

Murφ[20] is a protocol verification tool that has been successfully applied to several industrial protocols, especially in the domains of multiprocessor cache coherence protocols and multiprocessor memory models. The Murφ language is a simple high-level language for describing nondeterministic finite-state machines. To use Murφ for verification, one has to model the protocol in the Murφ language and augment this model with a specification of the desired properties. The *state* of the model consists of the values of all global variables. The transition from one state to another is performed by *rules.* The desired properties of a protocol can be specified in Murφ by invariants, which are Boolean conditions that have to be true in every reachable state. The Murφ system automatically checks, by explicit state enumeration, if all reachable states of the model satisfy the given specification. Most Murφ models are nondeterministic since states typically allow execution of more than one ac-

tion. Murφ can only guarantee correctness of the down-scaled version of the protocol, but not correctness of the general protocol. If a state is reached in which some invariant is violated, Murφ prints an error trace – a sequence of states from the start state to the state exhibiting the problem. Murφ proves that there exist attacks on some known protocols such as Needham-Schroeder Public-key Authentication Protocol, TMN protocol and a simplified version of Kerberos V5[21–23].

9.1.2.4 Interrogator

The Interrogator[24] is a Prolog program developed by Jonathan Millen, Sidney Clark and Sheryl Freedman in 1985. Using the Interrogator, a protocol engineer can search for security vulnerabilities in network protocols for automatic cryptographic key distribution. Given a formal specification of a protocol, the Interrogator searches for message modification attacks that defeat the protocol objective and reveal secret information. The current version of the Interrogator assumes that the adversary is trying to learn private information, and the only way in which he can get that information is to read a message in which it is transmitted as a data item. A black-box view of the Interrogator is simple: for input it receives a protocol specification and a target data item; its output is a message history, consistent with the protocol specification, showing how the adversary could obtain the data item, if this is possible. The Interrogator and its associated graphical interface were implemented using LM-Prolog on an LISP machine. The user interface takes advantage of the windowing, graphics and mouse capabilities of the LISP machine. Within the Interrogator, protocols are modeled using a state-transition approach, principals being represented as communicating finite-state machines. This method allows a wider class of protocols to be supported and permits variations in message sequencing. The Interrogator interface has two main components: a preprocessor that converts textual protocol specifications into an internal Prolog form, and a display interface for graphical user interaction. To conduct an analysis, a protocol is specified in a textual format, edited with normal LISP machine facilities, parsed and loaded. The interactive graphical display is then used to establish penetration objectives. If a flaw is found, it is displayed in the form of a message sequence, showing messages before and after modification by an adversary. The Interrogator has been developed to the extent where it has succeeded in finding a multiple-modification penetration of the Needham-Schroeder protocol and some others with known vulnerabilities. Given a protocol specification and a target component to uncover, the Interrogator searches for a scenario involving adversary actions which reveal the target. The history of messages sent and modified is displayed, allowing the user to examine how the attack was carried out and evaluate it for feasibility and possible counter-measures.

9.1.3 Automated analysis tool based on theorem proving

SyMP[25] stands for "Symbolic Model Prover" and it is a general purpose prover generator for generating special purpose theorem provers in various application domains. SyMP is proposed by S. Berezin and A. Groce of Carnegie Mellon University. The core of the tool is a generic prover which is connected to several proof system modules. Each such module defines an input specification language, a proof system, and a rule application mechanism, and the generic prover provides all the proof management and an interactive user interface. Note that the SyMP prover does not have a built-in model checker

SyMP has two proof systems: the default proof system, and Athena. The default proof system implements a general framework for combining model checking and theorem proving, and has a hardware-oriented specification language. The main purpose of the language is to provide a convenient environment for fast and clean prototyping of new (mostly hardware) verification methodologies based on model checking with some elements of theorem proving. It can also be used as an intermediate representation in translation between other specification languages.

The Athena proof system is specialized in verifying security protocols, uses Strand Spaces[26] as the basis for the protocol representation and is based on the Athena[27] technique developed by D. Song. Each protocol in this framework is a set of roles, and each role is a sequence of actions where actions are separated by an optional semicolon. Two built-in actions, "send" and "receive", send and receive messages to and from the environment. An instance of a role with concrete parameters defines a strand, or a particular run of a particular principal in the protocol. Properties are specified in a propositional logic that specifies which strands must or must not appear in any protocol execution.

9.1.4 CAPSL specification language

CAPSL, the Common Authentication Protocol Specification Language[28], is a high-level language intended to support the analysis of cryptographic protocols using formal methods. The development of CAPSL started in 1996 and is being managed by Jonathan Millen. Its goal is to permit a protocol to be specified once in a form that is usable as an interface to any type of analysis tool or technique, given appropriate translation software. The CAPSL Intermediate Language (CIL) acts as an interface to analysis tools, allowing protocols specified in CAPSL to be examined by these tools. CIL is designed to make the translation to tool-specific representations as easy as possible. A CAPSL specification is parsed and translated into CIL, and at that point a different translator can convert from CIL to whatever form required for each

tool. The translator from CAPSL to CIL can deal with the universal aspects of input language processing, such as parsing, type checking, and unraveling a message-list protocol description into the underlying separate processes.

Messages in cryptographic authentication protocols are constructed using cryptographic operators and functions. In principle, all functions used in CAPSL, and the data types they operate on, must be specified axiomatically with abstract data type specifications, called typespecs. Several commonly used data types and operators are defined in a standard prelude. Type specifications in this prelude are considered built-in, and do not need to be supplied by a designer or imported explicitly. When a protocol is being analyzed or simulated, the analyst may have to specify which principals are to be run. Other run-specific information, such as the initial knowledge of the attacker, may also have to be supplied. A CAPSL environment contains specifications detailing this kind of information. Environment specifications, like type specifications, are separated from the definition of a protocol. The content and interpretation of an environment specification depend on the analysis tool. However, CAPSL does provide syntax, keywords and organization so that different tools can take advantage of the CAPSL parser. Declarations to name principals and other constants can be placed in an environment, and sessions can be defined. More than one session may be declared and these sessions will run concurrently by default.

9.2 Automated cryptographic protocol analysis based on trusted freshness

As we have seen, conducting an analysis manually is often tedious, error-prone and not very productive in the majority of situations. To facilitate the effective use and application of a logic system, it is needed to make the logic analysis automated and the user interface graphical.

In this section, we put forward a cryptographic protocol analysis analyzer based on the belief multiset formalism logic (for short, the BMF logic), the BMF analyzer, which is also a completely graphic-based environment. A benefit of the BMF analyzer is that it would make protocol analysis operations accessible to a wider range of individuals, since it would remove the requirement that protocol designers need to be experts in a large number of specialized engineering techniques.

9.2.1 Analyzer frame based on belief multiset formalism

The BMF analyzer consists of 5 parts: visual BMF, BMF analyze engine, BMF result view, BMF rule engine and BMF attack engine, and it involves

operations on the database of facts, goals, rules and attacks. The parts BMF analyze engine, BMF rule engine and BMF attack engine are the core of the BMF analyzer, while the parts visual BMF and BMF result view are the user-friendly interfaces to the BMF analyzer. The frame of the BMF analyzer illustrates in Fig. 9.1.

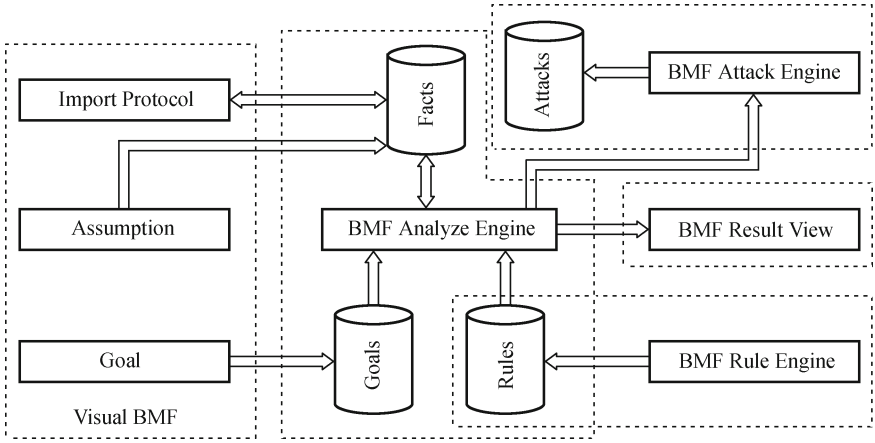


Fig. 9.1 The frame of the BMF analyzer.

