7

Identity-Based Key Agreement

7.1 Introduction

Identity-based public key cryptography was first proposed by Shamir in 1984 [665]. The idea is to avoid the need for public key certificates by making the public key publicly computable from the identification information of the owner. The identification information can include any desired fields such as real name, physical description or identification numbers. Identity-based cryptography avoids the difficulty of having to distribute public keys and thus avoids the need for a public key infrastructure, although parties still need to obtain and manage private keys.

In 1984 Shamir proposed an algorithm for identity-based signatures but was unable to obtain an identity-based encryption algorithm. In 1987 Okamoto [589, 590] published the first identity-based key agreement protocol, using the same format of key pairs as in Shamir's original identity-based signatures. For over a decade there was limited activity in the area of identity-based cryptography, until in 2000 the first practical identity-based encryption schemes were proposed [123, 647]. These schemes exploit the bilinearity property of elliptic curve pairings. Following this discovery, there was an explosion of interest in identity-based cryptography based on pairings. This included a variety of different cryptographic primitives and protocols, including scores of key agreement protocols.

There are many similarities between identity-based key agreement and key agreement using standard public key cryptography. Indeed, many key exchange protocols use generic building blocks, such as encryption or signatures, and can be instantiated with either identity-based or conventional public key versions of these building blocks. Essentially, the aim in designing a good identity-based key agreement protocol is to achieve all the properties of the best conventional key agreement protocols but without the need for certified public keys, and at the same time trying to maximise efficiency.

We continue to use the notation ID_I to denote the identifying string of entity I. In many identity-based protocols it is not acceptable for an arbitrary string, which may be chosen by the adversary, to be used directly as input to the private key generation process. This can allow construction of new private keys corresponding to algebraic

C. Boyd et al., *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, https://doi.org/10.1007/978-3-662-58146-9_7

combinations of other identities. Therefore ID_I is typically the output of a one-way hash function applied to the identifying data.

7.1.1 Security Model for Identity-Based Cryptosystems

In an identity-based cryptosystem, the public key of any entity is determined by that entity's identity string and any public parameters of the system. This is a very attractive way of obtaining keying material, but unfortunately there is a significant drawback. No principal can be allowed to generate its own private key. If this were possible then *any* entity could do the same; this would mean that any entity could masquerade as any other. Therefore, in all identity-based schemes the private key of each principal must be generated by a trusted third party. Such a degree of trust may not always be reasonable, although it is probably acceptable in a corporate environment. This trusted party is usually known as the *key generation centre* (KGC).

In order to generate private keys, the KGC must have some secret information that depends on the public parameters of the system. (All parties must obtain authentic copies of the public parameters.) This secret information is known as the *master secret* of the identity-based cryptosystem. We usually assume that the public system parameters and the master secret are generated by the KGC when the system is initialised. The private keys for an entity I can be generated as needed by the KGC as a function of the identity information ID_I and the master secret. An interesting property of identity-based cryptosystems is that the public key can be used before the private key has even been generated. In identity-based key agreement this can lead to the situation where one party A possesses a key that is implicitly shared with a partner B who is unable to compute that shared key until B contacts the KGC to obtain its private key.

Generation of private keys is often known as *key extraction* in the literature. In formal security models, we normally expect the adversary to have the ability to extract private keys for any parties which are not the target of its attack. This may seem a strong assumption but it reflects the reality that the adversary may be able to obtain private keys for many different entities.

Desirable security properties of identity-based key agreement include all those discussed earlier for key agreement based on certified public keys. An additional property that is desirable is an extended version of forward secrecy with respect to the KGC. Although knowledge of the master secret will allow the adversary to masquerade as any entity, there is no reason that it should also allow previously used session keys to become compromised. Since the KGC private key can be used to obtain any user private key, this is arguably even more important than forward secrecy for conventional key agreement.

Definition 34. *An identity-based key agreement protocol provides* KGC forward secrecy *if compromise of the KGC's master secret does not compromise the session keys established in previous protocol runs.*

The necessity of a KGC to generate user keys is a limitation of identity-based cryptography. It is often referred to as the *escrow* problem, since the KGC can at any time generate a spare user private key. Ways to mitigate the escrow problem have been suggested in the literature. These generally involve a compromise between truly identity-based schemes and ordinary public key schemes using certificates. In Sect. 7.5.2, we will look at Girault's classification and scheme for dealing with the problem. Another possibility, discussed in Sect. 7.5.3, is to use *certificateless* cryptography, for which the KGC generates only a part of the user private key.

7.1.2 Elliptic Curve Pairings

Pairings are functions which take as input two elliptic curve points and *pair* them to form an output in a subgroup of a finite field. Different pairing functions have been used for pairing-based cryptography, but they all have the critical feature of being *bilinear*, that is they are linear in both of the input components.

The first cryptographic application of elliptic curve pairings was in cryptanalysis of elliptic curve cryptosystems. Menezes, Okamoto and Vanstone [544] used the Weil pairing to map elliptic curve discrete logarithms into a finite field, which, for certain curve types, results in a much easier way of solving the problem. This is often called the *MOV attack* on elliptic curve cryptosystems. It was only later that it was realised that pairings can be used constructively to design cryptosystems with properties previously unavailable. The Weil pairing was modified by Boneh and Franklin [123] as a concrete construction for their identity-based encryption scheme. Later, other pairings have been proposed for cryptographic purposes, particularly the Tate pairing and variants thereof.

A full explanation of the construction of elliptic curve pairings is beyond the scope of this book and the reader is referred elsewhere for the mathematical details [221]. In general, the two points that are paired come from different elliptic curve groups, which we denote $\mathbb{G}_1$ and $\mathbb{G}_2$. The output of the pairing is a finite-field point from a subgroup which we denote $\mathbb{G}_T$, sometimes called the *target group*. There is potential for confusion about the notation because the early literature on identity-based cryptography used additive notation in groups $\mathbb{G}_1$ and $\mathbb{G}_2$ as is traditional for elliptic curve groups. However, more recently it has become normal to use multiplicative notation for all three groups so we adopt this convention in this chapter.

Using the multiplicative notation, we let $\mathbb{G}_1$ be a group of prime order q and $\mathbb{G}_2$ be a group of the same order q. We assume the existence of the pairing map $\hat{e}$ from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. The mapping $\hat{e}$ must be efficiently computable and have the following properties.

Bilinear: for $w,x \in \mathbb{G}_1$ and $y,z \in \mathbb{G}_2$, both

$$\hat{e}(w,yz) = \hat{e}(w,y) \cdot \hat{e}(w,z) \quad \text{and} \quad \hat{e}(wx,z) = \hat{e}(w,z) \cdot \hat{e}(x,z).$$

Non-degenerate: for some elements $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, we have $\hat{e}(g,h) \neq 1$.

When $a \in \mathbb{Z}_q$ and $w \in \mathbb{G}_1$, we write w^a for exponentiation in $\mathbb{G}_2$ (traditionally called elliptic curve scalar multiplication). Owing to bilinearity, for any $w \in \mathbb{G}_1$, $y \in \mathbb{G}_2$ and $a,b \in \mathbb{Z}_q$ we have

$$\hat{e}(w^a, y^b) = \hat{e}(w, y)^{ab} = \hat{e}(w^{ab}, y) = \hat{e}(w, y^{ab}).$$

For some types of elliptic curves (specifically, supersingular curves), it is possible to assume that the two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same. Such pairings are often called *symmetric pairings*. However, such a choice restricts the efficiency of the resulting protocols, so in general it is preferable to avoid this assumption. The details of the properties of different pairings are complex for the non-specialist to appreciate, and so Galbraith *et al.* [289] summarised the important properties and classified pairing groups into three types. The relevant issues include:

- the availability of an efficient homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$;
- the possibility to hash efficiently into $\mathbb{G}_2$;
- the efficiency of exponentiation in each group;
- the size of element representation in each group.

Chen *et al.* [192] have considered in detail the effect of different pairing types on the efficiency of a wide variety of identity-based key agreement schemes. They also introduced a fourth pairing type in addition to the three considered by Galbraith *et al.* [289]. Table 7.1 summarises the defining properties of the different pairing types. Note that although Type 1 looks the most favourable in this table, its efficiency is limited, especially at higher security levels.

Table 7.1: Pairing types of Galbraith *et al.* [289] and Chen *et al.* [192]

	Symmetric	Homomorphism $\mathbb{G}_2 \to \mathbb{G}_1$	Hash to $\mathbb{G}_2$
Type 1	✓	✓	✓
Type 2	✗	✓	✗
Type 3	✗	✗	✓
Type 4	✗	✓	✓

A pairing-based key agreement scheme usually combines long-term identity-based keys with ephemeral keys (both private and public). When setting up such a scheme, a decision has to be made regarding which group each key will lie in. Extraction of private keys from identities normally entails hashing, so this may not be possible for certain pairing types if long-term keys are to lie in $\mathbb{G}_2$. However, pairings require one input to be from $\mathbb{G}_1$ and the other from $\mathbb{G}_2$ so that, depending on the way that the values are combined, it may be necessary to make use of a homomorphism to take keys from $\mathbb{G}_2$ into corresponding values in $\mathbb{G}_1$. Again, this does not exist for all pairing types.

Making a precise comparison between protocols turns out to be very difficult, since it depends ultimately on the cost of operations on elliptic curves and finite fields, whose optimisation may depend on the details of the specific computing platform. Therefore, in this chapter we will limit ourselves to general observations about the relative efficiency of protocols. We will also explain most protocols using symmetric pairings for ease of exposition, but will also remark on the possibility of using other pairing types. An exception is our treatment of the SOK protocol below, which we use to illustrate the effect of using asymmetric pairings.

For security of key agreement protocols in finite fields we typically need to assume that the Diffie–Hellman problem (and hence also the discrete logarithm problem) is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. In pairing-based key exchange a natural problem to base security upon is the bilinear version of the Diffie–Hellman problem.

Definition 35 (*Bilinear Diffie–Hellman (BDH) problem).* *Given $\mathbb{G}_1$, $\mathbb{G}_2$ and $\hat{e}$ as above, the BDH problem is to compute $\hat{e}(g_1,g_2)^{xyz} \in \mathbb{G}_T$ given $\langle g_1,g_2,g_1^x,g_2^y,g_1^z,g_2^z\rangle$ with $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $x,y,z \in \mathbb{Z}_q$.*

This computational assumption is just one of many which have been used in security proofs for different identity-based cryptographic primitives [144]. Sometimes it is possible to relate such assumptions to each other or to existing accepted assumptions. For example, the BDH problem is no harder to solve than solving the Diffie–Hellman problem either in $\mathbb{G}_1$ or in $\mathbb{G}_2$. However, as usual, we do not have any absolute guarantee of the difficulty of any of these problems.

7.1.3 Sakai–Ohgishi–Kasahara Protocol

The Sakai–Ohgishi–Kasahara (SOK) protocol [647] is a fundamental building block in many identity-based key agreement protocols. It is a *non-interactive protocol* and can be regarded as an identity-based analogue of static Diffie–Hellman using traditional public key certificates. In an identity-based infrastructure, it allows any two parties to establish a shared secret without exchange of any messages. The security of SOK relies on the difficulty of the BDH problem.

The SOK protocol makes use of a pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The master secret is a value $s \in \mathbb{Z}_q$ chosen randomly by the KGC. In order to extract private keys we need to hash identity strings onto points in $\mathbb{G}_1$ or $\mathbb{G}_2$, so we define two functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \{0,1\}^* \to \mathbb{G}_2$. Note that defining such an H_2 is not possible for all pairing types so we need to be careful about how to make the protocol work. We consider three variants.

