Security in Key Agreement: Two-Party Certificateless Schemes

by

Colleen Marie Swanson

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2008

Abstract

The main goal of cryptography is to enable secure communication over a public channel; often a secret shared among the communicating parties is used to achieve this. The process by which these parties agree on such a shared secret is called key agreement. In this thesis, we focus on two-party key agreement protocols in the public-key setting and study the various methods used to establish and validate public keys. We pay particular attention to certificateless key agreement schemes and attempt to formalize a relevant notion of security. To that end, we give a possible extension of the existing extended Canetti-Krawzcyk security model applicable to the certificateless setting. We observe that none of the certificateless protocols we have seen in the literature are secure in this model; it is an open question whether such schemes exist. We analyze several published certificateless key agreement protocols, demonstrating the existence of key compromise impersonation attacks and even a man-in-the-middle attack in one case, contrary to the claims of the authors. We also briefly describe weaknesses exhibited by these protocols in the context of our suggested security model.

Acknowledgements

I would like to acknowledge my advisor, David Jao, without whom this thesis would certainly not exist, and my readers, Alfred Menezes and Edlyn Teske, for all the helpful comments and discussions.

To my friends and family, scattered about the globe as you might be, I would like to extend my gratitude for your help and support. A very special thanks to Farshid Hajir—your friendship and your mastery of witty instant messaging has not gone unappreciated.

For Jason,
who has been a driving force of my life
and without whom
I would never have begun my studies in cryptography.

Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis Outline

The structure of this thesis is as follows. In the remainder of Chapter 1, we establish notation, introduce common cryptographic primitives, including hash functions and pairings, and discuss standard number theoretic assumptions. Chapter 2 discusses public-key cryptography and reviews the various methods used to produce and verify public keys. Chapter 3 presents a summary of key agreement protocols and common attacks and concludes with representative examples from the different public-key settings. We discuss the notion of provable security and suggest a possible extension of the eCK model relevant to certificateless key agreement protocols in Chapter 4. Chapter 5 mentions related work on the subject of key compromise impersonation attacks and Chapter 6 gives an attack analysis of several certificateless key agreement protocols from the literature. Lastly, we conclude in Chapter 7.

1.2 Notation

We use $\{0,1\}^*$ to denote the set of finite binary strings and $\{0,1\}^{\leq L}$ (where $L \in \mathbb{N}$) to denote the set of binary strings of length at most L. The symbol "$\in_R$" means "selected uniformly at random from." We write G^* for the set of non-identity elements of a group G. By 1_G, we mean the identity element of the group G.

1.3 Cryptographic Primitives

The main goal of cryptography is to achieve secure communication over an insecure channel. A cryptosystem, in particular, provides a method by which two parties, Alice and Bob, can send private messages to each other over a public channel. Say Alice wishes to send Bob some kind of message m, which we call the *plaintext*; she does so by disguising, or *encrypting*, m using some predetermined key and sending the resulting *ciphertext* over the channel. Bob can then *decrypt* (using either the same or a different key) and read the plaintext, but any eavesdroppers to the conversation will not have the decryption key and so will be unable to understand the message. We give the formal definition below.

Definition 1.3.1. A *cryptosystem* is a 5-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where

1. $\mathcal{P}$ is a set of possible *plaintexts*;
2. $\mathcal{C}$ is a set of possible *ciphertexts*;
3. $\mathcal{K}$ is the *keyspace*, a finite set of possible keys;
4. For each $k \in \mathcal{K}$ there is a corresponding encryption function $E_k \colon \mathcal{P} \to \mathcal{C} \in \mathcal{E}$ and decryption function $D_k \colon \mathcal{C} \to \mathcal{P} \in \mathcal{D}$, such that for $x \in \mathcal{P}$, we have $D_k(E_k(x)) = x$.