1. Visual BMF

The visual BMF functions as a user-friendly interface to the BMF analyzer, and the input components include protocol messages, premises and goals. These components are constructed to BMF statements necessary for analysis via the visual BMF interface, and then passed on to BMF analyze engine. The protocol message import interface supports the input of the messages including the principals, nonce etc., supports the chosen of cryptographic schemes, and the construction of the BMF language component and the terms. From the view point of a participant in a protocol run, the terms owned by each principal are completely different. The premise import interface supports the input of the initial security assumptions of the public-key and private key in public-key case, the long-term key in the shared-key case, and also principal which has generated the freshness identifier used in the protocol messages. The goal import interface supports the security goal configuration of an input protocol, such as UA-secure, MA-secure, UK-secure MK-secure.

2. BMF analyze engine

The BMF analyze engine supports the formal presentations of terms, initial assumptions, security goals, inference rules, and it functions as a protocol security analyzer that generates all of the BMF beliefs and possessions that can result from the systematic application of the inference rules to a set of

initial assumptions and message steps. The BMF analyze engine mainly includes the database of premises, the database of goals, the database of rules, and the analyzing engine based on the trusted freshness.

3. BMF result view

The BMF result view subsystem supports the presentations of the security analysis result of a cryptographic protocol based on the belief multiset formalism.

4. BMF rule engine

The BMF rule engine is an interface of a new belief multiset formalism inference rule input. Based on the freshness principle, the belief multiset formalism inference rules could be extended to meet variety applications in the real world, such as new liveness rules, new confidential rules, new freshness rules, new association rules, also other new rules about non-repudiation, fairness etc. A new rule could be inputted into the BMF analyzer via a user-friendly interface, and then it is converted to a standard format rule with a series of operations via BMF analyze engine, and then it is passed on to the rules database.

5. BMF attack engine

Recall that the security analysis of a protocol based on the belief multiset formalism can either establish the correctness of the protocol when it is in fact correct, or identify the absence of the security properties and the structure to construct attacks based on the absence. The BMF attack engine supports the presentations of attacks in the belief multiset formalism, and supports the construction of attacks from the absence of the security properties.

9.2.2 Comparision of two initial implementations of BMF

The focus of the automation of a cryptographic protocol analyzer is to give a correct analysis result, and to remove as much of the complexity and tedium surrounding protocol analyzing as possible and to provide a user-friendly, effective and powerful environment that can be used by the researchers. The automation of the belief multiset formalism based on the freshness principle could be implemented with the object-oriented development language like C#, C++, Java, etc.; or with artificial intelligence language like Prolog. To determine which is more suitable for the implementation of the belief multiset formalism, under the prerequisite to ensure the correctness of the analysis results, we compare the efficiency, usability of the automation tools developed under the object-oriented development language C# and the Prolog-based implementation language. The initial implementations of these two development environments mainly focus on the visual BMF part, the BMF analyze engine part and the BMF result view part of the BMF analyzer.

BMF analyzer 1[29] is developed using the object-oriented development language C#. The corresponding classes based on the belief multiset formalism include the Principal Class, FreshnessComponent Class, Expectation Class, Fragment Class, Timestamp Class, BeliefMultisetsPresentation class, Key class (AsymmetricKey Class, SymmetricKey Class, SharedKey Class), Belief Class (PrincipalBelief Class, MarkBelief Class, KeyBelief Class, KeyKnown Class), as illustrated in Fig. 9.2.

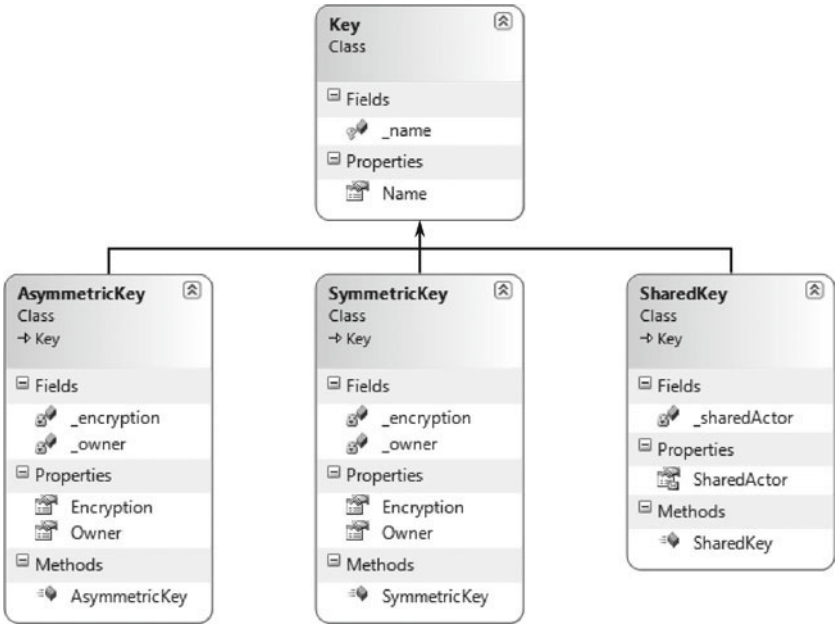


Fig. 9.2 Class illustrations of BMF analyzer 1.

The configuration and the analysis results, which are in the XML format, are stored in the database of facts and the database of goals, and are provided to the user with a completely graphic-based environment for message input and result show, as illustrated in Fig. 9.3.

The BMF analyze engine in the BMF analyzer 1 supports optimized rule search, rule class operations, rule search method triggered by new generated beliefs, remove of plenty of useless intermediate results. Applying the strategy model and abstract factory model to define the database of rules, the belief multiset formalism inference rules could be modeled, and the operations on the rules are independent of the custom applying the database of rules. E.g., for associate class (shared key subclass, shared key with TTP subclass, private key subclass), the operation of the class implements the inference function of rules.

The BMF analyzer 1 is implemented under the object-oriented development language C#. The implementation under the object-oriented develop-

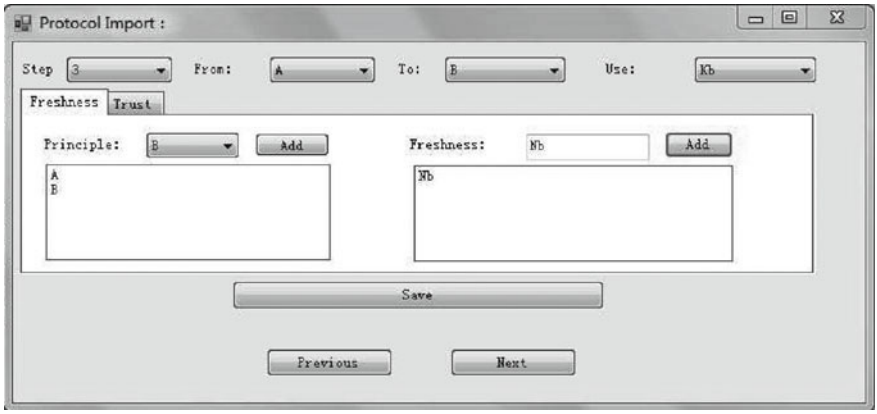


Fig. 9.3 User interface of BMF analyzer 1.

ment environment is model-based, hence it is clear and convenient to develop and debug this analyzer. However, it is difficult to add new classes and to extend the belief multiset formalism rules in the analyzer since they are embedded within the analyzer.