- First, suppose that we are going to use a Type 1 pairing. This means that we can assume $\mathbb{G}_1 = \mathbb{G}_2$, and we only need to have one hash function, H_1. We denote the public keys of A and B as $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$. The private keys of parties A and B will then be the values $d_A = q_A^s$ and $d_B = q_B^s$, each of which can only be computed by the KGC, which is in possession of the master secret s. With the above parameters, any two principals A and B with identities ID_A, ID_B can efficiently calculate a shared secret as:

  $$F_{AB} = \hat{e}(q_A,q_B)^s = \hat{e}(d_A,q_B) = \hat{e}(q_A,d_B).$$

  This variant of the SOK protocol only works with a Type 1 symmetric pairing, since any principal's public/private key pair needs to be defined in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

- Assume now that $\mathbb{G}_1 \neq \mathbb{G}_2$. In order to allow pairing between the keys of any pair of users, we can define the keys of all users to lie in $\mathbb{G}_2$ and use the homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ to move one key into $\mathbb{G}_1$ when the protocol is run. Now we denote the public keys of A and B as $q'_A = H_2(ID_A) \in \mathbb{G}_2$ and $q'_B = H_2(ID_B) \in \mathbb{G}_2$. The corresponding private keys are the values $d'_A = (q'_A)^s \in \mathbb{G}_2$ and $d'_B = (q'_B)^s \in \mathbb{G}_2$. We also need some way of deciding whether party A or party B's key should be moved into $\mathbb{G}_1$. One easy way of doing this is to use any natural ordering on the strings q'_A and q'_B and say that A's key will be moved into $\mathbb{G}_1$ if and only if $q'_A < q'_B$. With the above parameters, any two principals A and B with identities ID_A, ID_B can efficiently calculate a shared key as

$$F_{AB} = \hat{e}(\psi(q'_A), q'_B)^s = \hat{e}(\psi(d'_A), q'_B) = \hat{e}(\psi(q'_A), d'_B).$$

  This SOK variant requires pairings where the homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ exists as well as the hash function H_2. Therefore it can be seen from Table 7.1 that only pairings of Type 1 and 4 are possible here.
- We can broaden the usable pairing types by making each party's identity-based key consist of two values. We now denote the public key of A as $(q_A, q'_A) = (H_1(ID_A), H_2(ID_A))$ and similarly for party B. The private key of party A with identity string ID_A will then be the pair $(d_A, d'_A) = (q_A^s, {q'_A}^s)$. With the above parameters, any two principals A and B with identities ID_A, ID_B can efficiently calculate the shared key as

$$F_{AB} = \hat{e}(q_A, q'_B)^s \cdot \hat{e}(q_B, q'_A)^s = \hat{e}(d_A, q'_B) \cdot \hat{e}(q_B, d'_A) = \hat{e}(q_A, d'_B) \cdot \hat{e}(d_B, q'_A).$$

  This variant can be used with pairings of Type 1, 3, and 4, but Type 2 is still ruled out owing to the need for hashing to $\mathbb{G}_2$. Dupont and Enge [261] analysed the security of this variant.

These variants illustrate the typical problems that arise when one tries to apply different pairing types to key agreement protocols. For Types 1 and 4, we can usually make any protocol work. Some protocols can use all pairing types by avoiding pairing between user long-term keys. Chen *et al.* [192] illustrated how this works for a number of protocols. In order to simplify the presentation, we will normally show symmetric pairings in the rest of this chapter.

7.2 Identity-Based Protocols without Pairings

In recent years almost all research in identity-based key establishment has made use of bilinear pairings. However, older protocols are worth studying too, and not just because of their historical interest. For one thing, the older protocols are based on better-established computational assumptions.

Several of the older protocols work in groups with a composite modulus. It is worth emphasising that this does *not* mean that the parameter sizes for such protocols need to be larger than those used in elliptic curve pairings. Although the discrete logarithm problem in elliptic curve groups can remain hard with much smaller parameter sizes than those used in $\mathbb{Z}_p^*$, this does not apply to pairing groups. This is because the possibility of the well-known MOV attack [544] requires that the target group $\mathbb{G}_T$ in the pairing is large enough to resist finite-field discrete logarithm attacks.

7.2.1 Okamoto's Scheme

Okamoto's scheme [589, 590] was the first published identity-based key agreement protocol. It uses a composite modulus $n = pq$ whose factorisation is known only to the KGC. The KGC chooses values e and d as in the RSA algorithm, so that $ed \bmod \phi(n) = 1$, and an element g that is a primitive root in both the integers mod p and the integers mod q. The values g and e are made public.

Before engaging in the key agreement protocol, each user must register with the authority to obtain a private key. User I's identification string, ID_I, is treated as an integer modulo n. The authority calculates the value $s_I = ID_I^{-d} \bmod n$ and distributes s_I securely to user I.

Protocol 7.1 shows the key agreement message flows. The shared secret is defined as $\mathbf{Z} = g^{er_Ar_B}$. On the assumption that it is necessary to know either s_A or s_B in order to find $\mathbf{Z}$, the scheme provides implicit key authentication.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: s_A such that $s_A^e = ID_A^{-1} \bmod n$.
Information known to B: s_B such that $s_B^e = ID_B^{-1} \bmod n$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_n & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{s_A t_A} & r_B \in_R \mathbb{Z}_n \\
 & & t_B = g^{r_B} \\
 & \xleftarrow{s_B t_B} & \\
\mathbf{Z} = ((s_B t_B)^e ID_B)^{r_A} & & \mathbf{Z} = ((s_A t_A)^e ID_A)^{r_B}
\end{array}$$

Protocol 7.1: Okamoto's identity-based protocol

Mambo and Shizuya [514] conducted a computational proof of security of Protocol 7.1 against a passive eavesdropper. They showed that any efficient algorithm that can find the shared secret can also break the Diffie–Hellman problem in $\mathbb{Z}_n^*$. Later, this analysis was extended by Kim *et al.* [430] to show a security proof against active attacks. These authors provided a reduction of attacks on the protocol to the Diffie–Hellman problem or to the RSA problem. However, success of the adversary requires that the complete key is returned, rather than being defined in terms of any partial information about the key being recovered. Later, Gennaro *et al.* [294, 295] provided a security proof assuming only the difficulty of the RSA problem and in a model that requires the adversary only to distinguish the session key from a random value. In this model, ephemeral keys cannot be obtained by the adversary. Gennaro *et al.* did, however, make two small modifications to the original Okamoto protocol. One was that the identities needed to be hashed with a function which they modeled as a random oracle. The other modification was that the computation of the key included an additional squaring operation so that the shared key became $\mathbf{Z} = g^{2er_Ar_B}$. Gennaro *et al.* further included a proof that Protocol 7.1 provides full forward secrecy (not just weak forward secrecy) but only by making a stronger computational assumption (the modified knowledge-of-exponent assumption).

In addition to full forward secrecy, Protocol 7.1 can also be shown to provide KCI resistance (Gennaro *et al.* [294] stated that this can be formally proven).[1] However, a significant weakness of the protocol is that it is very sensitive to compromise of ephemeral keys. If a single ephemeral value, say r_A, becomes available to the adversary then A's long-term secret is also known to the adversary. Thus, although the protocol provides full forward secrecy, it is insecure when any ephemeral key of the victim party is revealed. This means that the protocol is not secure in some models, such as eCK.

Protocol 7.2 is a variant of Protocol 7.1, proposed by Okamoto and Tanaka [590], which includes a hashed value to allow explicit authentication. Timestamps T_A and T_B, are included inside the hashes c_A and c_B, respectively, to ensure freshness. The shared secret is again $\mathbf{Z} = g^{er_Ar_B}$, but this value is calculated in a different way in this variant.

There does not seem to have been any formal analysis carried out on Protocol 7.2. It seems reasonable to assume that the properties of Protocol 7.1 would carry over to Protocol 7.2 once the modifications of Gennaro *et al.* [294, 295] discussed above are incorporated. In addition, mutual explicit authentication is claimed to be achieved as long as synchronised timestamps are available. Using timestamps instead of nonces allows mutual authentication to be completed with only two messages.

A similar variant was later published by Shieh *et al.* [667] with claimed computational advantages, but Yen [757] showed that explicit authentication fails. A further variation is to provide one-pass key establishment (see also Sect. 7.5.5) suitable for applications such as electronic mail. A scheme originally proposed by Okamoto and Tanaka [590] was shown by Tsai and Hwang [714] to be vulnerable to attacks by insiders, and Tsai and Hwang proposed new schemes of their own. Later, Tanaka and Okamoto [707] designed another scheme to prevent the KGC from obtaining session keys (although the KGC can always masquerade as any user).

[1] In the first edition of this book we erroneously stated that Protocol 7.1 is vulnerable to a KCI attack.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: s_A such that $s_A^e = ID_A^{-1} \bmod n$.
Information known to B: s_B such that $s_B^e = ID_B^{-1} \bmod n$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_n, u_A = g^{er_A} & & \\
c_A = H(u_A, ID_A, ID_B, T_A) & & \\
v_A = s_A g^{c_A r_A} & \xrightarrow{u_A, v_A} & c_A = H(u_A, ID_A, ID_B, T_A) \\
 & & ID_A \overset{?}{=} u_A^{c_A} / v_A^e \\
 & & r_B \in_R \mathbb{Z}_n, u_B = g^{er_B} \\
 & & c_B = H(u_B, ID_B, ID_A, T_B) \\
 & \xleftarrow{u_B, v_B} & v_B = s_A g^{c_B r_B} \\
c_B = H(u_B, ID_B, ID_A, T_B) & & \\
ID_B \overset{?}{=} u_B^{c_B} / v_B^e & & \\
\mathbf{Z} = u_B^{r_A} & & \mathbf{Z} = u_A^{r_B}
\end{array}$$

Protocol 7.2: Okamoto–Tanaka identity-based protocol

7.2.2 Günther's Scheme

Günther [336] proposed an identity-based scheme in the familiar setting of $\mathbb{Z}_p^*$. The KGC has private and public key pair x_S and $y_S = g^{x_S}$. In the registration phase, user I obtains an ElGamal signature (u_i, v_i) generated by the KGC by choosing k_i randomly in $\mathbb{Z}_p^*$ (but coprime to $p-1$) and setting $u_i = g^{k_i}$, $v_i = (ID_I - x_S u_i)/k_i \bmod p-1$. The signature pair (u_i, v_i) is given to I. The verification equation of the ElGamal signature scheme can be written as

$$u_i^{v_i} = g^{ID_I} y_S^{-u_i}.$$

Although any party can verify this equation, the idea is not to reveal v_i, but rather to reveal that I is the only party who is in possession of the discrete logarithm of $g^{ID_I} y_S^{-u_i}$ to the base u_i. Protocol 7.3 shows a successful protocol run. The shared secret is $\mathbf{Z} = u_A^{r_B v_A} u_B^{v_B r_A}$. In the description of Protocol 7.3 (and also for Protocol 7.4), we have omitted the calculation of the value ID_A by B and the value ID_B by A from the basic identifying information, which was explicitly included in the original description.

It can be seen that compromise of the long-term secrets v_A and v_B enables the adversary to find old session keys, so that forward secrecy is not provided. Therefore Günther also proposed Protocol 7.4 as an extension of the basic protocol that provides forward secrecy at the cost of an extra exponentiation on each side. A and B choose additional random values r'_A and r'_B, respectively. This time the shared secret is the same as in the basic protocol but multiplied by the ephemeral Diffie–Hellman key: $\mathbf{Z} = u_A^{r_B v_A} u_B^{v_B r_A} g^{r'_A r'_B}$. This idea of incorporating an unauthenticated

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: ElGamal signature of KGC on ID_A, (u_A, v_A), so that $g^{ID_A} = y_S^{u_A} u_A^{v_A}$.
Information known to B: ElGamal signature of KGC on ID_B, (u_B, v_B), so that $g^{ID_B} = y_S^{u_B} u_B^{v_B}$.

$$\begin{array}{ccc}
A & & B \\
 & \xrightarrow{ID_A, u_A} & \\
 & \xleftarrow{ID_B, u_B} & X_B = g^{ID_A} y_S^{-u_A} \\
X_A = g^{ID_B} y_S^{-u_B} & & \\
r_A \in_R \mathbb{Z}_q & & \\
w_A = u_B^{r_A} & \xrightarrow{w_A} & r_B \in_R \mathbb{Z}_q \\
 & \xleftarrow{w_B} & w_B = u_A^{r_B} \\
\mathbf{Z} = w_B^{v_A} X_A^{r_A} & & \mathbf{Z} = w_A^{v_B} X_B^{r_B}
\end{array}$$

Protocol 7.3: Günther's key agreement protocol

Diffie–Hellman value into the shared secret may seem strange at first but has often been used in other protocol designs.