The encryption and decryption functions E_k and D_k might be the same, or perhaps easy to derive from each other, as in the case of *symmetric cryptosystems*, or it could be that information about E_k does not reveal D_k, as in *asymmetric* or *public-key cryptosystems*. We discuss these notions further in Chapter 2, focusing on current key generation methods used in public-key cryptography.

Other important cryptographic primitives include *signature schemes*, in which a party can electronically sign a document, a special type of function called a *hash function*, and *key agreement protocols*, in which two parties agree on a shared secret. We introduce the notion of hash functions in Section 1.4 and devote Chapter 3 to key agreement protocols. We will treat digital signatures only informally, observing that these signatures serve much the same function as ordinary signatures on paper. Digital signatures "bind" the signer to the given message and can be verified as valid and not forged. A standard example of a digital signature is the RSA signature; we give a sketch of this scheme in Section 1.5.1 and refer the reader to [30, 41] for more on the subject of signature schemes.

1.4 Functions

We require the following general definitions concerning functions.

Definition 1.4.1. We say that a real-valued function $\epsilon(k)$ is *negligible* in k if for all $c > 0$, there exists $k_c > 0$ such that $k > k_c$ implies $\epsilon(k) < \frac{1}{k^c}$. We say a function that is not negligible is *non-negligible.*

We now turn to the question of how easy it is to compute a given function. In general (and unless otherwise specified), we classify computational problems as "easy" (or "hard") based on the existence (or non-existence, respectively) of a polynomially bounded algorithm that solves an arbitrary instance of the problem with high probability, and refer to "hard" problems as *computationally infeasible*. By *high probability*, we mean that the probability the algorithm succeeds is non-negligibly greater than the corresponding probability of success of a random function; if the codomain is infinite, the probability of success must be non-negligibly greater than zero. In the context of computing functions, this gives the following definition.

Definition 1.4.2. We say that a function f is *efficiently computable* if there exists a polynomially bounded algorithm that computes f with high probability.

1.4.1 Hash Functions

Hash functions are a core component of many other primitives in cryptography. Such functions are often used to provide a short "fingerprint" of data—by recalculating the hash value of a piece of data and seeing if it has changed, we can check whether the data has been altered. We will use hash functions in key agreement protocols both as a final measure to increase the security of the resulting key and to uniquely identify an entity with an element of a given group. That is, we will map an identifying binary string of arbitrary length to an appropriate group in a way that preserves uniqueness, i.e., we do not want this hashed "identity" to be used for more than one entity. We give the formal definition of a hash function below.

Definition 1.4.3. A *hash function* h is an efficiently computable function of the form $h\colon \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is a finite set such that $|\mathcal{X}| \geq |\mathcal{Y}|$.

We note that the domain $\mathcal{X}$ may or may not be finite; in the finite case, h is sometimes referred to as a *compression function*. If we use the output of a hash function h as some kind of key, we will often refer to h as a *key derivation function (*kdf*)*. There is also the notion of a *keyed hash function*, which is essentially an element of a family of hash functions parametrized by some key k. Hash functions that are not parametrized by a given key (such as in the most general definition above) are referred to as *unkeyed hash functions*.

Definition 1.4.4. A *keyed hash family* $\mathcal{H}$ with parameters $(\mathcal{X}, \mathcal{Y}, \mathcal{K})$ is a family of hash functions, where

1. $\mathcal{K}$ is the keyspace;
2. For each $k \in \mathcal{K}$, there is a hash function $h_k \in \mathcal{H}$ of the form $h_k\colon \mathcal{X} \to \mathcal{Y}$, which we refer to as a *keyed hash function*.

Definition 1.4.5. A *message authentication code (MAC)* is a keyed hash family $\mathcal{H}$ with corresponding parameters $(\mathcal{X}, \mathcal{Y}, \mathcal{K})$ that satisfies the following property, known as *computation-resistance*:

- Given zero or more pairs of the form $(x_i, h_k(x_i)) \in \mathcal{X} \times \mathcal{Y}$, it is computationally infeasible to compute any pair $(x, h_k(x))$ for any $x \neq x_i$ for all i.