The BMF analyzer 2[30] is implemented based on the artificial intelligence language Prolog that is simple and reliant as an implementation language. The BMF analyzer 2 is similar to the GYNGER in the SPEAR II Framework[6]. The initial BMF analyzer 2 includes three parts: the BMF statement construction interface (visual BMF), a Prolog-based BMF analyze engine (BMFGER) and the result presentation interface (BMF result view). Similar to the GNY analysis engine GYNGER in SPEAR II, the Prolog-based analysis engine BMFGER relies on a forward-chaining inference engine to generate all of the BMF beliefs that can result from the systematic application of the inference rules to a set of initial assumptions and message steps. The visual BMF functions as a user-friendly interface for the input of the protocol, the initial assumptions and the goals. The BMF result view functions as a user-friendly interface for the output of the analysis results of a specific protocol. The initial assumptions goals and analysis results are presented as the belief multiset formalism statements in the Prolog-style, and the belief multiset formalism statements are represented using a tree-like approach. The messages and initial assumptions pertaining to the protocol to be analyzed are specified in the form of fact/3 predicates, and the target goals are specified in the form of goal/2 predicates. The BMF result view converts the analysis results in the Prolog-style into an English-style BMF syntax for every successful goal that was specified, as illustrated in Fig. 9.4.

Note that the rules, the protocol messages in the BMF analyzer 2 are all presented in the Prolog style, hence the database of rules could be stored independently from the BMF analyzer 2, so we could extend the belief multiset rules and add these rules to the BMF analyze engine conveniently in the future, even by a text editor.

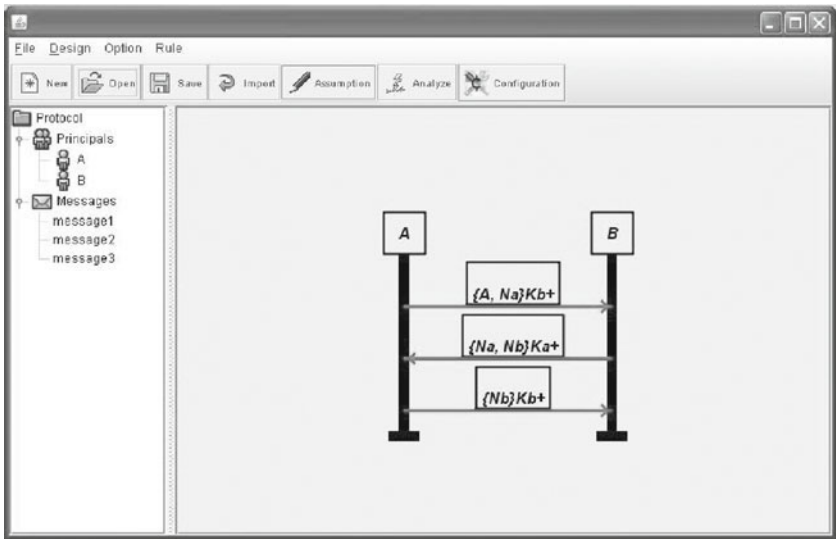


Fig. 9.4 User interface of BMF analyzer 2.

After taking security analysis test of cryptographic protocols under the above two environments, we know that the cryptographic protocol analysis result could be given definitely under both implementations of the BMF analyzers, and the time for analyzing under both cases is short. However, the BMF analyzer 1 which is implemented with embedded belief multiset formalism rules is more difficult to extend than the BMF analyzer 2 which has an independent rule database from the analyzer.

9.2.3 Implementation of the belief multiset formalism

The focus of the BMF analyzer is to provide a user-friendly, effective and powerful environment that can be used to facilitate the analysis of the existing cryptographic protocols and the creation of secure cryptographic protocols based on the notion of the trusted freshness.

From the comparison in Subsection 2, we decide to develop the BMF analyzer mainly on the BMF analyzer 2, that is, using Prolog to develop the BMF analyze engine and using Java source code generation to develop the visual BMF and the BMF result view.

Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics, and it remains among the most popular such languages today, with many free and commercial implementations available. The Prolog program logic is expressed in terms of relations, represented as facts and rules, and a computation is initiated by running a query over these relations. Prolog is well suited for implementing a

BMF analyzer as it is straight forward to map all of the BMF constructs into suitable Prolog counterparts which can be easily manipulated and queried.

The BMF analyzer based on Prolog helps the protocol engineers to derive all possible BMF statements applicable to a given protocol and to determine whether a given protocol achieves its design objectives.

Java is an object-orientated technique that is employed in order to facilitate expansion and understanding of the source code. It is clear and convenient to develop and debug user-friendly interfaces like the visual BMF and the BMF result view.

The BMF analyzer is a graphically based application that incorporates cryptographic protocol modeling, automated BMF-based protocol security analysis and the analysis results presentation. The BMF analyzer is implemented similarly to the SPEAR II Framework except the analysis logic is the belief multiset formalism but GNY, and it incorporates a number of enhancements. Because of the architectural and security analysis logic differences between the SPEAR II and the BMF analyzer, the Java code generation and scenario simulation features are not the same in the BMF analyzer implementation.

To conduct an analysis with the BMF analyzer, a protocol engineer needs to specify a protocol's messages, initial assumptions and target goals in a Prolog-style BMF syntax via the visual BMF interface. The Prolog-based BMF rules are then imported and employed in the analysis, where a forward-chaining inference engine generates all of the BMF beliefs that can result from the systematic application of the inference rules to the set of initial assumptions, message steps and the cryptographic goals. A proof is then generated in an English-style syntax for every successful goal that was specified. This English-style proof lists all of the statements involved in the derivation of the successful goal, indicating the postulates that were used and the premises which were employed in the postulate's application.

The BMF analyzer is extensible, it allows extended security analysis of a protocol such as non-repudiation analysis for future use, and it also allows further engineering and design techniques to be incorporated, since the inference rules are stored separately from the BMF analyzer itself.

For interest of concision, some atoms and the predicates in the BMF analyzer are only briefly introduced. For strict or inquisitive readers, please refer to [5, 6] or contact us for detailed information.

9.2.3.1 Facts, goals and rules

In belief multiset formalism, the relations in Prolog include: assumptions, terms, medium results, goals and rules. These important concepts are represented as Prolog-style predications, and stored in the database of facts, the database of goals and the database of rules respectively.

The Prolog-style predications are formulae, statements, etc. Formulae are the components which are used to construct protocol messages and typically contain constants such as principal names, nonce, shared keys, shared parts

of keys, etc. The statements will be used to specify the initial and target beliefs and possessions of principals as well as extensions to be appended to the message components.

Facts

The idealized protocol messages, initial assumptions, terms and medium results are represented as *fact/3*, and are all stored as instances of fact/3 in the database of facts. Before conducting an automated BMF-based analysis, the messages and initial assumptions pertaining to the protocol to be analyzed must be specified in a Prolog file in the form of fact/3 predicates.

The predicate fact/3, which defines an inference step, appears as follows:

$$\text{fact}(\textit{Index}, \textit{Statement}, \textit{Reason}(\textit{PremiseList}, \textit{Rule}))$$

The integer *Index* is used to reference instances of this fact/3, while the argument *Statement* is bound to a derived statement, including the idealized protocol messages, initial assumptions, terms and medium results. The last argument *Reason(PremiseList*, *Rule*) is the reason to derive this fact/3, the parameter *PremiseList* is a list containing the indices of the premises that were used in deriving this *Statement* through the application of *Rule*, and the parameter *Rule* represented by characters enclosed in single quotes is the applied rule to derive this fact/3. If the statement represents terms or initial assumptions, the *PremiseList* would be empty and *Rule* would be either "*Term*" or "*Assumption*". For example, fact(*Index*, *Statement*, Reason([], "*Term*")) and fact(*Index*, *Statement*, Reason([], "*Assumption*")).

There are two important predicates for representing "send" and "receive" terms:

$$\text{send}(\textit{Identity}, \textit{Statement}, \textit{Step})$$

and

$$\text{receive}(\textit{Identity}, \textit{Statement}, \textit{Step})$$

which means that the principal '*Identity*' has sent or received the term '*Statement*' in this step '*Step*' respectively, where the integer '*Step*' is used to reference the time point to send or receive this term.

New added fact/3 involved in the analysis process must be extracted and sorted in ascending order by their indices, and any duplicates in this list must also be removed. The predicate *getMaxFactIndex/1* collects all of the indices within fact/3 into a list and then finds the maximum in this list, and the maximum index is returned in the argument *MaxIndex*.