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: ElGamal signature of KGC on ID_A, (u_A, v_A), so that $g^{ID_A} = y_S^{u_A} u_A^{v_A}$.
Information known to B: ElGamal signature of KGC on ID_B, (u_B, v_B), so that $g^{ID_B} = y_S^{u_B} u_B^{v_B}$.

$$\begin{array}{ccc}
A & & B \\
 & \xrightarrow{ID_A, u_A} & \\
 & \xleftarrow{ID_B, u_B} & X_B = g^{ID_A} y_S^{-u_A} \\
X_A = g^{ID_B} y_S^{-u_B} & & \\
r_A, r'_A \in_R \mathbb{Z}_q & & \\
w_A = u_B^{r_A} & & \\
t_A = g^{r'_A} & \xrightarrow{w_A, t_A} & r_B, r'_B \in_R \mathbb{Z}_q \\
 & & w_B = u_A^{r_B} \\
 & \xleftarrow{w_B, t_B} & t_B = g^{r'_B} \\
\mathbf{Z} = w_B^{v_A} X_A^{r_A} t_B^{r'_A} & & \mathbf{Z} = w_A^{v_B} X_B^{r_B} t_A^{r'_B}
\end{array}$$

Protocol 7.4: Günther's extended key agreement protocol

There is no security proof, or even formal security property claimed, for the original protocol of Günther, but later Fiore and Gennaro [277] did provide a proof for the extended version (Protocol 7.4), with a slight modification to the signature algorithm and where a key derivation function is applied. Informally, we may observe a similarity with the MTI protocol A(0) and, consequently, it seems reasonable to assume that one of v_A or r_B, and one of v_B or r_A, are required to find $\mathbf{Z}$. With this assumption, it follows that resistance to key compromise impersonation is provided. Indeed, resistance to key compromise impersonation and weak forward secrecy were later proven by Fiore and Gennaro [277] for their adapted version of Protocol 7.4.

Fiore and Gennaro [277] discovered a reflection attack, applicable to both Protocol 7.4 and Protocol 7.3, which allows an adversary to impersonate A to herself and obtain the correct session key. The protocols should therefore not be used in a scenario where A and B may be the same entity (such as where A shares the same key across different devices). As with Okamoto's protocol, if users cannot choose their own identities then the unknown key-share attacks on the MTI protocols do not carry over. Burmester [167] showed that his triangle attack is applicable to both versions of the protocol.

A potential disadvantage of Protocols 7.4 and 7.3 is that four messages are required, although it is worth noting that messages 2 and 4 can be combined in both protocol versions. Saeednia [640] proposed Protocol 7.5 as a variant of Günther's scheme that reduces the number of messages to only two. The idea is to replace the equation for the user's secret v_i with $v_i = ID_I k_i + x_S u_i \bmod (p-1)$ while, as before, the public key is $u_i = g^{k_i}$. Consequently, A can generate her random value $t_A = g^{r_A}$ for the protocol without waiting to receive B's public value u_B. The shared secret becomes $\mathbf{Z} = g^{v_A r_B + v_B r_A}$. The change effectively replaces the ElGamal signature in Günther's protocol with a variant signature (in fact, it is variant 3 in Table 11.5 in the *Handbook of Applied Cryptography* [550]). Saeednia also showed how to add forward secrecy to Protocol 7.5 without further message exchanges.

Fiore and Gennaro [277] provided a security proof for Protocol 7.5 after making modifications to the signature algorithm and a modification where a key derivation function is applied, similarly to how they obtained a proof for Protocol 7.4. Their security analysis includes a proof of weak forward secrecy, and they also argued for security against key compromise impersonation. Unlike Protocol 7.4, Protocol 7.5 is also secure against reflections attacks, according to the analysis of Fiore and Gennaro.

7.2.3 Fiore–Gennaro Scheme

Fiore and Gennaro [277, 278] proposed a protocol which can be viewed as an improvement of Protocol 7.5. The major difference is that a Schnorr signature is used to form the private keys of users. This allows for a more efficient protocol. Moreover, Fiore and Gennaro provided a proof of security in the Canetti–Krawczyk model. The proof covers (weak) forward secrecy and KCI resistance in addition to basic protocol security. The messages exchanged are shown in Protocol 7.6.

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: Signature of KGC on ID_A, (u_A, v_A), so that $g^{v_A} = y_S^{u_A} u_A^{ID_A}$.
Information known to B: Signature of KGC on ID_B, (u_B, v_B), so that $g^{v_B} = y_S^{u_B} u_B^{ID_B}$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{ID_A, u_A, t_A} & \\
 & & r_B \in_R \mathbb{Z}_q \\
 & & t_B = g^{r_B} \\
 & \xleftarrow{ID_B, u_B, t_B} & \\
X_A = u_B^{ID_B} y_S^{u_B} & & X_B = g^{ID_A} y_S^{u_A} \\
\mathbf{Z} = t_B^{v_A} X_A^{r_A} & & \mathbf{Z} = t_A^{v_B} X_B^{r_B}
\end{array}$$

Protocol 7.5: Saeednia's variant of Günther's key agreement protocol

Shared information: Public key y_S of KGC, where $y_S = g^{x_S}$ for KGC private key x_S.
Information known to A: Signature of KGC on ID_A, (u_A, v_A), so that $g^{v_A} = u_A y_S^{H_1(ID_A, u_A)}$.
Information known to B: Signature of KGC on ID_B, (u_B, v_B), so that $g^{v_B} = u_B y_S^{H_1(ID_B, u_B)}$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{ID_A, u_A, t_A} & \\
 & & r_B \in_R \mathbb{Z}_q \\
 & & t_B = g^{r_B} \\
 & \xleftarrow{ID_B, u_B, t_B} & \\
z_1 = (t_B u_B y_S^{H_1(ID_B, u_B)})^{(r_A + v_A)} & & z_1 = (t_A u_A y_S^{H_1(ID_A, u_A)})^{(r_B + v_B)} \\
z_2 = t_B^{r_A} & & z_2 = t_A^{r_B} \\
\mathbf{Z} = H_2(z_1, z_2) & & \mathbf{Z} = H_2(z_1, z_2)
\end{array}$$

Protocol 7.6: Fiore–Gennaro key agreement protocol

The security proof for Protocol 7.6 relies on a computational assumption known as the *strong Diffie–Hellman* assumption. It also models the hash functions H_1 and H_2 as random oracles. Cheng and Ma [198] pointed out that Protocol 7.6 is not secure if ephemeral keys may be leaked, as assumed in the eCK model, but this was not allowed in the model used by Fiore and Gennaro.

7.2.4 Comparison

Table 7.2 summarises the protocols we have examined in this section, comparing their efficiency and security properties. Most of these protocols were originally published a long while back when security properties and modelling of protocols had not been extensively developed. It is therefore not surprising that originally many of these protocols did not carry security proofs. Recent work has 'modernised' some of these protocols, for example with the proof of Okamoto's protocol by Gennaro *et al.* [294, 295] and the Fiore–Gennaro version of Saeednia's protocol. An asterisk in the table indicates that a property may not hold for the original protocol; consult the section about that protocol for details.

Table 7.2: Summary of ID-based protocols without pairings. FS: forward secrecy; KCIR: key compromise impersonation resistance; ROM: random oracle model

Protocol	**Message passes**	**Modulus type**	**FS**	**KCIR**	**Proof**	**Exponentiations on/offline**
Okamoto (7.1)	2	Composite	Full*	Yes*	ROM*	1/1
OT (7.2)	2	Composite	Yes	Yes	No	2/2
Günther (7.3)	4	Prime	No	Yes	No	3/0
Günther (7.4)	4	Prime	Weak	Yes	ROM*	4/0
Saeednia (7.5)	2	Prime	Weak	Yes	ROM*	2/1
Fiore–Gennaro (7.6)	2	Prime	Weak	Yes	ROM	2/1

Most of the protocols in this section provide some form of forward secrecy, but usually this is only weak forward secrecy. An important feature of the Okamoto protocol in its refinement by Gennaro *et al.* [294, 295] is that it provides full forward secrecy. As noted in Sect. 7.2.1 this comes at the cost of fragile security in the face of compromise of ephemeral keys.

Although their origins are from quite some time ago, the protocols in this section are still competitive with the more modern pairing-based protocols examined in the next section. The computational requirements shown in Table 7.2 are divided into two parts, online and offline. The offline computations are those that can be computed before the protocol run starts. We have counted as offline those computations that require knowledge of the identity of the peer. Multi-exponentiations are counted the same as a single exponentiation in Table 7.2, while simpler computations are ignored altogether. Overall, the computational comparison should only be regarded as indicative.

7.3 Pairing-Based Key Agreement with Basic Message Format

We now turn to the popular case of identity-based key agreement using elliptic curve pairings. It is reasonable to ask what advantage there is in identity-based key agreement based on pairings in comparison with the older identity-based protocols considered in Section 7.2 above. Generally, the answer might be expected to be the same advantages as that of using elliptic curves over older public key technology, namely a saving in computation and key size. This may be true with regard to savings in bandwidth since message exchanges can be considerably shorter. However, it is not necessarily the case in terms of computation, because the pairing operation can be quite costly. Research still continues into deciding how to implement pairings most efficiently. In Sect. 7.3.9 we compare the efficiency of many pairing-based key agreement protocols. Another possible reason for choosing pairing-based key agreement is to exploit the infrastructure for identity-based cryptography, with its many other benefits.

In this section we survey a number of protocols, focusing on those which have two message passes, one in each direction between principals A and B, and which do not provide explicit authentication. (Protocols which include explicit authentication are discussed in Sect. 7.4.) There are three ingredients defining most of these protocols: the format of the key pair, the format of the exchanged messages, and the construction of the session key. We consider each of these in turn.

Key pair. Most protocols use the key construction from the first protocol of Sakai *et al.* [647], which was discussed in Sect. 7.1.3. We call this type of key the *SOK type*. There also a few examples of protocols using an alternative key type first suggested by Sakai and Kasahara [646], which we call the *SK type*.

- SOK-type keys make use of a hash function, H_1, which outputs members of the elliptic curve group $\mathbb{G}_1$. Boneh and Franklin [123] suggested an explicit H_1 function for a particular elliptic curve which costs one exponentiation in the underlying field. Then the SOK-type public key for entity A is $q_A = H_1(ID_A) \in \mathbb{G}_1$ and the private key is $d_A = q_A^s$.
- In contrast, SK-type private keys use a hash function $\hat{H}_1$ whose output is a scalar in $\mathbb{Z}_q$. In this case any regular hash function can be used for $\hat{H}_1$; the output bit string can be mapped to a number in $\mathbb{Z}_q$ in the natural way. The public key for entity A is then $q'_A = g^{s+\hat{H}_1(ID_A)}$, which can be calculated as $g^s \cdot g^{q_A}$. so that it depends on the master public key, g^s, as well as on ID_A. The SK-type private key is $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$. Note that this construction implies that $\hat{e}(d'_A, q'_A) = \hat{e}(g,g)$.

Message structure. In order to obtain the best efficiency, it is desirable to minimise the length of messages. Many protocols send only one message element typically consisting of an elliptic curve point, which can be viewed as an ephemeral key. This section is limited to protocols with only two messages. In Sect. 7.4 we look at protocols which include an authentication value, which is checked by the recipient before the session key is accepted.

Session key construction. There are many different ways in which the exchanged messages can be combined in order to derive the session key. Each party uses the received message together with its private long-term key and its short-term random input.

In the following protocol descriptions, we will assume that all users have access to the public parameters for the identity-based system. A random value $s \in \mathbb{Z}_q$ plays the role of the *master secret* of the KGC. The published values include descriptions of the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing $\hat{e}$, a point g that generates $\mathbb{G}_1$, and a master public key $h = g^s$. The KGC distributes to each party P_i with identity ID_i a long-term key pair of either SOK or SK type. We will usually assume that the pairing is a symmetric (Type 1) pairing so that $\mathbb{G}_1 = \mathbb{G}_2$. As with the key agreement protocols examined in Chap. 5 using public key certificates, we assume here that parties agree on a shared secret $\mathbf{Z}$ from which the session key will be derived using an appropriate key derivation function. Usually we do not mention the KDF explicitly but sometimes protocol designers have had a specific KDF in mind, which may have an effect on the security properties. For example, including the protocol messages in the KDF can prevent some kinds of attack. Table 7.3 summarises the notation.