We now list desirable security attributes of a hash function $h\colon \mathcal{X} \to \mathcal{Y}$. Hash functions suitable for use in key agreement protocols and data integrity applications (as mentioned in the beginning of this section) will generally satisfy these properties. As no formal definitions for these properties exist, our descriptions are necessarily vague. In particular, we do not make our use of the word "difficult" precise; we intend to convey that an adversary with *reasonable* abilities (for instance, with regard to time or space requirements) should be unable to successfully solve the given problem.

- *Preimage resistance*: We say h is preimage resistant or *one-way* if given $y \in_R \mathcal{Y}$ it is difficult to find $x \in \mathcal{X}$ such that $h(x) = y$.
- *Second preimage resistance*: We say h is second preimage resistant if given $x \in_R \mathcal{X}$, it is difficult to find $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.
- *Collision resistance*: We say h is collision resistant if it is difficult to find $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$. We note that collision resistance implies second preimage resistance, but not necessarily preimage resistance.

Security proofs for cryptographic primitives, which we discuss in Chapter 4, often require that any hash functions used be modeled by *random oracles*, or idealized hash functions, which essentially serve to prevent an adversary from using the underlying structure of the hash function to his advantage. The random oracle will, given an arbitrary input, either produce an output selected uniformly at random or if it has seen the input before, output the same value as it did previously. More precisely, a random oracle $h\colon \mathcal{X} \to \mathcal{Y}$ has the property that for each new $x \in \mathcal{X}$, the output $h(x)$ is independently and uniformly selected at random from $\mathcal{Y}$, so any element of $\mathcal{Y}$ is equally likely. The random oracle behaves like a hash function in the sense that if $y = h(x)$ has already been computed, h will output y whenever x is given as the input. The use of random oracles is somewhat controversial in the cryptographic community—as no random oracles exist, the real-world security implications of proofs using them (said to be in the *random oracle model*) are not necessarily clear [10].

1.4.2 Pairings

We note that pairings come in two flavors, the first of the form $e\colon G_1 \times G_1 \to G_T$ where G_1 and G_T are groups of prime order q, and the second of the form $e\colon G_1 \times G_2 \to G_T$, where G_1, G_2, and G_T are groups of prime order q. The group G_T is often referred to as the "target" group, hence the use of the subscript "T." Pairings of the first form are a special case of the latter; we choose to address only this type in detail as the protocols we analyze all assume a pairing of this form. In practice, G_1 and G_2 are groups of points on an elliptic curve over a finite field, and G_T is a subgroup of a multiplicative group of a related finite field. Pairings are generally based on the Weil and Tate pairings and those of the first form are implemented using supersingular curves. We refer the reader to [17, 39] for details on elliptic curves and the use of pairings in cryptography.

We formally define a pairing as follows.

Definition 1.4.6. Let G_1 be a cyclic additive group of prime order q and G_T be a cyclic multiplicative group also of order q. An *admissible pairing* is a map $e\colon G_1 \times G_1 \to G_T$ with the following properties.

1. *Bilinearity*: $\forall P, Q, R \in G_1$ we have $e(P+Q, R) = e(P, R)e(Q, R)$ and $e(P, Q+R) = e(P, Q)e(P, R)$.
2. *Non-degeneracy*: For all $P \neq 1_{G_1}$, we have $e(P, P) \neq 1_{G_T}$.
3. The pairing is efficiently computable.

The bilinearity property implies for any $a, b \in \mathbb{Z}_q$ and $P, Q \in G_1$, we have

$$e(aP, bQ) = e(abP, Q) = e(P, abQ) = e(P, Q)^{ab}$$

and remembering G is cyclic, $e(P, Q) = e(Q, P)$. Also worth noting is that if P is a generator of G_1, $e(P, P)$ is a generator of G_T.