Goals

In order to perform a protocol security analysis based on the belief multiset formalism, a designer must know what goals the protocol under inspection is expected to achieve. The security analysis will essentially involve a researcher determining the class of the protocol that he wishes to analyze, and then ensuring that the expected goals are fulfilled. For an authentication

protocol, one should verify the identities of participants and then ensures that they agree on an encryption key for later use.

The protocol goals are represented in a similar way to fact/3 with the *goal/2* predicate and stored in the database of goals. The goal/2 predicate appears as follows:

$$\text{goal}(\textit{Index}, \textit{Statement})$$

The integer *Index* is used to reference instances of this goal/2, and the argument *Statement* is bound to the anticipated security objectives (the security properties in the belief multisets, including the beliefs about principals and the beliefs about a freshness identifier). A belief is presented as the predicate:

$$\textit{Belief}\,(\textit{Identity}, \textit{Trust})$$

it means that the principal *Identity* believes the *Trust*. Any target goals must be specified in the same file by using goal/2 predicates.

Rules

The BMF inference rules are all specified through the use of a *rules/0* predicate and stored in the database of rules. For each BMF rule, there is at least one instance of the rules/0 predicates, some requiring more because of multiple conclusions. The basic pattern followed in a typical instance of the rules/0 predicate is to first check that all of the premises of the respective BMF rule are true and then to assert the conclusion in the Prolog database if it has not yet been asserted. After asserting the conclusion, the *addedFacts* atom is also asserted in the database to indicate that a conclusion was derived during the current cycle. If addedFacts has been asserted, then all the remaining instances are retracted from the database, and the *done/0* predicate fails. Otherwise, if addedFacts was not asserted then done/0 would succeed and forward-chaining would not commence. The done/0 predicate checks whether any new beliefs or possessions have been added to the Prolog database in the current cycle.

Example 9.1 Recall that the fragment rule A1(a) in the belief multiset formalism is:

$$\textbf{A1(a)}\ -\{...N, N'...\}_{K_{P_iP_j}} \wedge B_{P_i,t}(\langle 11K_{P_iP_j}P_iP_j\rangle) \wedge B_{P_i,t}(\langle ...1N...\rangle)$$
$$\Rightarrow B_{P_i,t}(\sim \{...N, N'...\}_{K_{P_iP_j}})$$

The code for the rules/0 predicate of the *Fragment Rule 1* in the BMF analyzer appears below:

```
rules :-
    fact(PremiseIndex1, told(P,encrypt(List, shared(K))),_),
    is_list(List),
    length(List, LengthOfList),
    LengthOfList > 1,
    fact(PremiseIndex2, believes(P, secret(shared(K))), _),
    fact(PremiseIndex3, believes(P,fresh(shared(K))), _),
    fact(PremiseIndex4, believes(P, associate(shared(K),P, Q)), _),
```

```
listMemberIsFresh(P, List, PremiseIndex5),
Conclusion = believes(P, bound(List)),
not(fact(_,Conclusion,_)),
getMaxFactIndex(MaxIndex),NewIndex is MaxIndex + 1,
PremiseIndices = [PremiseIndex1, PremiseIndex2, PremiseIndex3,
                  PremiseIndex4, PremiseIndex5],
asserta(fact(NewIndex,Conclusion,reason(PremiseIndices,'Fragment1'))),
asserta(addedFacts).
```

9.2.3.2 Visual BMF

The visual BMF provides graphical interface for the import of the protocol messages, initial assumptions, and security goals.

The protocol messages

Similar to SPEAR II, the visual BMF environment represents BMF statements using a tree-like structure combined with pop-up menus to allow easy interaction and to produce a meaningful representation of BMF information.

All statements of the same type form part of the same tree structure, a heterogeneous set of BMF statements are represented by a collection of separate trees. These representation techniques help users to easily create syntactically correct BMF statements without the need to be acquainted with the BMF syntax and notation. These structured trees representing the protocol messages could be exported and read as English-style text.

With a graphical interface, users have to remember few details about a system's structure and functionality, since this information is available within the interface. Furthermore, to use the visual BMF environment, users do not need to be familiar with the semantics and concepts underlying the belief multiset logic, however, they'd better be familiar with the logic to use it effectively.

The protocol's messages are specified in an optimum order as follows: the principals involved in the protocol, the role selection of each principal (sender, receiver and the trusted freshness), the cryptographic mechanism in the protocol, the actual communication between the principals etc. Once the protocol specification is complete, the protocol will be translated into graphical display as shown in Fig. 9.5.

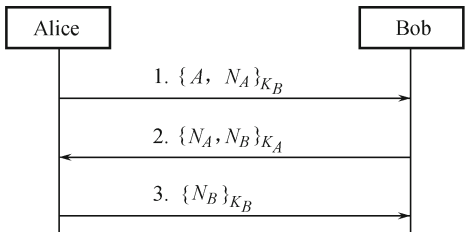


Fig. 9.5 View of a cryptographic protocol in the BMF analyzer.

Here, Alice means the principal A, Bob means the principal B.

The initial assumptions

The initial assumptions, also called the initial BMF beliefs, should be declared at the start of the protocol analysis, and any amendments could be made at a later stage. If a principal has generated a freshness identifier used in the protocol messages, the belief about the freshness of this identifier by the generator is also an initial assumption.

Once the protocol specification is complete, an analyzer can definitely make recommendations as to what the initial conditions for an analysis should be in certain cases, that is, the initial assumptions could be derived directly from the import protocol messages, as shown in Fig. 9.6.

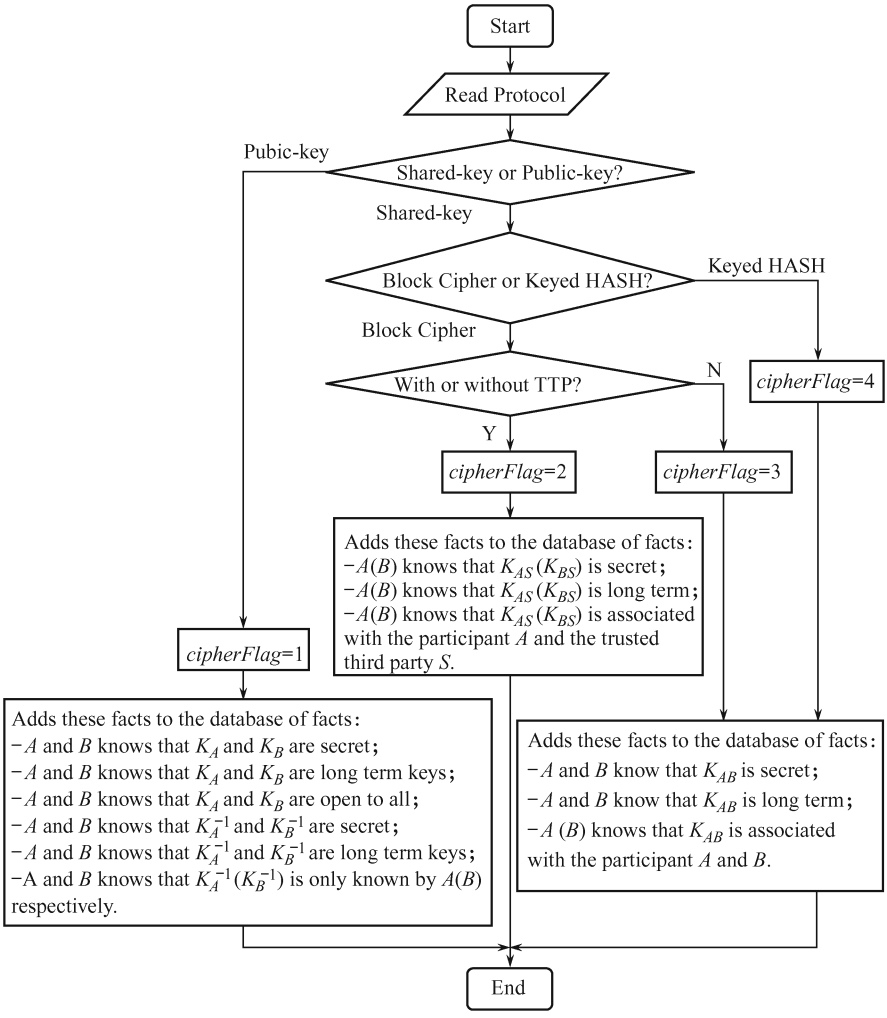


Fig. 9.6 The premise set of BMF analysis.