Table 7.3: Notation and terminology for pairing-based schemes

ID_I	Identity string for entity I
KGC	Key generation centre
s	KGC master secret
h	KGC master public key: $h = g^s$.
$\hat{e}$	Elliptic curve pairing: $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
g	Generator of group $\mathbb{G}_1$
q_A	Public key of user A for SOK type: $q_A = H_1(ID_A)$
q'_A	Public key of user A for SK type: $q'_A = g^{s+\hat{H}_1(ID_A)}$
d_A	Private key of user A for SOK type: $d_A = q_A^s$
d'_A	Private key of user A for SK type: $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$
$\mathbf{Z}$	Shared secret

7.3.1 Smart's Protocol

Smart [679] seems to have been the first to propose, in 2002, an identity-based authenticated key agreement protocol based on pairings. The exchanged messages are no different from ordinary ephemeral Diffie–Hellman keys on elliptic curves. The pairing is used to combine identity-specific information about the parties so that only the participants should be able to obtain the shared secret $\mathbf{Z}$. Smart's protocol and its variants all use SOK-type keys, so $d_A = q_A^s$.

The messages and key computation are shown in Protocol 7.7. When both principals follow the protocol without interference, the shared secret is

Shared information: Master public key $h = g^s$ for KGC private key s. $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{t_A} & r_B \in_R \mathbb{Z}_q \\
 & & t_B = g^{r_B} \\
 & \xleftarrow{t_B} & \\
\mathbf{Z} = \hat{e}(q_B^{r_A}, h) \cdot \hat{e}(d_A, t_B) & & \mathbf{Z} = \hat{e}(q_A^{r_B}, h) \cdot \hat{e}(d_B, t_A)
\end{array}$$

Protocol 7.7: Smart's identity-based key agreement protocol

$$\mathbf{Z} = \hat{e}(q_B^{r_A}, h) \cdot \hat{e}(q_A^{r_B}, h) = \hat{e}(q_B^{r_A} q_A^{r_B}, h).$$

Smart's protocol comes with no proof of security but it has been the basis of a number of later protocols which have been proven secure, and this can give confidence in the basic security properties of the protocol. However, the protocol does not provide forward secrecy. An adversary who obtains the two long-term private keys d_A and d_B can compute the shared key of an observed protocol run as $\mathbf{Z} = \hat{e}(d_B, t_A) \cdot \hat{e}(d_A, t_B)$. Since the KGC can generate d_A and d_B from knowledge of s, this also means that KGC forward secrecy is not provided either.

In order to provide a high level of security, it is essential that A and B check that the received values t_B and t_A lie in the group generated by g. The cost of this check is relatively cheap in the case that the protocol is implemented using a Type 1 pairing or, in general, that g lies in $\mathbb{G}_1$ when $\mathbb{G}_1$ and $\mathbb{G}_2$ are different. Chen *et al.* [192] pointed out a simple certificational attack in which the adversary simply multiplies one of the exchanged messages by a low-order value outside the group. Since this low-order element will disappear during the exponentiation with a reasonably high probability, A and B will not have matching conversations, so the protocol is broken in a strong security model.

In the original paper [679], Smart's protocol was defined only for Type 1 symmetric pairings. As pointed out by Chen *et al.* [192], it can be run using any pairing type as long as the long-term private keys are generated in $\mathbb{G}_1$.

7.3.2 Variants of Smart's Protocol

Noticing the lack of forward secrecy in Smart's protocol, Chen and Kudla [194] proposed a simple change in 2003. In addition to computing the original shared secret, they proposed to compute a straightforward ephemeral Diffie–Hellman key using the exchanged values t_A and t_B. The messages in Smart's protocol remain unchanged, but the Diffie–Hellman key $g^{r_A r_B}$ is included in the shared secret value. The secret value therefore becomes

$$\mathbf{Z} = \hat{e}(q_B^{r_A} q_A^{r_B}, h), g^{r_A r_B}.$$

Subsequently, Chen *et al.* [192] provided a security proof for this extended protocol using a key derivation function which combines $\mathbf{Z}$, the protocol messages, and the identities of the participants. This proof shows that the protocol provides KGC forward secrecy as well as resistance to key compromise impersonation, on the assumption that the BDH problem is hard.

Later, Choie *et al.* [207] in 2005 proposed another variant of Smart's protocol which has the same basic idea of incorporating the ephemeral Diffie–Hellman value using the exchanged values. The difference in their protocol is that a hash value $f = H(g^{r_A r_B})$ is computed by both parties, where $H : \mathbb{G}_1 \to \mathbb{Z}_q$ is a hash function. The value f is then included as an exponent and the shared secret becomes

$$\mathbf{Z} = \hat{e}(q_B^{r_A} q_A^{r_B}, h)^f,$$

which is computed by A as $\mathbf{Z} = \hat{e}(q_B^{f r_A}, h) \cdot \hat{e}(d_A, t_B^f)$ and by B in a symmetrical fashion. Choie *et al.* [207] provided no security proof but claimed that the protocol provides KGC forward secrecy as well as key compromise impersonation resistance. Boyd and Choo [133] pointed out an attack on the protocol in which an adversary can obtain the session key by querying a non-matching session. However, this attack is not due to the basic structure of the protocol but due to the lack of session-specific information in the key derivation function. The attack can be avoided by including a session identifier consisting of the concatenation of the protocol messages inside the key derivation function.

7.3.3 Ryu–Yoon–Yoo Protocol

The protocol due to Ryu, Yoon and Yoo [639] has a simple and elegant structure. Again this protocol uses SOK-type keys, so $d_A = q_A^s$. Protocol 7.8 describes the protocol.

At the end of the protocol execution, both A and B will compute the shared secret $\mathbf{Z} = g^{r_B r_A}, \hat{e}(q_A, q_B)^s$. This is simply the concatenation of the ephemeral Diffie–Hellman key using the exchanged values and the SOK non-interactive key examined in Sect. 7.1.3. Since the SOK protocol can be regarded as analogous to static Diffie–Hellman, we can say that there is an analogy between Protocol 7.8 and the Unified Model protocol (Protocol 5.12). It is therefore not surprising that the properties of these two protocols are similar.

Ryu *et al.* [639] claimed that the protocol provides KCI resistance, but this is not the case [133, 727]. It is easy to see that computing the key for the SOK protocol requires knowledge of only one of d_A and d_B. Therefore, in a KCI attack, an adversary who knows d_A can compute the SOK key for any party claiming to share a key with A. Boyd and Choo [133] also described a 'key replicating attack' on Protocol 7.8. However, like the similar attack on the protocol of Choie *et al.* [207] mentioned in

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{t_A} & r_B \in_R \mathbb{Z}_q \\
 & & t_B = g^{r_B} \\
 & \xleftarrow{t_B} & \\
\mathbf{Z} = t_B^{r_A}, \hat{e}(d_A, q_B) & & \mathbf{Z} = t_A^{r_B}, \hat{e}(d_B, q_A)
\end{array}$$

Protocol 7.8: Ryu–Yoon–Yoo protocol

Sect. 7.3.2, this can be avoided by including a suitably defined session identifier in the key derivation function.

Wang *et al.* [728] provided a security proof for Protocol 7.8 when used together with a specific key derivation function. Specifically, the session key $\mathbf{K}$ is defined as

$$\mathbf{K} = H(ID_A, ID_B, \mathbf{Z}, t_A, t_B),$$

where H is a hash function modelled as a random oracle. The computational assumption is that the BDH problem is hard. A proof was also provided for KGC forward secrecy. Because the identity-based keys are used on both sides of the pairing, Protocol 7.8 can only be implemented on Type 1 and Type 4 pairings.

7.3.4 Shim's Protocol

Another early proposal for identity-based key agreement was Shim's protocol, published in 2003 [668]. This protocol uses SOK-type keys, so $d_A = q_A^s$. The original version had some serious problems, but it has later formed the basis of other protocols which have been proven secure.

The messages and key computation are shown in Protocol 7.9. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(t_A d_A, t_B q_B)^s = \hat{e}(g,g)^{s r_A r_B} \cdot \hat{e}(q_A, g)^{s r_B} \cdot \hat{e}(g, q_B)^{s r_A} \cdot \hat{e}(q_A, q_B)^s.$$

Since $\mathbf{Z}$ contains the SOK key $\hat{e}(q_A, q_B)^s$ as well as the bilinear Diffie–Hellman key $\hat{e}(g,g)^{s r_A r_B}$, it might intuitively be expected to be secure. However, Sun and Hsieh [701] found that the protocol is completely insecure, as shown in Attack 7.1.

The adversary plays in the middle between A and B and alters the messages sent between them. Once A and B complete the protocol, they have agreed on keys which can be computed by the adversary C. Note that because C chooses both u and v, C can compute $\mathbf{Z} = \hat{e}(t_A q_A, h^v)$ and $\mathbf{Z}' = \hat{e}(t_B q_B, h^u)$.

Shared information: Master public key $h = g^s$ for KGC private key s. $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{t_A} & r_B \in_R \mathbb{Z}_q \\
 & & t_B = g^{r_B} \\
\mathbf{Z} = \hat{e}(h^{r_A} d_A, t_B q_B) & \xleftarrow{t_B} & \mathbf{Z} = \hat{e}(h^{r_B} d_B, t_A q_A)
\end{array}$$

Protocol 7.9: Shim's protocol

$$\begin{array}{ccccc}
A & & C & & B \\
r_A \in_R \mathbb{Z}_q & & & & \\
t_A = g^{r_A} & & & & \\
 & \xrightarrow{t_A} & u \in_R \mathbb{Z}_q & & \\
 & & t'_A = g^u / q_A & & \\
 & & & \xrightarrow{t'_A} & r_B \in_R \mathbb{Z}_q \\
 & & v \in_R \mathbb{Z}_q & \xleftarrow{t_B} & t_B = g^{r_B} \\
 & & t'_B = g^v / q_B & & \\
 & \xleftarrow{t'_B} & & & \\
\mathbf{Z} = \hat{e}(h^{r_A} d_A, t'_B q_B) & & & & \mathbf{Z}' = \hat{e}(h^{r_B} d_B, t'_A q_A) \\
= \hat{e}(h^{r_A} d_A, g^v) & & & & = \hat{e}(h^{r_B} d_B, g^u) \\
= \hat{e}(g^{r_A} q_A, h^v) & & & & = \hat{e}(g^{r_B} q_B, h^u)
\end{array}$$

Attack 7.1: Sun and Hsieh's attack on Shim's protocol

Later, in 2005, Yuan and Li [769] proposed a simple variation of Shim's protocol in order to avoid Attack 7.1. As in the Chen and Kudla variant of Smart's protocol, the change is simply to add the ephemeral Diffie–Hellman value to the definition of the shared secret, which therefore becomes

$$\mathbf{Z} = \hat{e}(t_A d_A, t_B d_B), g^{r_A r_B}.$$

Yuan and Li did not provide any formal security analysis, but Chen *et al.* [192] later provided a proof of security under the BDH assumption. The proof shows that this protocol provides all the desirable properties, including KGC forward secrecy and KCI resistance.

Huang and Cao [368] proposed another variant, which is very similar to the Yuan and Li protocol. They make use of the twin Diffie–Hellman construction of Cash *et al.* [184]. The only difference from Yuan and Li's protocol is that twin public keys are constructed and used in a duplicate way in the protocol. The consequence of this is to make the proof of security simpler and more complete.

Because the identity-based keys are used on both sides of the pairing, Protocol 7.9 can only be implemented on Type 1 and Type 4 pairings.

7.3.5 Scott's Protocol

Scott [660] published an early pairing-based protocol in which the user's private key is stored as a combination of a user password and a secret stored on a physical device. In the following description, we ignore the implementation details and use the same assumptions as usual with regard to the ways that keys are constructed.