1.5 Number Theoretic Assumptions

In this section we introduce some standard hard problems in cryptography, particularly those which are used in Diffie-Hellman style key agreement schemes (Section 3.1). As before, we say a computational problem is hard, or computationally infeasible, if there exists no polynomially bounded algorithm that solves an arbitrary instance of the problem. We say a computational problem π *reduces* to a problem π' if there exists a polynomially bounded algorithm solving π that uses an oracle (or hypothetical subroutine) that solves π'. If π reduces to π' and π' reduces to π, we say the two problems are *computationally equivalent.*

1.5.1 RSA and Factoring

We first briefly describe the RSA encryption and signature schemes; for more details we refer the reader to [41, 30]. An *RSA modulus* is a number $n = pq$ for large, distinct primes p and q. In the RSA cryptosystem, the public encryption key e and private decryption key d satisfy $ed \equiv 1 \pmod{\phi(n)}$. Here $\phi(n) = (p-1)(q-1)$ is the number of positive integers less than and relatively prime to n. Given a plaintext message $m \in \mathbb{Z}_n$, the corresponding ciphertext is $c = m^e \pmod{n}$. Decryption works by taking $c^d \pmod{n}$, which is equivalent to m. Alternatively, we can sign a piece of data $x \in \mathbb{Z}_n$ by computing the signature as $\sigma = x^d \pmod{n}$, which can be verified by checking $\sigma^e = x$; this signature scheme is often referred to as *textbook RSA* [31]. In practice, we improve the security of the system by either hashing m, as in *RSA with appendix*, or ensuring that m has a lot of redundancy, as in *RSA with message recovery* [8].

A standard assumption is that factoring is a hard problem. That is, given $n = pq$ for large primes p and q, it is computationally infeasible to find p and q. This problem has been shown to be computationally equivalent to the problem of finding the decryption key d, given an RSA modulus n and encryption key e. The *RSA problem* is the problem of finding $m^d \pmod{n}$ given n, e, and m; it is an open problem whether or not this is equivalent to factoring. The corresponding *RSA assumption* is that there exists no polynomially bounded algorithm solving the RSA problem. This is the assumption used in Girault's self-certified key agreement scheme, described in Section 3.6.5.

1.5.2 Diffie-Hellman and Related Problems

Let G denote a multiplicatively-written cyclic group of order n. We write $G = \langle g \rangle$ to specify that G is generated by g. We can define discrete logarithms over general cyclic groups as follows.

Definition 1.5.1. For any $h \in G$, the *discrete logarithm* of h with respect to the generator g, denoted $\log_g h$, is the unique $x \in \mathbb{Z}_n$ satisfying $h = g^x$. The problem of finding $\log_g h$ is called the *discrete logarithm problem (DLP).*

Groups in which the discrete logarithm problem is hard will be of particular interest to us. More precisely, we say that the *discrete logarithm assumption* holds in G if there exists no polynomially bounded algorithm that can solve an arbitrary instance of the DLP. We have the following list of related problems and corresponding assumptions.

Definition 1.5.2. Given $h_1 = g^x \in G$ and $h_2 = g^y \in G$, the *computational Diffie-Hellman problem (CDH)* is the problem of computing $h_3 = g^{xy} \in G$. The *CDH assumption* holds in G if there exists no polynomially bounded algorithm that can solve an arbitrary instance of the CDH problem.

Definition 1.5.3. Given a triple $(g^x, g^y, g^z) \in G^3$, the *decisional Diffie-Hellman problem (DDH)* is the problem of deciding if $g^z = g^{xy}$. If this condition holds, we say the triple is a *DDH triple.* The group G satisfies the *DDH assumption* if there exists no polynomially bounded algorithm that can solve an arbitrary instance of the DDH problem.