The *cipherFlag* is the flag of the cryptographic mechanism applied in this protocol:

— for *cipherFlag*=1, the cryptographic mechanism applied is Public-key scheme;
— for *cipherFlag*=2, the cryptographic mechanism applied is shared-key scheme with the trusted third party;
— for *cipherFlag*=3, the cryptographic mechanism applied is shared-key scheme without the trusted third party;
— for *cipherFlag*=3, the cryptographic mechanism applied is keyed hash scheme.

The extraction of terms

In general, the extraction of terms could be done meanwhile with the protocol message specification. If there exist nests of the cryptographic one-way transformations, the condition of repetition of the terms should be prudently considered. If the maximum terms of different messages are the same, there may exist the replay attack, hence the designed protocol should be reconstructed, as shown in Fig. 9.7 for the extraction of the terms, where *maxStep* is the largest number of the protocol message steps.

The security goals

The goal import interface supports the security goal configuration of an input protocol, such as UA-secure, MA-secure, UK-secure and MK-secure. The target goals are specified in the form of goal/2 predicates, that is *Belief*(*Identity*, *Trust*). The *Trust* beliefs are predicates about the security properties, including *Existing*(*Identity*), *Secret*(*Identifier*), *Fresh*(*Identifier*) and *Associate*(*Identifier, Identity*). The predicate *Existing*(*Identity*) means that a principal has the trust about the lively communication of this *Identity* principal. The predicate *Secret*(*Identifier*) means that a principal has the trust about the security of this *Identifier*. The predicate *Fresh*(*Identifier*) means that a principal has the trust that the freshness *Identifier* is a new generated TVP for this protocol. The predicate *Associate*(*Identifier, Identity*) means that a principal has the trust that the freshness *Identifier* is a TVP for a protocol related with the principal *Identity*.

Example 9.2 Here is an illustration of the security goals of a protocol in the BMF analyzer. Suppose the two communication principals are A and B, the new session key they want to establish is k_{AB}.

As for a UA-secure authentication protocol, if A wants to authenticate the principal B, then A has the security goal to achieve: *Belief*(A, *Existing*(B)).

As for an MA-secure authentication protocol, if A and B want to authenticate each other, the security goals to achieve are *Belief*(A, *Existing*(B)) and *Belief*(B, *Existing*(A)).

As for a UK-secure authentication protocol, the security goals to achieve are *Belief*(A, *Existing*(B)), *Belief*(A, *Secret*(k_{AB})), *Belief*(A, *Fresh*(k_{AB})), *Belief*(A, Associate(k_{AB}, A)) and *Belief*(A, *Associate*(k_{AB}, B)).

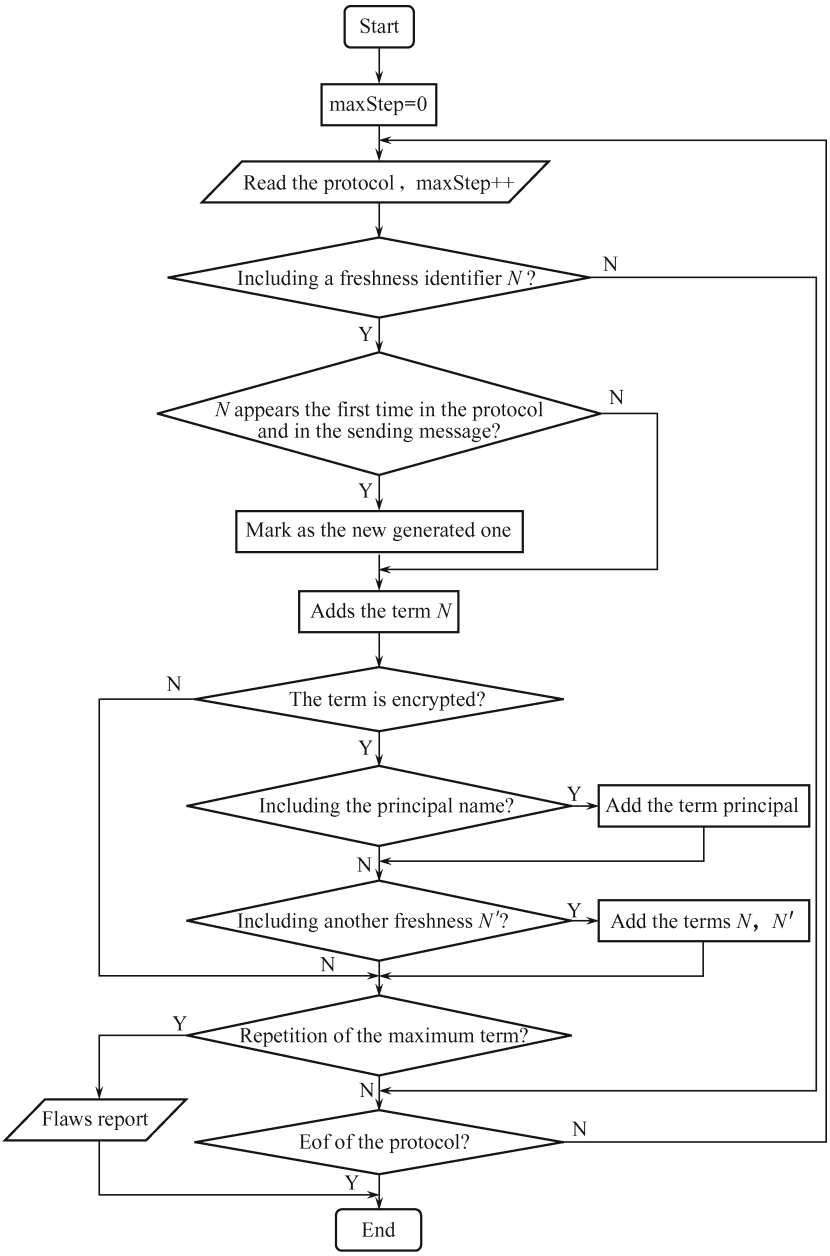


Fig. 9.7 Abstraction of BMF term generation.

As for an MK-secure authentication protocol, the security goals to achieve for the principal A are $Belief(A, Existing(B))$, $Belief(A, Secret(k_{AB}))$, $Belief(A, Fresh(k_{AB}))$, $Belief(A, Associate(k_{AB}, A))$ and $Belief(A, Associate$

(k_{AB}, B)); the security goals to achieve for the principal B are *Belief*(B, *Existing*(B)), *Belief*(B, *Secret*(k_{AB})), *Belief*(B, *Fresh*(k_{AB})), *Belief*(B, *Associate*(k_{AB}, A)) and *Belief*(B, *Associate*(k_{AB}, B)).

9.2.3.3 BMF analyze engine

Principals, messages, terms and initial assumptions specified in visual BMF are exported to BMF analyze engine in Prolog-style statements for analysis, and all of the BMF statements derived during an analysis, as well as the proofs for successful goals, are stored in the database of facts and are accessible through the result view pane.

Before conducting an automated BMF-based analysis, the messages, terms and initial assumptions pertaining to the protocol to be analyzed have been specified in the form of fact/3 predicates in the database of facts. Any target goals have also been specified in the form of goal/2 predicates in the database of goals.

Similar to the forward-chaining inference engine in SPEAR II, the BMF analyze engine applies all the inference rules to the set of statements consisting of the protocol messages, initial assumptions and medium results, until all of the statements which are derivable have been generated. If there exist new generated fact/2 predicates in the database of facts, the addedFacts atom is also asserted in the database to indicate that a conclusion was derived during the current cycle. If addedFacts has been asserted, the BMF analyze engine applies all the inference rules to the set of the term statements in this step, until there are not any new generated fact/3 predicates inserted into the database of facts. Then the analysis results are compared to determine whether one or more statements describing the goals of a specific protocol are derivable from a given set of initial assumptions. If the security goals are met, a proof can be generated for this security goal in the database, showing all of the steps and inference rules that were required to generate the result. Results from a BMF analysis conducted by the BMF analyze engine are returned to the BMF result view environment and appropriately displayed in English-style BMF syntax. A formal proof will then be constructed to show that a finite number of conclusions in a finite number of steps can be derived from using the inference rules based on the initial assumptions and messages of a given protocol.