In contrast to the previous protocols in this section, Scott's protocol uses messages which are dependent on the identity of the recipient. This results in the pairing operation being used before the message is sent, but it is not needed to compute the final shared secret. Protocol 7.10 describes the message exchange.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
p_A = \hat{e}(d_A, q_B)^{r_A} & \xrightarrow{\quad p_A \quad} & r_B \in_R \mathbb{Z}_q \\
 & & p_B = \hat{e}(d_B, q_A)^{r_B} \\
\mathbf{Z} = p_B^{r_A} & \xleftarrow{\quad p_B \quad} & \mathbf{Z} = p_A^{r_B}
\end{array}$$

Protocol 7.10: Scott's protocol

At the end of the protocol execution, both A and B will compute the shared secret $\mathbf{Z} = \hat{e}(q_A, q_B)^{s r_A r_B}$ which is equal to the SOK key raised to the power $r_A r_B$. The message exchange can be regarded as analogous to the MTI C(0) protocol and the key computation and protocol properties are rather similar.

Scott [660] presented an argument that the security of Protocol 7.10 can be reduced to the BDH problem, but he did not consider a full formal model. He also argued that the protocol provides forward secrecy, and this also appears to extend to KGC forward secrecy. However, the protocol does not provide resistance to KCI attacks since either of the long-term private keys is sufficient to compute the SOK key that forms part of the basis of the protocol.

Because the identity-based keys are used on both sides of the pairing, Protocol 7.10 can only be implemented on Type 1 and Type 4 pairings.

7.3.6 Chen–Kudla Protocol

Chen and Kudla [194] designed a number of protocols aimed at improving the efficiency and security of Smart's protocol. One of these has already been discussed in Sect. 7.3.2. The protocol described below reduces the number of pairing computations for each party from two to one in comparison with Protocol 7.7. Notice that the messages exchanged are also different. This protocol uses SOK-type public keys.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
w_A = q_A^{r_A} & & \\
 & \xrightarrow{w_A} & r_B \in_R \mathbb{Z}_q \\
 & \xleftarrow{w_B} & w_B = q_B^{r_B} \\
\mathbf{Z} = \hat{e}(d_A, w_B q_B^{r_A}) & & \mathbf{Z} = \hat{e}(d_B, w_A q_A^{r_B})
\end{array}$$

Protocol 7.11: Chen–Kudla protocol

The messages and key computation are shown in Protocol 7.11. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(q_A^s, q_B^{r_B+r_A}) = \hat{e}(q_B^s, q_A^{r_A+r_B}) = \hat{e}(q_A, q_B)^{s(r_A+r_B)}.$$

In the original published paper [194], Chen and Kudla claimed a complete security proof for this protocol. However, subsequently [195] they pointed out a flaw in their argument (finding this flaw is attributed to Zhaohui Cheng) and only claimed security when the adversary is prevented from obtaining old session keys via reveal queries.

Protocol 7.11 does not provide forward secrecy, except for partial forward secrecy when only one of the private keys is revealed. It does, however, provide KCI resistance as proven by Chen and Kudla under the BDH assumption [194, Theorem 2], but again only when the adversary is prevented from obtaining old session keys. Chen and Kudla proposed a modification of Protocol 7.11 which adds in a separate unauthenticated Diffie–Hellman exchange, in the same manner as in their modification of Protocol 7.7. Because the identity-based keys are used on both sides of the pairing, Protocol 7.11 can only be implemented on Type 1 and Type 4 pairings.

7.3.7 Wang's Protocol (IDAK)

Wang [731, 732] designed a protocol with the same message exchange as inProtocol 7.11 but with a more complex computation of the shared secret. In compensation for this extra work a higher level of security is achieved – specifically, there is a security proof which allows reveal queries and proves forward secrecy. Wang called this protocol IDAK, to indicate identity-based and authenticated key agreement. This protocol uses SOK-type public keys.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
w_A = q_A^{r_A} & & \\
 & \xrightarrow{w_A} & r_B \in_R \mathbb{Z}_q \\
 & & w_B = q_B^{r_B} \\
s_A = H_2(w_A, w_B) & \xleftarrow{w_B} & s_A = H_2(w_A, w_B) \\
s_B = H_2(w_B, w_A) & & s_B = H_2(w_B, w_A) \\
\mathbf{Z} = \hat{e}(d_A^{r_A+s_A}, w_B q_B^{s_B}) & & \mathbf{Z} = \hat{e}(d_B^{r_B+s_B}, w_A q_A^{s_A})
\end{array}$$

Protocol 7.12: Wang's protocol

The messages and key computation are shown in Protocol 7.12. The protocol makes use of an additional hash function $H_2 : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{Z}_q^*$. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(q_A^{r_A+s_A}, q_B^{r_B+s_B})^s = \hat{e}(q_A, q_B)^{s(r_A+s_A)(r_B+s_B)}.$$

Wang [731, 732] proved the security of Protocol 7.12 based on the decisional BDH problem. The proof is in the random oracle model, assuming that both H_1 and H_2 act as random oracles. The security proof includes forward secrecy and key compromise impersonation resilience, but the protocol does not provide KGC forward secrecy. KGC forward secrecy can be achieved by adding a separate unauthenticated Diffie–Hellman exchange.

Wang made some concrete suggestions for how to implement the function H_2 efficiently. One is to use a hash function with range $\mathbb{Z}_{q/2}^*$. Although this invalidates the full proof, Wang claimed that there was formal evidence that the protocol was still secure. This choice of H_2 allows the protocol to save half an exponentiation, since the size of s_A and s_B will be half the size of q.

Because the identity-based keys are used on both sides of the pairing, Protocol 7.11 can only be implemented on Type 1 and Type 4 pairings.

Protocol 7.13 is a refinement of Protocol 7.12 due to Chow and Choo [212]. The values s_A and s_B in Protocol 7.12 are replaced by values h_A and h_B in Protocol 7.13. The main effect of this change seems to be that it reduces the amount of *online* computation required by each party: A can compute h_A and $d_A^{r_A+h_A}$ before the protocol run starts.

Shared information: $q_A = H_1(ID_A)$ and $q_B = H_1(ID_B)$.
Information known to A: Private key $d_A = q_A^s$.
Information known to B: Private key $d_B = q_B^s$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
w_A = q_A^{r_A} & & \\
 & \xrightarrow{w_A} & r_B \in_R \mathbb{Z}_q \\
 & & w_B = q_B^{r_B} \\
h_A = H_2(w_B, ID_A) & \xleftarrow{w_B} & h_A = H_2(w_B, ID_A) \\
h_B = H_2(w_A, ID_B) & & h_B = H_2(w_A, ID_B) \\
\mathbf{Z} = \hat{e}(d_A^{r_A+h_A}, w_B q_B^{h_B}) & & \mathbf{Z} = \hat{e}(d_B^{r_B+h_B}, w_A q_A^{h_A})
\end{array}$$

Protocol 7.13: Chow and Choo's protocol

Chow and Choo provided a proof of security for Protocol 7.13 in the Canetti–Krawczyk model. They also formally defined a protocol variant which includes a separate Diffie–Hellman exchange, and showed that this variant provides weak KGC forward secrecy. In addition to considering the usual protocol properties, Chow and Choo also defined a protocol extension designed to provide anonymous key agreement. This extension uses a ring signature so that the peer of a party involved in the protocol can only know that the party is one out of a set (ring) of parties.

7.3.8 McCullagh–Barreto Protocol

McCullagh and Barreto [531] were the first to propose usage of SK-type keys for identity-based key agreement. One advantage of this type of key is that the hash function used, $\hat{H}_1$, does not have to map to an elliptic curve point as in SOK-type keys. Recall that the SK-type public key is $q'_A = hg^{\hat{H}_1(ID_A)} = g^{s+\hat{H}_1(ID_A)}$, with corresponding private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.

The messages and key computation are shown in Protocol 7.14. When both principals follow the protocol without interference, the shared secret is

$$\mathbf{Z} = \hat{e}(g,g)^{r_A r_B}.$$

McCullagh and Barreto pointed out that Protocol 7.14 does not provide KGC forward secrecy. To see this note that the KGC can compute $z_A^{(s+\hat{H}_1(ID_B))^{-1}} = g^{r_A}$

Shared information: $q'_A = g^{s+\hat{H}_1(ID_A)}$ and $q'_B = g^{s+\hat{H}_1(ID_B)}$.
Information known to A: Private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.
Information known to B: Private key $d'_B = g^{1/(s+\hat{H}_1(ID_B))}$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
z_A = (q'_B)^{r_A} & \xrightarrow{z_A} & r_B \in_R \mathbb{Z}_q \\
 & & z_B = (q'_A)^{r_B} \\
\mathbf{Z} = \hat{e}(z_B, d'_A)^{r_A} & \xleftarrow{z_B} & \mathbf{Z} = \hat{e}(z_A, d'_B)^{r_B}
\end{array}$$

Protocol 7.14: McCullagh–Barreto protocol

and $z_B^{(s+\hat{H}_1(ID_A))^{-1}} = g^{r_B}$ from its knowledge of s and the exchanged messages, and then obtain $\mathbf{Z} = \hat{e}(g^{r_A}, g^{r_B})$. They therefore also proposed a second protocol aimed at providing KGC forward secrecy using an asymmetric pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are different and are generated by unrelated elements g_1 and g_2. Private keys of users are generated in $\mathbb{G}_2$; for example, A's private key becomes $d'_A = g_2^{1/(s+\hat{H}_1(ID_A))}$ but public keys remain in $\mathbb{G}_1$ as before. The protocol messages and key computation can then be identical to Protocol 7.14. Note that user keys are applied only on the right-hand side of the pairing, which means that there is no need for a homomorphism between the pairing groups. Also, the SOK-type private key does not require hashing to the pairing group. Therefore any pairing type can be used to implement Protocol 7.14.

Protocol 7.14 was found to be subject to some weaknesses. Firstly, Xie [744] pointed out that the protocol is not resistant to KCI attacks. As a result of this attack, McCullagh and Barreto proposed a protocol variant (included only in the extended version of their paper [531] on the IACR ePrint Archive). This variant avoids the KCI attack but no longer provides forward secrecy, as subsequently pointed out by Xie [744].[2] Xie [745] also proposed a new version of the protocol with the same exchanged messages but a different key computation and shared key $\mathbf{Z} = \hat{e}(g,g)^{r_A r_B + r_A + r_B}$. Xie claimed that this change ensured forward secrecy and resistance to KCI attacks, but this variant was itself broken by Shim [670] and by Li *et al.* [486]. The latter also provided their own variant of Protocol 7.14, designed to avoid the known attacks but without any formal analysis provided.

Meanwhile, Choo [210] had shown another attack on the original version of Protocol 7.14. This attack enables an adversary to recover the session key, given the usual adversary capabilities assumed in the Bellare–Rogaway model. In order to prevent this attack, it is necessary to forbid the adversary from obtaining session keys

[2] Note that there are multiple versions of both the paper of McCullagh and Barreto [531] and the paper of Xie [744], which were updated as understanding developed. These different versions can all be obtained from the IACR ePrint Archive.

from other sessions between the same participants. In the specification of Protocol 7.14, the session key is defined to be a hash of the secret value $\mathbf{Z}$. The final IACR ePrint Archive version of the McCullagh and Barreto paper [531] claims security only when the adversary is restricted from making any reveal queries.

Later, Cheng and Chen [199] showed that all of the existing proofs of Protocol 7.14 and its variants could not be correct, owing to a technical problem. They also provided their own proof of a modified protocol which consists of McCullagh and Barreto's own variant but with an explicit key derivation function, as shown in Protocol 7.15.

Shared information: $q'_A = g^{s+\hat{H}_1(ID_A)}$ and $q'_B = g^{s+\hat{H}_1(ID_B)}$.
Information known to A: Private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.
Information known to B: Private key $d'_B = g^{1/(s+\hat{H}_1(ID_B))}$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
z_A = (q'_B)^{r_A} & \xrightarrow{z_A} & r_B \in_R \mathbb{Z}_q \\
 & & z_B = (q'_A)^{r_B} \\
\mathbf{Z} = \hat{e}(z_B, d'_A)\hat{e}(g,g)^{r_A} & \xleftarrow{z_B} & \mathbf{Z} = \hat{e}(z_A, d'_B)\hat{e}(g,g)^{r_B}
\end{array}$$

$$\mathbf{K} = H_2(ID_A, ID_B, z_A, z_B, \mathbf{Z})$$

Protocol 7.15: Modified McCullagh–Barreto protocol of Cheng and Chen

Protocol 7.15 shows the modified protocol. The messages exchanged are the same as in Protocol 7.14 but the shared secret is computed differently, to obtain $\mathbf{Z} = \hat{e}(g,g)^{r_A+r_B}$. Cheng and Chen provided a proof of security for Protocol 7.15 based on a rather complex computational assumption. However, this protocol still does not provide forward secrecy.