We observe that the DDH problem reduces to the CDH problem, since if we can compute g^{xy} given g^x and g^y, we can obviously decide whether (g^x, g^y, g^z) is a DDH triple. Similarly, the CDH problem reduces to the DLP, since if we can find x given g^x and y given g^y, we can easily computer g^{xy}.

We have a special variant of the computational Diffie-Hellman problem for pairings on elliptic curves, namely the Bilinear Diffie-Hellman problem, defined below. As before, we assume $e\colon G_1 \times G_1 \to G_T$ is an admissible pairing, where G_1 is an additive cyclic group and G_T is a multiplicative cyclic group. For practical purposes, we will generally assume the DLP is hard in both G_1 and G_T. We point out that the DLP in G_1 reduces to the DLP in G_T, since if we have $x = \log_{e(P,P)} e(Q, P)$ in G_T, then $x = \log_P Q$ in G_1, where $P, Q \in G_1$.

Definition 1.5.4. Given $P, aP, bP, cP \in G_1$ for $a, b, c \in \mathbb{Z}_q^*$, the *bilinear Diffie-Hellman problem (BDH)* is the problem of computing $e(P, P)^{abc} \in G_T$. The *BDH assumption* holds if there exists no polynomially bounded algorithm that can solve an arbitrary instance of the BDH problem.

Definition 1.5.5. Given $P, aP, bP, cP \in G_1$ for $a, b, c \in \mathbb{Z}_q^*$ and $z \in G_T$, the *decisional Bilinear Diffie-Hellman problem (DBDH)* is the problem of deciding whether $z = e(P, P)^{abc}$. The *DBDH assumption* holds if there exists no polynomially bounded algorithm that can solve an arbitrary instance of the DBDH problem.

Chapter 2

Public-Key Cryptography

We distinguish between *symmetric cryptography*, which involves a single private key shared among participants, and *asymmetric* or *public-key cryptography (PKC)*, in which each party has a unique public-key pair. Rather than using the same key for encryption and decryption, in asymmetric cryptosystems the public key is used for encryption and the private key for decryption. The obvious advantage of the latter system is that participants do not need a previously established secret in order to interact. Often public/private key pairs can remain the same for long periods of time, whereas symmetric keys should be changed frequently in order to stave off attacks. In practice, however, symmetric cryptosystems tend to be much more efficient, both in terms of throughput rates and key sizes. That being said, cryptosystems are frequently designed to take advantage of the positive aspects of both, by using public-key cryptography to establish a shared secret for use in a symmetric scheme.

A significant problem in public-key cryptography is verification of the authenticity of public keys. For obvious reasons, the security of any public-key cryptosystem rests heavily on users being assured that they cannot be fooled into using a false public key. We spend the remainder of the chapter discussing various methods to accomplish this, namely the traditional public-key infrastructure, identity-based, certificate-based, certificateless, and self-certified public keys.

2.1 Public-Key Infrastructure

The traditional method of authenticating public keys is through a *public-key infrastructure (PKI)*. A *certification authority (CA)* is a trusted third party responsible for establishment and verification of the authenticity of public keys. In a PKI system, users acquire (via a secure channel) the public key of the CA. The CA distributes *certificates*, which consist of a *data part* and a *signature part*. The data part generally contains a string identifying a particular user as well as that user's public key, and might contain additional information. The signature part is the

CA's digital signature on the data part. In this way, the certificate binds a user to his public key, as any user can simply check (by using the CA's public key) that the signature part is a valid signature on the associated data part.

In addition to creating certificates, the CA might be responsible for creating a public-key pair and sending a copy of this information to the specified user over a secure channel. Alternatively, users can create their own public-key pair and transfer their public keys to the CA in a secure manner, who then creates the necessary certificates. In both cases it is necessary for the CA to verify the identity of the user before granting the corresponding public-key certificates. If the CA does not know the private key corresponding to a given public key, it must verify that the associated user does know this information, as otherwise a dishonest user might claim someone else's public key as their own.