Figure 9.8 illustrates the analysis procedure of the BMF analyze engine.

In Chapter 4, we have illustrated the security analysis procedure using the belief multiset formalism. As we have seen, the useful deriving statements, medium results, are developed through the application of the belief multiset inference rules manually, so the analysis procedure is relatively simple. While in the case of automated analysis, the analyzer will generate a lot of medium results, including not only the useful facts but also many needless medium results. The key problem to improve the performance of the analyzer is to discard the needless medium results effectively.

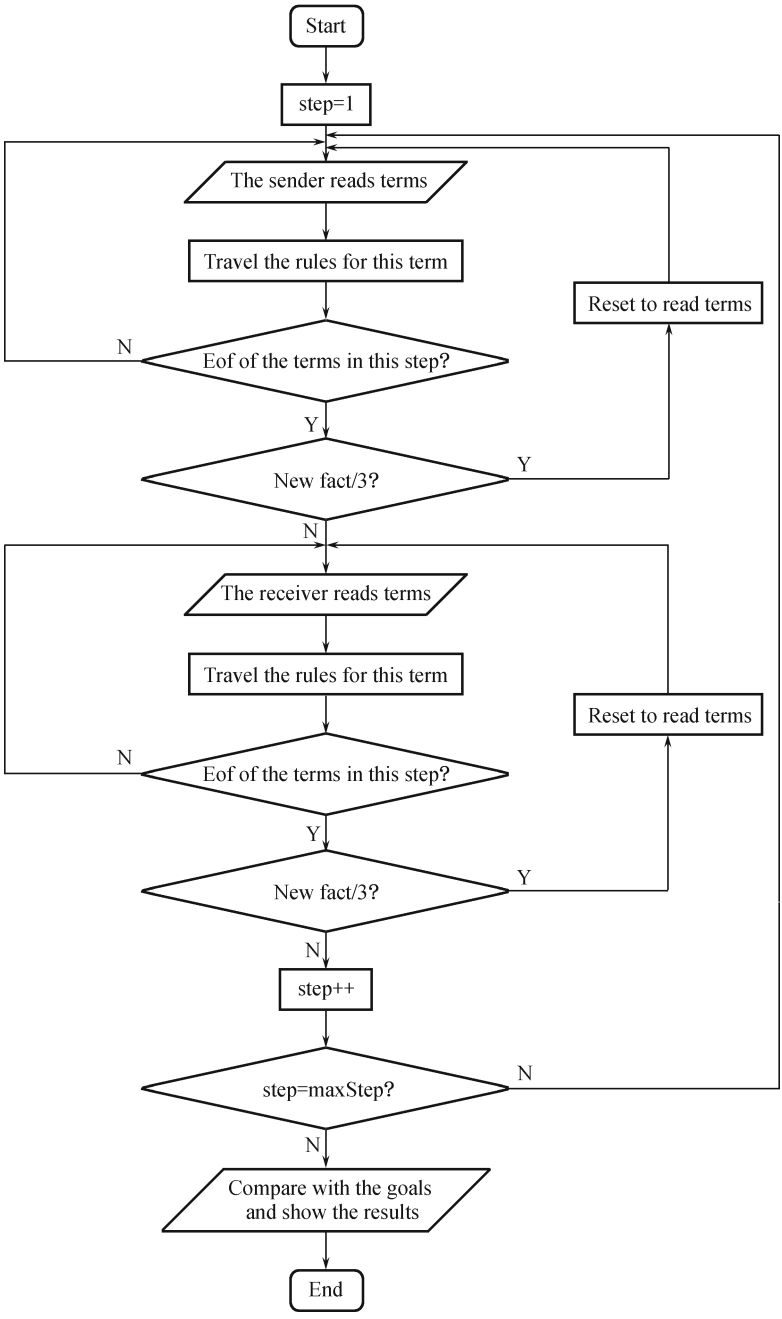


Fig. 9.8 BMF analyze engine.

9.2.3.4 BMF result view

The BMF result view provides a graphical result view environment, including protocol security analysis results showing the failed and successful beliefs in English-style BMF syntax, and the English-style proof list of a successful protocol goal.

The BMF result view ensures that an analysis statement representing in a Prolog-style formula is converted to an appropriate textual representation. The fact that the proof is in an English-style syntax makes it more readable and comprehensible.

This English-style proof lists all of the statements involved in the derivation of the successful goal, indicating the postulates that are used and the premises which are employed in the postulates' application. If a goal fails, a proof cannot be generated, and then the text 'FAILED!' appears instead of a proof.

9.2.3.5 BMF rule engine

The BMF rule engine provides a graphical environment for the new added belief multiset formalism inference rule input. As we know, the belief multiset rules presented in Chapter 4 are for authentication protocols which apply traditional cryptographic mechanisms, the researchers need to extend the inference rules used in the BMF analyze engine to meet variety applications in the real world.

The BMF rule engine includes the following steps to construct a new inference rule in the BMF analyzer:

1) Give the general rules/0 representation of the rule to be inserted.

```
rules   :-
        fact(PremiseIndex1, Statement1ToBeInserted, _),
        fact(PremiseIndex2, Statement2ToBeInserted, _),
         ...  ...
        Conclusion =ConclusionToBeInserted
        not(fact(_, Conclusion, _)),
        getMaxFactIndex(MaxIndex), NewIndex is MaxIndex + 1,
        PremiseIndices = [PremiseIndex1, PremiseIndex2, ... ],
        asserta(fact(NewIndex, Conclusion, reason(PremiseIndices,
                    'RuleNameToBeInserted'))),
        asserta(addedFacts).
```

2) Convert the new inference rule in the belief multiset formalism into the inference rule of the BMF analyzer in Prolog-style.

Replace the *Statement1ToBeInserted*, *Statement2ToBeInserted*, etc. with

the conditions of the new inference rule in the belief multiset formalism; replace the *ConclusionToBeInserted* with the conclusions of the new inference rule in the belief multiset formalism; replace the *RuleNameToBeInserted* with the new inference rule name.

3) Show the new inference rule to the rule design researcher, and allow him to improve this rule until the researchers submit this rule to the system.

To ensure that the new generated rule produces the correct results for different inputs, the rule should be tested individually by specifying all of the rule premises using fact/3 predicates, running the analyzer, and then examining the results.

Recall that the belief multiset inference rules are in the style of Prolog which is a programming language associated with artificial intelligence, and the rule database is stored independently from the BMF analyzer and it is dynamically loaded when the security analysis is ongoing. Hence, it is convenient to edit the inference rules outside the system, and the changing could be applied immediately. That is, besides the insertion of the new inference rule in the graphical environment of the BMF analyzer, the inference rules could even be edited, deleted and inserted in a text editor such as the Notepad in the Windows system. Therefore, the flexibility of the BMF analyzer has been greatly improved.

9.2.3.6 BMF attack engine

The attack engine supports the construction of attacks from the absence of the security properties derived from the security analysis results based on the belief multiset formalism. As we all know, the automation construction of an attack requires quite a high level of user expertise in a large number of specialized engineering techniques, while the manually construction of the attack may seem relatively simple. Here we only give a naive attack engine model for attack construction based on the trusted freshness approach, as shown in Fig. 9.9.

1) Indicate the principal to be deceived from the absence of the security properties of this protocol being analyzed.

2) Construct the first message to cheat the principal whose security properties about this protocol are not met.

3) Complement other messages to form an instance of the protocol run with full messages.

4) Find the messages that couldn't be constructed in this instance. If any, continue Step 5; if not, terminate this protocol construct procedure, and this protocol instance is the attack that we want to construct on this flawed protocol.

5) Start another instance of this flawed protocol in order to generate the key messages that couldn't be constructed from the instance in Step 4. Complement other messages to form this interleaved instance with full messages. Thus, these two interleaved instances, that are the instance in Step 4 and the instance in Step 5, construct the attack on this flawed protocol.