7.3.9 Comparison

Table 7.4 summarises the properties of the protocols which have been described in this section. Earlier surveys by Chen, Cheng and Smart [192] and by Boyd and Choo [133] have their own tables of comparison. The present table includes only the protocols which we have explicitly listed – many other protocols are known, some of which have been mentioned in the text. Recall that the protocols in this section use unauthenticated messages, and private keys are not used in their construction. The asterisks next to the properties for the Shim protocol indicate that these properties refer to the modified version discussed in Sect. 7.3.4.

As discussed in the text, many protocols have evolved over time and sometimes variants have been proposed by others. In the table an asterisk indicates that a prop-

Table 7.4: Summary of implicitly authenticated two-message ID-based protocols. FS: forward secrecy; KCIR: key compromise impersonation resistance; ROM: random oracle model

Protocol	Private key	Message type	FS	KCIR	Proof	Computation on/offline	Pairing types
Smart [679] (7.7)	SOK	g^{r_A}	No	Yes	No	$1P/2E+1P$	All
RYY [639] (7.8)	SOK	g^{r_A}	No	No	ROM [728]	$1E/1E+1P$	1,4
Shim [668] (7.9)	SOK	g^{r_A}	Yes*	Yes*	ROM [192]	$1P/2E$	1,4
Scott (7.10)	SOK	$\hat{e}(d_A,q_B)^{r_A}$	KGC	No	No	$1E/1E+1P$	1,4
CK [194] (7.11)	SOK	$q_A^{r_A}$	No	Yes	Restricted	$1P/2E$	1,4
Wang [731] (7.12)	SOK	$q_A^{r_A}$	Yes	Yes	ROM	$2E+1P/1E$	1,4
CC [212] (7.13)	SOK	$q_A^{r_A}$	KGC	Yes	ROM	$1E+1P/2E$	1,4
MB [531] (7.14)	SK	$(q'_B)^{r_A}$	Yes	No	Restricted	$1E+1P/1E$	All
MBCC [199] (7.15)	SK	$(q'_B)^{r_A}$	No	Yes	ROM	$1E+1P/1E$	All

erty may not hold for the original protocol – consult the section about that protocol for details.

There are some interesting comparisons possible between the protocols seen in Table 7.4 and various protocols using conventional Diffie–Hellman in finite fields. For example, the RYY protocol has strong similarities to the Unified Model protocol. Also, the CK protocol is closely related to the MTI A(0) protocol. Table 7.4 notes whether each protocol provides forward secrecy and key compromise impersonation resistance and has a security proof. In all cases which have forward secrecy, only weak forward secrecy is provided for these two-message protocols.

Table 7.4 also summarises the computation done by each party. We only record pairings (P) from group $\mathbb{G}_1$ to $\mathbb{G}_2$, and exponentiations (E) in either $\mathbb{G}_1$ or $\mathbb{G}_2$. For simplicity, we do not differentiate between exponentiations in $\mathbb{G}_1$ and exponentiations in $\mathbb{G}_2$. Computational requirements are divided into two parts, online and offline. The offline computations are those that can be done before the protocol run starts. We have counted as offline those computations that require knowledge of the identity of the peer. This may not always be realistic. Some computations are also independent of the peer's identity.

The amount of communication required by each protocol can be estimated by looking at the message type sent, as listed in Table 7.4. (Only the message sent from A to B is shown, but all protocols in Table 7.4 are symmetrical in their messages.) Well-known techniques for elliptic curve point compression allow points to be expressed as an element in the underlying field plus a single bit. The message length used is considerably less than for an RSA-based protocol such as Protocol 7.1 if only one point is sent. Protocols that require online pairing computation may be rather inefficient, since a pairing requires several times the computation of an elliptic curve multiplication. However, the exact computation required varies considerably depending on the choice of curve and various implementation details.

Most protocol descriptions ignore the cofactor check that may be required to ensure that the point sent is a member of the prime-order subgroup. Such a check may be important for security reasons (to avoid small subgroup attacks such as those by Lim and Lee [492]). However, when the received point is used in a pairing, the effort required to check that the point is in $\mathbb{G}_1$ is only a small part of the overall computation required.

7.4 Pairing-Based Key Agreement with Explicit Authentication

All of the protocols which we have considered in Sect. 7.3 could be extended to include explicit authentication. This is typically achieved by using a key, generated from the shared secret independently from the session key, to form an explicit authentication tag in each direction. This would usually extend the number of message flows from two to three. In this section we consider some protocols which include explicit authentication.

7.4.1 Boyd–Mao–Paterson Protocol

Boyd, Mao and Paterson [139] proposed a protocol which uses pairings only to authenticate a Diffie–Hellman exchange. The protocol uses the SOK protocol, described in Sect. 7.1.3, to derive a static shared secret, which is then used to authenticate the exchanged messages. Protocol 7.16 shows the message exchange and the computation of the shared secret.

Shared information: Static key F_{AB}, derived from SOK key: $F_{AB} = H_1(\hat{e}(q_A, q_B)^s)$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$t_A = g^{r_A}$		
	$\xrightarrow{t_A}$	$r_B \in_R \mathbb{Z}_q$
		$t_B = g^{r_B}$
Verify hash	$\xleftarrow{t_B, H(F_{AB}, ID_B, t_A, t_B)}$	
	$\xrightarrow{H(F_{AB}, ID_A, t_B, t_A)}$	Verify hash
$\mathbf{Z} = H_2(t_B^{r_A})$		$\mathbf{Z} = H_2(t_A^{r_B})$

Protocol 7.16: Boyd–Mao–Paterson protocol

The authenticator used to define the protocol was proven by Boyd *et al.* to satisfy the definition of a secure authenticator in the sense of Canetti and Krawczyk [178] using the random oracle model and assuming that the bilinear Diffie–Hellman problem is hard. Therefore the protocol inherits a proof of security in the Canetti–Krawczyk model, including the forward secrecy property. However, as noted by the original authors, the protocol does not provide resistance to key compromise impersonation, since the long-term key of either party is sufficient to compute the static secret F_{AB}.

7.4.2 Asymmetric Protocol of Choi *et al.*

Choi *et al.* [206] designed a protocol for use between a low-power client A and a server B. A distinctive feature of their protocol is that A performs no pairings, even though it is a pairing-based protocol. Despite many recent advances, pairing computations are still more expensive than exponentiations, so it is desirable to reduce the number of pairings; by eliminating pairing computation altogether on the client side, there is a corresponding saving in implementation cost too. Protocol 7.17 shows the structure; three hash functions are used, $\hat{H}_1$ is the usual function for SK keys from identity strings to $\mathbb{Z}_q$, and H_2 and H_3 output bit strings.

Shared information: $q'_A = g^{s+\hat{H}_1(ID_A)}$ and $q'_B = g^{s+\hat{H}_1(ID_B)}$.
Information known to A: Private key $d'_A = g^{1/(s+\hat{H}_1(ID_A))}$.
Information known to B: Private key $d'_B = g^{1/(s+\hat{H}_1(ID_B))}$.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
t_A = \hat{e}(g,g)^{r_A} & & \\
z_{A,1} = (q'_B)^{r_A} & & \\
z_{A,2} = (d'_A)^{r_A+\hat{H}_1(t_A)} & & \\
 & \xrightarrow{ID_A,\, z_{A,1},\, z_{A,2}} & t_A = \hat{e}(z_{A,1}, d'_B) \\
 & & \hat{e}(z_{A,2}, q'_A) \overset{?}{=} t_A \cdot \hat{e}(g,g)^{\hat{H}_1(t_A)} \\
 & & r_B \in_R \mathbb{Z}_q \\
 & & z_B = H_2(t_A, r_B, z_{A,1}, z_{A,2}, ID_A, ID_B) \\
 & \xleftarrow{z_B,\, r_B} & \\
z_B \overset{?}{=} H_2(t_A, r_B, z_{A,1}, z_{A,2}, ID_A, ID_B) & & \\
\end{array}$$

$$\mathbf{K} = H_3(t_A, r_B, z_B, z_{A,1}, z_{A,2}, ID_A, ID_B)$$

Protocol 7.17: Protocol of Choi, Hwang, Lee and Seo

Part of Protocol 7.17 is similar to Protocols 7.14 and 7.15 in that the value $z_{A,1}$ can be used by B to recompute the value $t_A = \hat{e}(g,g)^{r_A}$. However, the other part of the message from A, $z_{A,2}$, is intended as a kind of signature to allow B to authenticate the message. We note, however, that B has no way to check if the message from A has been replayed, so it does not provide explicit entity authentication. The server sends its input r_B in cleartext and both parties can then compute the session key as a hash of $\hat{e}(g,g)^{r_A}$, r_B and other public values. B also includes a value z_B, which can be recomputed by A to authenticate the message and to explicitly authenticate B.

Protocol 7.17 provides only partial forward secrecy; compromise of the long-term key of the client, A, does not reveal expired session keys, but compromise of the long-term key of B does. Owing to the authentication of each message, the protocol appears to achieve key compromise impersonation resistance, although this has not been proven. Choi *et al.* [206] provided a proof of security of Protocol 7.17 on the assumption that the hash functions are random oracles and using two computational assumptions known as the 'k-value modified bilinear inverse Diffie–Hellman' and the 'k-value collusion attack algorithm' assumptions.

Later, Wu and Tseng [743] proposed a protocol related to Protocol 7.17, intended for the same application scenario. The Wu and Tseng protocol uses SOK-type private keys instead of the SK-type keys used in Protocol 7.17. Both protocols avoid the use of pairings on the client side and have the same online computational requirements for the client A. A comparison of the two given by Wu and Tseng [743] indicates that they can save one exponentiation overall for the server compared with Protocol 7.17.

7.4.3 Identity-Based Key Agreement without Random Oracles

In all of the ID-based protocols which we have looked at so far, it has been necessary for the identity string to be hashed before being used in the protocol. In order for the security proof to work, this hash function is usually modelled as a random oracle. Such a requirement is typical of the ID-based encryption algorithms which were developed in the first decade of the explosion in research on pairings which started around the year 2000. Later, it was seen as an important research goal to remove random oracle assumptions wherever possible and some different solutions to that problem were found. It seems a natural goal to do the same for ID-based key exchange, but there has not been a lot of focus on this goal.

One protocol which achieves this goal is due to Tian *et al.* [711]. It is based on the ID-based encryption scheme of Gentry [297] and uses the same parameters and private keys. In Gentry's scheme, the public parameters consist of three values (g, g_1, h) for randomly chosen $g, h \in \mathbb{G}$ and $g_1 = g^{\alpha}$, where α is the master secret key. The private key of entity A is a pair (e_A, h_A), with $h_A = (hg^{-e_A})^{1/(\alpha - ID_A)}$. The protocol also makes use of a secure MAC. The protocol message exchange is shown in Protocol 7.18.

Through the first two messages of Protocol 7.18, A and B exchange what are ciphertexts of empty messages with Gentry's scheme. This allows them to obtain two shared secrets: $K_A = \hat{e}(g,h)^{r_A}$, generated implicitly by A, and $K_B = \hat{e}(g,h)^{r_B}$, generated implicitly by B. These keys are used firstly as keys for MACs which provide explicit authentication, and secondly to form the shared secret $\mathbf{Z} = \hat{e}(g,h)^{r_A r_B}$. The protocol is relatively expensive, requiring two pairings and five exponentiations on each side.

Tian *et al.* [711] provided a security proof in a Bellare–Rogaway-style model. Like Gentry's scheme, their security proof relies on a rather complex computa-

Shared information: Public parameters (g, g_1, h) with $g, h \in_R \mathbb{G}$ and $g_1 = g^{\alpha}$ where α is the master secret key
Information known to A: Private key (e_A, h_A) with $h_A = (hg^{-e_A})^{1/(\alpha - ID_A)}$.
Information known to B: Private key (e_B, h_B) with $h_B = (hg^{-e_B})^{1/(\alpha - ID_B)}$.