It is clear that a great deal of trust is invested in the CA. Users take on faith that the CA behaves honestly, i.e., that the CA does not create false certificates and verifies the identity of a given user before granting a certificate. We emphasize that the CA is the only entity capable of producing certificates, so the existence of two valid certificates for the same identity implies that the CA issued both. Since we generally assume that users have unique public keys for a particular application, this scenario implies the CA behaved dishonestly. In any case, there is no possibility of other users creating false or multiple public keys (either for themselves or others) *without the involvement of the CA*. By construction, we give the CA the responsibility of checking that these public keys should be certified, and we can therefore blame the CA if false certificates are issued. However, we observe that it is possible for the CA to *temporarily* replace public keys (and forge a corresponding certificate) in order to impersonate a user, and then reset the value afterward to avoid detection. The success of this attack depends heavily on timing. For instance, suppose Bob wishes to send a message to Alice for the first time. Assuming he has never looked up Alice's public key and the corresponding certificate, he will not suspect anything amiss if the CA replaces Alice's public information, and the CA will then be able to distinguish any ciphertext Bob produces. While the CA could be caught immediately if Alice herself happens to check her public information during the duration of the attack, or later if Bob rechecks the public-key directory and suspects foul play rather than a legitimate key change, the CA's bad behavior could very well remain undetected.

Several problems have been identified with PKI, perhaps the most obvious of which are the enormous storage and upkeep requirements (both for publishing public-key directories and certificates). It is necessary for the CA to have the ability to *revoke* certificates, for reasons including, but not limited to, a compromise of the user's private data. One option for this is the distribution of lists of revoked certificates, called *certificate revocation lists (CRLs)*, which of course increases the system management burden. Another issue with CRLs is time—the CA must distribute the CRLs to all parties, who are then responsible for checking whether a given certificate has expired. This distribution is expensive (especially if the CRL and/or number of users is large), and may not occur often enough to adequately warn users of compromised public keys. Denial-of-service (DoS) attacks are easy to launch by repeatedly requesting CRLs, and even if this does not occur, honest CRL requests are frequently enough to cause network congestion. Another option, the *online certificate status protocol (OCSP)*, requires the CA to respond to status queries by the online signing of the certificate's current status. Although this reduces communication costs to one signature per query (rather than an entire list of revoked certificates), computation costs go up accordingly. This solution also leaves the system vulnerable to denial-of-service attacks, particularly if the CA is centralized, whereas a decentralized CA is only as secure as its individual servers [18].

Other problems include PKIs with multiple CAs, which are generally arranged in a hierarchical trust model. CAs lower in the trust model are issued *signer certificates* by higher CAs, and in turn these CAs can issue certificates at will. Users create *certificate chains* to detect forgery, whereby in addition to checking the validity of a certificate issued by a particular CA, the user must check the signer certificates of all the CAs above until a trusted (root) CA is reached. Needless to say, this can become quite complicated, especially if multiple hierarchies are involved, and a user authenticated by a CA in one hierarchy wishes to check the public key of a user authenticated by a CA in another. These CAs can *cross-certify* each other by signing each other's certificates, but this leads to multiple possible authentication paths to a trusted CA. In addition to this structural question, it seems that X.509 certificates, which are commonly used in practice, suffer from several problems, among them the possibility of multiple certificates with the same name and little flexibility. We refer the reader to [21, 30] for more details on problems associated with public-key certificates and PKI.