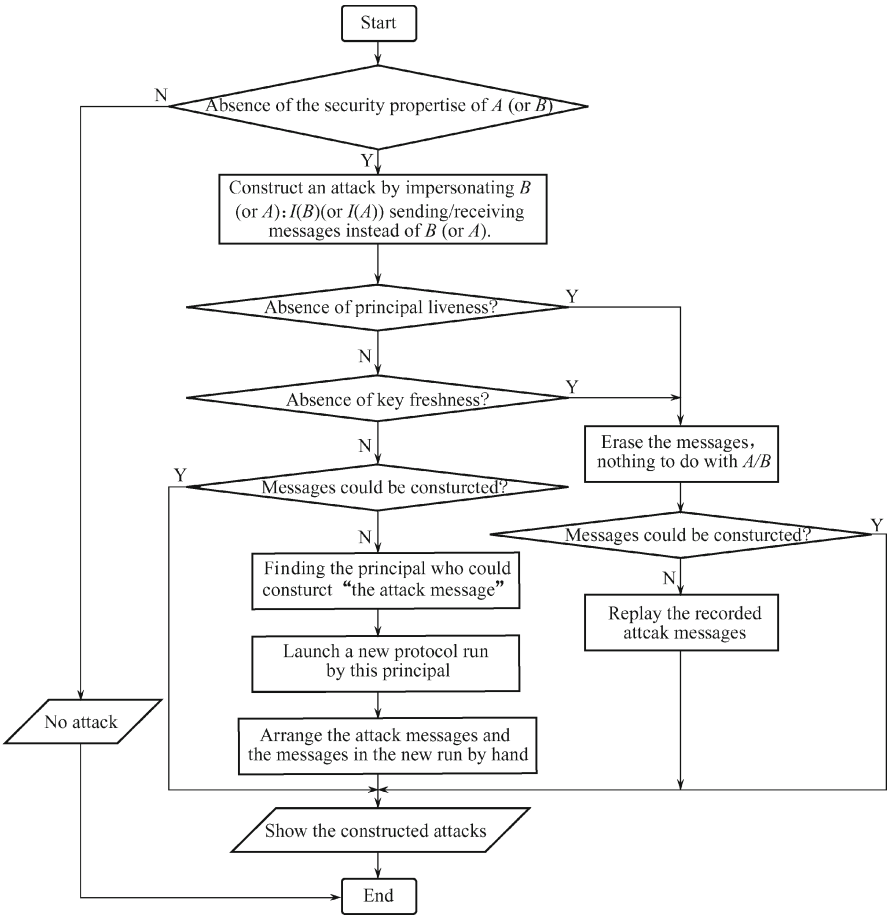


Fig. 9.9 BMF attack engine.

6) Arrange the order of the messages in the two interleaved instances, and then terminate this protocol construct procedure.

Let's review the Needham-Schroeder public-key protocol in Example 1.2. From the security analysis of the Needham-Schroeder protocol using the belief multiset formalism in Subsection 5, A believes that B is in lively correspondence in this protocol run, and the shared parts of both N_A and N_B are secret, fresh, and also associated with the principal A and the principal B. However, although B believes that A is in lively correspondence in this protocol run, N_B is secret, fresh, and associated with the principal A and the principal B, but B has not gotten any corroborative evidence that N_A is fresh and is associated with the principal A and the principal B.

Example 9.3 Here is an illustration of the attack construction procedure of the Needham-Schroeder public-key protocol in the BMF attack engine.

Here, Alice means the principal A, Bob means the principal B, while Malice means the adversary I.

1) It is the principal B whose security properties are absent, we need to construct an attack to cheat B.

2) Construct the first message Message 1 to cheat B.

Message 1 $\quad I(A) \rightarrow B: \quad \{A, N_A\}_{K_B}$

3) Complement Message 2 and Message 3 to form an instance of Needham-Schroeder public-key protocol with full messages.

Message 1 $\quad I(A) \rightarrow B: \quad \{A, N_A\}_{K_B}$

Message 2 $\quad B \rightarrow I(A): \quad \{N_A, N_B\}_{K_A}$

Message 3 $\quad I(A) \rightarrow B: \quad \{N_B\}_{K_B}$

4) Message 3 couldn't be constructed in the above instance since the adversary I impersonating A namely $I(A)$ could not get the freshness identifier N_B.

If the adversary I wants to generate the message $\{N_B\}_{K_B}$, I must know N_B. Since N_B appears only in Message 2 and Message 3, while Message 2 is a one-way transformation sent from the victim B to A and I does not have the knowledge of the decryption key K_A^{-1}, hence the adversary I could not get N_B from Message 2, hence I could only get N_B from Message 3. Note that Message 3 is encrypted under the receiver's public-key, so the adversary I must be the receiver to perform a one-way transformation in order to get N_B from Message 3 using I's private key K_I^{-1}. Hence, there should exists:

Message $3'$ $\quad ?? \rightarrow I: \quad \{N_B\}_{K_I}$

5) Since N_B appears only in Message 2 which is a one-way transformation sent from the victim B to A, only A could get N_B from Message 2, so the Message $3'$ in Step 4 could only be exchanged between the principal A and the adversary I, that is:

Message $3'$ $\quad A \rightarrow I: \quad \{N_B\}_{K_I}$

Hence, the new instance to generate the key Message $3'$ is between A and I, and this idea is consistent with the security property that A is in lively correspondence in this protocol run.

Complement Message $1'$ and Message $2'$ to form this new instance:

Message $1'$ $\quad A \rightarrow I: \quad \{A, ??\}_{K_I}$

Message $2'$ $\quad I \rightarrow A: \quad \{??, N_B\}_{K_A}$

Message $3'$ $\quad A \rightarrow I: \quad \{N_B\}_{K_B}$

If the adversary I wants to generate Message $3'$, then I should know N_B, since N_B is encrypted under A's public-key, I could only replay the recorded message Message 2 $\{N_A, N_B\}_{K_A}$ including $\{??, N_B\}_{K_A}$ in order to get N_B, that is:

Message $2'$ $\quad I \rightarrow A: \quad \{N_A, N_B\}_{K_A}$

To make the principal A believe that Message $2'$ is really from I, then the unknown "??" in Message $1'$ could only be the freshness identifier N_A that

is the same as that in Message 2′. Hence we have:

Message 1′ $A \rightarrow I:$ $\{A, N_A\}_{K_I}$

6) From the above construction procedure, we have the first instance

Message 1 $I(A) \rightarrow B:$ $\{A, N_A\}_{K_B}$

Message 2 $B \rightarrow I(A):$ $\{N_A, N_B\}_{K_A}$

Message 3 $I(A) \rightarrow B:$ $\{N_B\}_{K_B}$

and the second instance

Message 1′ $A \rightarrow I:$ $\{A, N_A\}_{K_I}$

Message 2′ $I \rightarrow A:$ $\{N_A, N_B\}_{K_A}$

Message 3′ $A \rightarrow I:$ $\{N_B\}_{K_B}$

Arrange the order of the messages in the two interleaved instances, and then we have the attack on the Needham-Schroeder public-key protocol, as shown in Fig. 9.10.

Message 1′ $A \rightarrow I:$ $\{A, N_A\}_{K_I}$

Message 1 $I(A) \rightarrow B:$ $\{A, N_A\}_{K_B}$

Message 2 $B \rightarrow I(A):$ $\{N_A, N_B\}_{K_A}$

Message 2′ $I \rightarrow A:$ $\{N_A, N_B\}_{K_A}$

Message 3′ $A \rightarrow I:$ $\{N_B\}_{K_I}$

Message 3 $I(A) \rightarrow B:$ $\{N_B\}_{K_B}$

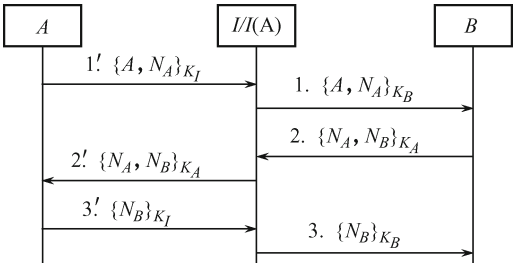


Fig. 9.10 View of attacks in BMF analyzer.