A		B
$r_A \in_R \mathbb{Z}_q$		
$z_{A,1} = (g_1 g^{-ID_B})^{r_A}$		
$z_{A,2} = \hat{e}(g,g)^{r_A}$		
	$\xrightarrow{ID_A, z_{A,1}, z_{A,2}}$	
		$r_B \in_R \mathbb{Z}_q$
$K_A = \hat{e}(g,h)^{r_A}$		$z_{B,1} = (g_1 g^{-ID_A})^{r_B}$
		$z_{B,2} = \hat{e}(g,g)^{r_B}$
		$K_A = \hat{e}(z_{A,1}, h_B) z_{A,2}^{e_B}$
	$\xleftarrow{ID_B, z_{B,1}, z_{B,2}, \mathsf{MAC}_{K_A}(ID_A, z_{A,1}, z_{A,2}, ID_B, z_{B,1}, z_{B,2})}$	
Verify MAC		
$K_B = \hat{e}(z_{B,1}, h_A) z_{B,2}^{e_A}$		
	$\xrightarrow{\mathsf{MAC}_{K_B}(ID_B, z_{B,1}, z_{B,2}, ID_A, z_{A,1}, z_{A,2})}$	
		$K_B = \hat{e}(g,h)^{r_B}$
$\mathbf{Z} = K_B^{r_A}$		Verify MAC
		$\mathbf{Z} = K_A^{r_B}$

Protocol 7.18: Protocol of Tian, Susilo, Ming and Wang

tional assumption known as the *truncated decisional ABDHE assumption*. They also claimed, without formal proof, that their protocol also achieves KCI resilience and forward secrecy.

We can divide Gentry's encryption scheme [297] into a key encapsulation method component and a data encapsulation method. This construction is therefore strongly related to the generic KEM-based construction examined in Sect. 5.8. Indeed, by using any identity-based KEM that is secure in the standard model, the construction in Sect. 5.8 can be used to construct alternative ID-based key agreement protocols without random oracles.

7.4.4 Comparison

Table 7.5 summarises the properties of the protocols which have been described in this section. Recall that the protocols in Table 7.5 include direct authentication information as a signature of some sort.

The protocols in this section have differing structures, so we have not tried to compare their message format except for noting the private key type. Each of them

Table 7.5: Summary of two-party ID-based protocols with explicit authentication. FS: forward secrecy; KCIR: key compromise impersonation resistance; ROM: random oracle model; Std: standard model

Protocol	**Private key**	**FS**	**KCIR**	**Proof**	**Computation on/offline**
BMP [139] (7.16)	SOK	KGC	No	ROM	$1E/1P+1E$
CHLS [206] (7.17)	SK	No	Yes	ROM	$-/3E$ (client), $3P+1E/-$ (server)
TSMW [711] (7.18)	Gentry	Yes	Yes	Std	$2P+3E/2E$

has a proof of security, with the TSMW protocol being the only explicit protocol we have listed which has a proof in the standard model.

As in Table 7.4, we have also summarised the computation of each party in Table 7.5. Again, we only record pairings (P) and exponentiations (E), and computational requirements are again divided into two parts, online and offline. For the CHLM protocol, the computation is different for the client (all can be done offline) and for the server (all online).

As mentioned at the start of this section, any of the protocols in Sect. 7.3 could be converted into a protocol with explicit authentication. This would make little difference to the computation on each side shown in Table 7.4, and in many cases would allow full strong forward secrecy to be achieved.

7.5 Identity-Based Protocols with Additional Properties

All of the protocols which we have examined in this chapter so far have the same basic infrastructure, namely a KGC which issues private keys to principals based on their identity information. There are a number of ways that this infrastructure can be extended, both for reasons of practicality and in order to provide new properties. In this section we consider a few of these ways, namely how to accommodate domains with different KGCs, how to incorporate user-generated keys, and how to allow more flexible ways to define which principals can participate. Finally, in this section we include a discussion of one-pass key establishment, which has a special significance for the identity-based case.

7.5.1 Using Multiple KGCs

In our descriptions of ID-based protocols we have assumed that all users rely on the same KGC to generate their private keys. In a large-scale system this is not practical. This issue has been noticed for ID-based cryptography in general since it was first described. There have been proposals for a hierarchical structure of KGCs to operate with identity-based encryption. Such structures spread the load on KGCs by allowing entities high in the hierarchy to issue keys for entities that act as KGCs for lower layers. There appears to have been little work on investigating the inclusion of hierarchies for ID-based key exchange in a generic fashion. However, some schemes have extensions allowing for multiple KGCs.

Chen and Kudla [194] presented a variant of Protocol 7.7 designed to accommodate the situation where two KGCs operate using the same public parameters (i.e. the same groups and generators) but different KGC master secrets. McCullagh and Baretto [531, Section 5] claimed a more efficient protocol. However, as pointed out before (see Sects. 7.3.1 and 7.3.8), both of these papers have limitations in their security proofs. Fujioka *et al.* [287] proposed a specific ID-based protocol using the key hierarchy of Gentry and Silverberg [300]. Guo and Zhang [338] considered a different but related setting in which ID-based and traditional PKI-based settings are combined.

One notable example of usage of multiple KGCs is the protocol of Schridde *et al.* [658], designed as a variant of Protocol 7.1 which allows users with different KGCs to agree on a secret. Although extra computation is required, it is not necessary for the two KGCs to communicate to set up their separate system parameters. Protocol 7.19 shows the protocol messages. The users need to employ the Chinese Remainder Theorem to compute new secret values s'_A and s'_B so that the session key derivation equation still works. The shared secret is a value in the integers modulo $n_1 n_2$, where n_1 is the modulus used by A's KGC and n_2 is the modulus used by B's KGC.

To see that both A and B compute the same value $\mathbf{Z} = (g_1 g_2)^{e_1 e_2 r_A r_B} \bmod n_1 n_2$ in Protocol 7.19, note that the value computed by A is

$$\begin{aligned}
\mathbf{Z} &= ((s'_B t'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 \\
&= ((s'_B)^{e_1 e_2} ID'_B)^{r_A} \cdot ((t'_B)^{e_1 e_2})^{r_A} \bmod n_1 n_2 \\
&= ((s'_B)^{e_1 e_2} ID'_B)^{r_A} \cdot ((g^{r_B})^{e_1 e_2})^{r_A} \bmod n_1 n_2 \\
&= ((s'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 \cdot \mathbf{Z}.
\end{aligned}$$

However,

$$\begin{aligned}
(s'_B)^{e_1 e_2} ID'_B \bmod n_2 &= s_B^{e_1 e_2} ID_B^{e_1} \bmod n_2 \\
&= (ID_B^{-1})^{e_1} (ID_B)^{e_1} \bmod n_2 \\
&= 1.
\end{aligned}$$

Similarly, $(s'_B)^{e_1 e_2} ID'_B \bmod n_1 = 1$, so that $(s'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 = 1$.

Gennaro *et al.* [294] presented a security proof for Protocol 7.19. They made a few adjustments to the protocol to ensure that the security proof holds. As for Protocol 7.1, it is necessary that the *ID* values are hashed and that the session key is obtained using a key derivation function which includes the exchanged messages as a session identifier. Moreover, Gennaro *et al.* noted that where the exponent $e_1 e_2$ is used in the derivation of $\mathbf{Z}$ in Protocol 7.19, it can be replaced by $E = \mathrm{lcm}(e_1, e_2)$ – this gives the same result and can be significantly more efficient. Another change is that the value of $\mathbf{Z}$ must be squared so that the shared secret becomes $\mathbf{Z} = g^{2E r_A r_B} \bmod n_1 n_2$. Finally, they defined g using the Chinese Remainder theorem so that $g \equiv g_1 \pmod{n_1}$ and $g \equiv g_2 \pmod{n_2}$.

Shared information: Public moduli n_1, n_2 and exponents e_1, e_2. Elements g_1, g_2 of high order in $\mathbb{Z}_{n_1}^*$ and $\mathbb{Z}_{n_2}^*$. Let $g = g_1 g_2$.
Using the Chinese Remainder Theorem, both parties compute ID'_A and ID'_B such that

$$\begin{aligned} ID'_A &\equiv ID_A^{e_2} \bmod n_1, \\ ID'_A &\equiv 1 \bmod n_2, \\ ID'_B &\equiv ID_B^{e_1} \bmod n_2, \\ ID'_B &\equiv 1 \bmod n_1. \end{aligned}$$

Information known to A: s_A such that $s_A^{e_1} = ID_A^{-1} \bmod n_1$. Using the Chinese Remainder Theorem, A computes s'_A such that

$$\begin{aligned} s'_A &\equiv s_A \bmod n_1, \\ s'_A &\equiv 1 \bmod n_2. \end{aligned}$$

Information known to B: s_B such that $s_B^{e_2} = ID_B^{-1} \bmod n_2$. Using the Chinese Remainder Theorem, B computes s'_B such that

$$\begin{aligned} s'_B &\equiv s_B \bmod n_2, \\ s'_B &\equiv 1 \bmod n_1. \end{aligned}$$

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_{n_1 n_2} & & \\
t'_A = g^{r_A} \bmod n_1 n_2 & \xrightarrow{s'_A t'_A} & r_B \in_R \mathbb{Z}_{n_1 n_2} \\
& \xleftarrow{s'_B t'_B} & t'_B = g^{r_B} \bmod n_1 n_2 \\
\mathbf{Z} = & & \mathbf{Z} = \\
((s'_B t'_B)^{e_1 e_2} ID'_B)^{r_A} \bmod n_1 n_2 & & ((s'_A t'_A)^{e_1 e_2} ID'_A)^{r_B} \bmod n_1 n_2
\end{array}$$

Both parties compute the secret
$\mathbf{Z} = (g_1 g_2)^{e_1 e_2 r_A r_B} \bmod n_1 n_2$

Protocol 7.19: Schridde *et al.* cross-domain identity-based protocol

7.5.2 Girault's Three Levels

Girault [305] introduced a three-level categorisation of key agreement based on a generalisation of identity-based schemes. In the schemes at level 1, the public key of the entity is the identity string ID_I, so these are exactly the normal identity-based schemes. At the higher levels, a value obtained from the KGC is used in combination with the partner's identity to derive a key that can only be calculated by a principal with the correct private key. We call such a value an *implicit certificate*. This allows the private keys to be kept secret from the KGC and can be regarded as a compromise between the basic identity-based scheme and conventional PKI-based schemes. The difference between levels 2 and 3 in Girault's classification depends on whether or not the owner of the public key can alone compute a valid public key (see Table 7.6).

Table 7.6: Girault's levels of extended identity-based schemes

Level	Properties	Example
1	KGC chooses, or can compute, private key.	Okamoto [589, 590], Protocol 7.1
2	KGC cannot find private key. Principal can generate contradictory public key.	Girault and Paillès [306]
3	Only KGC can generate valid public key.	Girault [305], Protocol 7.20

In order to understand the motivation behind the level 3 schemes, recall that a certificate of a public key is a signature by a third party on certain information that includes the value of the public key. A principal who receives such a certificate, including the owner of the private key, cannot use it to form another contradictory certificate (for example, one that shows a different public key). This is simply because only the third party can form new signatures. A malicious certification authority can generate its own private key and produce a certificate that shows that this private key belongs to any victim principal. This would allow the authority to masquerade as the principal. However, this certificate will contradict the real certificate of the principal. Because only the authority is able to produce a certificate, the two contradictory certificates can be used to show that the authority has cheated. In Girault's level 2 schemes, principals can choose their own private key and also use their private information to form new implicit certificates. Since both the authority and the principal can form contradictory certificates, it is impossible to tell which of them has cheated when two certificates appear. This situation is arguably little better than when the authority has access to the private key, since it is able to masquerade as any principal without being caught.

Girault and Paillès' protocol [306] was classified by Girault as level 2. There is a strong connection with Okamoto's Protocol 7.1. In Girault and Paillès' protocol the public key, y_A, of A satisfies $y_A^e ID_A = g^{-ex_A} \bmod n$, where g^{x_A} is given to the server but x_A is kept secret by A. But if we rearrange this we find that $y_A g^{x_A} = ID_A^{-d}$, so that A could calculate y_A herself when given the same private key from Okamoto's scheme. In the Girault and Paillès protocol A sends the message $y_A g^{x_A - r_A}$ to B. But when we rearrange this message it becomes $ID_A^{-d} g^{-r_A}$, which is the same as the corresponding message sent in Protocol 7.1 except for a change of sign. Similarly, the shared secrets are identical in the two protocols. We conclude that there is essentially no benefit in A choosing the extra secret x_A, even though the Okamoto scheme is at level 1 while the Girault and Paillès protocol is at level 2.