The attack involves two simultaneous runs of the Needham-Schroeder public-key protocol. In the first run, A establishes a valid session with the adversary I; in the second run, I impersonating A tries to establish a bogus session with B. Upon the termination of this attack, B believes that B has correctly established a session with A and they shared exclusively the secret nonce N_A and N_B to generate the new session key.

As we have seen, the attack construction procedure is complex, and the formalization of the intelligence activities in the attack construction is difficult. The BMF attack engine presented is still naive, and a lot of jobs need to be done.

In the above chapters, we have presented security definition, security specifications, freshness principle, manual analysis method, belief multiset formalism and the automation tools based on the trusted freshness for analyzing cryptographic protocols, which clarify whether a cryptographic protocol is secure or not.

The central ingredient in the trusted freshness approach is the observation that a participant's beliefs about key exchange security should depend only on the received fresh and confidential messages and the beliefs already possessed by this party. Analysis based on trusted freshness captures exact authentication information of each principal, which suggests the correctness of a protocol or the way to construct attacks intuitively from the absence of security properties.

First, Chapter 4 presents the security definitions, the security specifications based on the indistinguishability approach and matching conversation, which check whether a cryptographic protocol is secure or not; Chapter 6 makes a more rigorous proof to assess that the indicated security specifications are not only necessary but also substantial under the computational model. This specific security adequacy captures the peculiarities of key exchange protocols that involve different sessions. Chapter 7 presents a belief multiset formalism for analysis of cryptographic protocols based on trusted freshness.

Chapter 4, Chapter 5 and Chapter 7 have exemplified the usability and the efficiency of the security specifications to guarantee the protocol security and the belief multiset formalism via a set of well-known protocols. The absence of certain security properties suggests the instant construction of many attacks (not only one) on the protocol or suggests the correction of the protocol. For example, in Kerberos pair-key protocol in DSNs (see Subsection 5), B could not guarantee the freshness of k_{AB} and the liveness of sensor node A, so the adversary can construct an attack by impersonating A and confuse B to regard an old key k'_{AB} as a new session key between B and A. From the absence of the association k_{AB} with A and B, the adversary can construct an attack and confuse B to believe that B shares a new session key k_{AB} with A, but in deed B shares k_{AB} with the attacker I.

The proofs of security based on trusted freshness are simple and precise, which can be easily accomplished not only by hand (Chapter 4 and Chapter 5) but also by formalism (Chapter 6 and Chapter 7). Moreover, the analysis process based on trusted freshness is rigorous and amenable for design (Chapter 8) and automation (Chapter 9).

References

[1] Dong L, Chen K, Zheng Y, Hong X (2008) The Guarantee of Authentication Protocol Security. Journal of Shanghai JiaoTong University, 42(4): 518 – 522 (in Chinese)

[2] Chen K, Dong L, Lai X (2008) Security Analysis of Cryptographic Protocols Based on Trusted Freshness. Journal of Korea Institute of Information Security and Cryptology, 18(6B): 1 – 13

[3] Dong L, Chen K, Lai X (2009) Belief Multisets for Cryptographic Protocol Analysis. Journal of Software, 20(11): 3060 – 3076 (in Chinese)

[4] Dong L, Chen K, Lai X, Wen M (2009) When is a Key Establishment Protocol Correct? Security and Communication Networks, 2(6): 567 – 579

[5] Saul E (2001) Facilitating the Modelling and Automated Analysis of Cryptographic Protocols. Master Thesis, University of Cape Town

[6] Saul E and Hutchison A (1999) SPEAR II: The Security Protocol Engineering and Analysis Resource. Second South African Telecommunications, Networks, and Applications Conference. http://pubs.cs.uct.ac.za/archive/00000128/01/saul1999_SPEAR_SATNAC.pdf. Accessed 5 May 2011

[7] Mao W (2004) Modern Cryptography: Theory and Practice. Prentice Hall, New Jersey

[8] Bekmann J, Goede P. de and Hutchison A (1997) SPEAR: a Security Protocol Engineering & Analysis Resource. In DIMACS Workshop on Design and Formal Verification of Security Protocols. http://dimacs.rutgers.edu/Workshops/Security/program2/hutch/spear.html. Accessed 5 May 2011

[9] Burrows M, Abadi M, and Needham R (1990) A logic of authentication. ACM Transactions on Computer Systems, 8(1): 18 – 36

[10] Lowe G (1996) Breaking and Fixing the Needham-Schroeder Public-key Protocol Using FDR. In: TACAS'96 Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Passau, 27 – 29 Mar 1996. Lecture Notes in Computer Science (Lecture Notes in Software Configuration Management), vol 1055. Springer-Verlag, Heidelberg, pp 147 – 166

[11] Lowe G (1995) An Attack on the Needham-Schroeder Public Key Authentication Protocol. Information Processing Letters, 56(3): 131 – 133, 1995

[12] Lowe G (1996) Some new Attacks Upon Security Protocols. In Proceedings of the 9th IEEE Computer Security Foundations Workshop, pages 162 – 169, Jun. 1996

[13] Roscoe A W (1994) Model Checking CSP. In: Roscoe A W (ed) A Clasical Mind: Essays in Honour of C.A.R Hoare. Prentice-Hall, 1994

[14] Meadows C (1996) The NRL Protocol Analyzer: an Overview. Journal of Logic Programming, 26(2): 113 – 131

[15] Dolev D, Yao AC (1983) On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2): 198 – 208

[16] Meadows C (1994) A Model of Computation for the NRL Protocol Analyzer. In: Proceedings of the 1994 Computer Security Foundations Workshop, Franconia, 14 – 16 June 1994

[17] Harkins D and Carrel D (1998) The Internet Key Exchange Protocol (IKE). IETF RFC 2409, Available at http://www.ietf.org/rfc/rfc2409.txt. Accessed 5 May 2011

[18] Kaufman C (2005) Internet Key Exchange (IKEv2) Protocol. IETF RFC 4306, Available at http://tools.ietf.org/html/rfc4306. Accessed 5 May 2011

[19] SET (1997) Secure Electronic Transaction. The SET Standard Specification. http://www.setco.org/set-specifications. Accessed 5 May 2011

[20] Mitchell J.C, Mitchell M and Stern U (1997) Automated Analysis of Cryptographic Protocols Using MurΦ. Proc. of 1997 IEEE Symposium on Security and Privacy, Oakland, California, pp 141 – 153

[21] Needham RM, Schroeder MD (1978) Using Encryption for Authentication in Large Network of Computers. Communication of the ACM 21(12): 993 – 999

[22] Tatebayashi M, Matsuzaki N, Newman D (1990) Key Distribution Protocol for Digital Mobile Communication Systems, CRYPTO'89, LNCS435

[23] Neuman C, Ts'o T (1994) Kerberos: An Authentication Service for Computer Networks, IEEE Communications Magazine, 32(9): 33 – 38

[24] Millen JK, Clark SC, Freedman SB (1987) The Interrogator: Protocol Security Analysis. IEEE Trans. Software Eng. 13(2): 274 – 288

[25] Berezin S, Groce A. SyMP: Symbolic Model Prover. Model Checking@CMU. http://www.cs.cmu.edu/ modelcheck/symp.html. Accessed 5 May 2011

[26] Fabrega FJT, Herzog JC, Guttman JD (1998) Strand Spaces: Why is a Security Protocol Correct? In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, 3 – 6 May 1998

[27] Song D (1999) Athena: A New Efficient Automatic Checker for Security Protocol Analysis. Proceedings of the 1999 IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, pp 192 – 202

[28] Millen J K (1996) CAPSLS: Common Authentication Protocol Specification Language. http://www.csl.sri.com/users/millen/capsl. Accessed 5 May 2011

[29] Wei MQ (2008) Automated Verification of Security Protocol. BE Thesis (in Chinese), Shanghai Jiaotong University

[30] Wang EJ (2008) Research of Automated Verification Tool for Security Protocol. BE Thesis (in Chinese), Shanghai Jiaotong University

Index

S

T

U

W