In 1991 Girault, [305] introduced *self-certified* public keys in order to avoid the limitations of level 2 schemes. These are keys that have an implicit certificate that can only be generated by the KGC, and consequently can be used in protocols to reach level 3 of Girault's classification. Girault's level 3 key agreement scheme [305] using self-certified keys has the same algebraic setting as that used for Okamoto's protocol (Protocol 7.1). An RSA modulus n and key pair e, d are chosen by the server, together with an element g of high order in $\mathbb{Z}_n^*$. When A registers to use the scheme, she chooses her private key x_A and provides g^{x_A} to the KGC, which calculates the self-certified public key $y_A = (g^{x_A} - ID_A)^d \bmod n$. (We have changed the sign of x_A from Girault's original in order to maintain our usual notation.) In order for the scheme to achieve level 3, it is essential that the KGC is unable to find x_A. Saeednia [641] has pointed out that a malicious server that chooses n may be able to find discrete logarithms relatively easily, and for this reason the size of n should preferably be 2048 bits. In addition the KGC should provide a proof that n is the product of two safe primes; Camenisch and Michels [175] provided a method to achieve such a proof.

In Girault's original paper [305], he only suggested using the self-certified keys to produce an authenticated static shared key. If the public keys are already available, this can be achieved with no message exchanges: A calculates $(y_B^e + ID_B)^{x_A}$ in order to produce $\mathbf{Z} = g^{x_A x_B}$ and B calculates the analogous value. However, since the public key is simply a way to find an implicitly certified key for each party, any of the MTI protocols (see Sect. 5.3) can be modified to provide a dynamic protocol. For example, Protocol 7.20 shows a version modified from MTI A(0), as suggested by Rueppel and van Oorschot [637]. The shared key is $\mathbf{Z} = g^{r_A x_B + r_B x_A}$. The usual extra checks should be made in order to avoid potential attacks, as discussed in Sect. 5.3. Analogues of the other MTI protocols can be made similarly, replacing y_A in the MTI original with $y_A^e + ID_A$ in the Girault version, and similarly for y_B.

Shared information: Public modulus n and exponent e. Element g of high order in $\mathbb{Z}_n^*$.
Information known to A: x_A with $y_A^e + ID_A = g^{x_A} \bmod n$, y_B.
Information known to B: x_B with $y_B^e + ID_B = g^{x_B} \bmod n$, y_A.

$$\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_n & & \\
t_A = g^{r_A} & & \\
 & \xrightarrow{t_A} & r_B \in_R \mathbb{Z}_n \\
 & \xleftarrow{t_B} & t_B = g^{r_B} \\
\mathbf{Z} = t_B^{x_A}(y_B^e + ID_B)^{r_A} & & \mathbf{Z} = t_A^{x_B}(y_A^e + ID_A)^{r_B}
\end{array}$$

Protocol 7.20: Girault's identity-based protocol, adapted by Rueppel and van Oorschot

Just like the MTI A(0) protocol, Protocol 7.20 does not provide forward secrecy: knowledge of x_A and x_B allows an adversary to compute $\mathbf{Z}$. However, KCI resistance does seem to be provided, but there is currently no proof of that, nor indeed that the protocol is secure at all.

Nyberg and Rueppel [585] suggested using their signature with message recovery, previously discussed in Sect. 5.2.2, as the basis of identity-based key agreement. The idea is to make the public key of each principal A equal to the signature of $g^{x_A} ID_A^{-1}$. After recovery of the message from the signature, the implicitly certified public key can be recovered by multiplying by ID_A. With this basis, any of the MTI-style protocols can be applied, in the same manner as in Protocol 7.20. Sakazaki *et al.* [648] explored use of elliptic curves and other variations in order to make the basic idea more efficient.

7.5.3 Certificateless Key Agreement

Despite the efficiency advantages of identity-based key exchange, it has the significant drawback that it has the so-called ‘escrow’ property: the KGC can obtain the private key of any party. If the protocol has KGC forward secrecy, this may not be as serious a problem as it would otherwise be, because the KGC will be unable to recover session keys from sessions in which it was not active. But the KGC can still masquerade freely as any user. Furthermore, if a malicious KGC is able to obtain ephemeral keys used in the session then it will be able to obtain the agreed session key (since it will know all the secrets).

For the above reasons there is value in considering the *certificateless* setting. Certificateless cryptography was introduced by Al-Riyami and Paterson [24] to provide some of the benefits of identity-based cryptography without allowing the KGC to know the secrets of users. To achieve this, users have two parts to their private key: one is issued by the KGC and the other chosen by the user. A user’s public key must be known to communication partners but is not certified. This can be seen as a similar idea to Girault’s level 3 scheme discussed in Sect. 7.5.2.

Since the idea of Al-Riyami and Paterson [24] was introduced in 2003, many certificateless cryptographic primitives have been designed, including encryption and signatures. There have also been several certificateless key agreement protocols; indeed, the first was in the original Al-Riyami–Paterson paper. However, it was not until 2009 that a formal model for security was defined [497, 702]. In such a model, we expect the adversary to obtain the KGC private key as well as the ephemeral private keys of the parties and still be unable to obtain the session key. This shows that the model is stronger than what can be achieved in the identity-based setting.

Swanson and Jao [702] showed that all of the protocols published before 2009 are insecure in a strong model of security. Lippold *et al.* [497] seem to have been the first to propose a certificateless key agreement protocol that is secure in a strong security model. Their protocol remains secure even if any two of the three keys (the identity-based private key, the user-selected private key, and the ephemeral private key) become compromised. The security proof is in the random oracle model and relies on the computational Diffie–Hellman assumption. However, the protocol is relatively expensive, requiring 10 elliptic curve pairings for each party. Slightly later, Lippold *et al.* [496] proposed another certificateless key exchange protocol which is secure in the standard model and is more efficient than their previous proposal. It is essentially the same as Protocol 5.43, where the key encapsulation mechanism used is a certificateless one. Yang and Tan [750] have proposed an even more efficient protocol without relying on pairings; it relies instead on the gap Diffie–Hellman problem.

7.5.4 Protocols with Generalised Policies

In the past few years, identity-based cryptography has been generalised in a few different ways to allow more flexible and expressive properties to be specified. For example, identity-based encryption can be generalised so that ciphertexts can be decrypted by any user who possesses certain properties, such as being a member of a group or being above a certain age. Identity-based encryption is then a special case of this, where the specific property required for decryption is having an identity equal to a specific value. Various flavours of generalisation have been defined, including attribute-based cryptography, predicate cryptography, and, more generally, functional cryptography [124]. At the time of writing, this is still a developing research area where many new results can be expected in the coming years. Here we just mention some early contributions to applying these generalisation to key exchange protocols.

At the same conference in 2010, papers were published on *predicate-based key exchange* by Birkett and Stebila [104] and on *attribute-based key exchange* by Gorantla *et al.* [327]. These two papers are related in that their intent is to generalise the access policy to the shared secret, but they also have significant differences. The first is concerned with two-party key exchange, and one main requirement is to hide the properties (attributes) held by the participants. The second presents a group key exchange protocol in which any user can participate by holding the necessary properties, while keeping the user identity secret. Either of these approaches may be useful, depending on the application scenario. Other papers have built on these ideas [694, 765]. A related notion is credential-based key exchange, introduced by Camenisch *et al.* [174].

7.5.5 One-Pass Identity-Based Protocols

One of the major benefits of identity-based cryptography is that users do not need to access keying material for communication partners before applying cryptographic processing based on the partner's identity. When using conventional (certificate-based) public keys, a user who wishes to encrypt information for a chosen recipient must know the correct public key of the recipient. In contrast, identity-based encryption requires only the recipient's identity (and the public parameters) to be known. In this sense, identity-based encryption is more useful than identity-based signatures. A signer can append a certificate to a conventional public signature to make it identity-based in the sense that only the public parameters (which can include the certifier's public key) and the identity are required to verify the signature.

We can develop this argument in the context of key exchange protocols to arrive at the conclusion that many key exchange protocols with two (or more) messages can be made identity-based simply by adding certificates to each of the (first two) messages. This will always work as long as the first message from the initiator does not depend on the public key of the responder. Of course, this does not mean that such protocols will be efficient or have other desirable properties. Also, it may be desirable to share public keys and parameters from an identity-based infrastructure used for encryption and reuse them for key exchange. Thus we certainly do not claim that the two- and three-message protocols explored in this chapter are not interesting. However, we can say that the relationship of one-pass key exchange to two-pass key exchange is similar to the relationship of identity-based encryption to identity-based signatures. The latter can always be obtained from conventional primitives, while the former cannot.

It is reasonable to expect that protocols with only one message will not achieve as high a level of security as those with more messages. Noticing that an adversary who obtains the recipient's private key has the same information as the recipient during the protocol, we can see that one-pass protocols cannot achieve full forward secrecy. The best that can be achieved is *sender forward secrecy* so that compromise of the sender's private key will not compromise the session key. Similarly, an adversary who can obtain the recipient's private key can impersonate the sender unless the single message is explicitly authenticated (and the adversary can always replay a message), so that key compromise impersonation to the recipient cannot generally be prevented.

Okamoto *et al.* [592] described two protocols and argued that their protocols provide sender forward secrecy and security against key compromise impersonation to the sender. Around the same time, Wang's protocol [731, 732], which we saw in Sect. 7.3.7, was published. Wang pointed out that Protocol 7.12 can be adapted to a one-pass protocol by setting $r_B = 1$ and $w_B = q_B$ so that the message w_B sent from B to A can be removed. However, none of these earlier protocols carries any proof of security. Gorantla *et al.* [325] seem to have been the first to provide a design for one-pass identity-based key exchange with a security proof. Their protocol is shown as Protocol 7.21.

$$
\begin{array}{ccc}
A & & B \\
r_A \in_R \mathbb{Z}_q & & \\
w_A = q_A^{r_A} & & \\
 & \xrightarrow{w_A} & \\
s = H_2(w_A, ID_A, ID_B) & & s = H_2(w_A, ID_A, ID_B) \\
\mathbf{Z} = \hat{e}(d_A^{r_A+s}, q_B) & & \mathbf{Z} = \hat{e}(s_B, w_A q_A^s)
\end{array}
$$

Protocol 7.21: Protocol of Gorantla, Boyd, and González-Nieto

There is a strong similarity between Protocol 7.21 and Protocol 7.12 by Wang. Indeed, Protocol 7.21 is a simplified version of Wang's one-pass version of Protocol 7.12 mentioned above. Gorantla *et al.* [325] provided a security proof of Protocol 7.21 in the random oracle model, assuming hardness of the BDH problem.

It is a reasonable question to ask whether there is any real difference between a one-pass key exchange protocol and a hybrid encryption scheme. In both cases, a key is set up which is then used to protect other exchanged data, and this key can depend on both the sender's and the recipient's long-term keys. This question was investigated by Gorantla *et al.* [324] who concluded that there is a duality between one-pass key exchange and a primitive known as a signcryption KEM. With suitable assumptions, it is possible to transform a one-pass key exchange protocol into a signcryption KEM and vice versa.

7.6 Conclusion

There has been a huge amount of research on identity-based cryptography, mostly since the year 2000. Identity-based key exchange has been a significant part of this and it may be fair to say that the area is reasonably mature, at least with regard to pairing-based solutions. We have a number of well-understood protocols which are practically efficient and with security proofs based on widely accepted computational assumptions.

One direction where we may see future developments is in lattice-based solutions. Lattices appear to be a more promising long-term foundation for identity-based cryptography. One reason for this is the likelihood that lattice-based algorithms will be able to withstand attacks from quantum computers, which threaten to undermine many current cryptographic technologies, including pairings. The other direction where we anticipate new results is in protocols whose principals are defined by something more flexible than identity. These can be expected to emerge from research into generalised forms of cryptographic primitives, particularly in the area of functional cryptography.