2.2 Identity-based Cryptography

The concept of *identity-based public-key cryptography (ID-PKC)* was first defined by Shamir [36] in 1984. The basic idea is to generate public keys from strings of information that uniquely identify the intended holder, for example the user's email, name, or physical address; this is generally done via hash functions. Once a public key has been created, the associated user can go to a *Key Generation Center (KGC)*, whose responsibility (similar to that of a CA) is to send the corresponding private key to the user via a secure channel. In this way, a recipient of an encrypted message doesn't even have to be registered with the KGC beforehand; he can simply go to the KGC to get the needed private key and then decrypt his message. This makes it very easy to send messages, and greatly simplifies key management, as the need for certificates is completely removed. Indeed, the KGC is only needed to issue new private keys and can even be closed for indefinite periods without affecting the normal workings of the network. The main fault, however, is that the KGC by definition has *key escrow*, or access to all users' private keys, so users must necessarily place a high level of trust in the KGC. In addition, the system requires the KGC to have access to a secure channel over which it distributes private keys.

The notion of ID-based systems gained popularity after 2001, when Boneh and Franklin constructed efficient, provably secure ID-based primitives relying on the Bilinear Diffie-Hellman problem over elliptic curve pairings [7]. There are myriad examples of ID-based primitives—we shall see a key agreement protocol in Section 3.6.2. Other examples may be found in [40, 14, 32]. Several suggestions exist for removing the escrow property, but (as we shall see in an example) these have limited success against an active KGC and add to the computational and communication complexity. We also note that the KGC's access to private keys implies that signature schemes in the ID-based setting do not offer non-repudiation. Since the KGC is always capable of forgery, a valid signature does not guarantee the user actually signed the given data.

2.3 Self-Certified Public-Key Cryptography

Self-certified public keys were introduced by Girault [19] in the hopes of reducing the amount of storage and computation by third parties. Both of his presented schemes require less of the CA—there is no key escrow, no need for hash functions in computing public keys, and no need for a secure channel between the CA and user. As usual, users are associated with a 3-tuple (ID, s, P), where ID is the user's identity, s the user-chosen private key, and P the public key. The public key is self-certifying in the sense that it doubles as a certificate; that is, the CA issues a certificate on a user's identity, which is then used as the public key. This is unlike traditional PKI, where users must have separate certificates validating their public keys. As this public key cannot be immediately derived from the user's identity, we note that this idea varies from ID-based systems as well.

Girault presents two possibilities, one of which relies on the hardness of factoring large integers and the discrete logarithm problem, and the other relying only on the latter. The first we explore in detail in Section 3.6.5. In the second, the pair (s, P) is the CA's ElGamal signature on the user's identity ID, with the interesting property that s remains unknown to the CA; we refer the reader to the original paper [19] for details. Related work has been done by Saeednia [34, 33], who has invented several self-certified schemes. His version of self-certified keys does not require the public key to actually *be* the certificate issued by the CA. Rather, the CA issues a certificate w, called a *witness*, on the user's identity and user-chosen public key (ID, P). The public key P must be easily computable from w, ID, and the CA's public key, while w must of course be difficult to compute without knowledge of the CA's secret information.

2.4 Certificate-based Cryptography

Certificate-based public-key cryptography (CB-PKC) was introduced by Gentry [18] in 2003, a bit before the similar concept of certificateless cryptography, which we discuss in the next section. The basic notion behind certificate-based encryption (CBE) is that, in order to decrypt, users need an up-to-date certificate as well as a secret key. The cryptosystem is designed to combine traditional PKI and ID-based schemes, in that it has no key escrow and allows implicit certification. By implicit certification in the encryption setting, we mean that a user should be able to (securely) encrypt a message for another merely by knowing the user's public key and the CA's public parameters. The recipient should only be able to decrypt said message if his certificate is fresh. Thus, the CA can simply stop sending certificates to revoked users, with no need for a complicated infrastructure to deal with, for example, CRLs.

As in PKI, users choose their own public-key pair and request a certificate from the CA validating the public key. The CA obliges as usual, but the provided certificate works as a partial private key as well as proof of current certification. Moreover, the CA uses an ID-based scheme to generate the certificate, so the certificate is bound to the associated user's ID. We emphasize that these certificates need not be confidential, even though they are needed for decryption. In this way, this model avoids the key-escrow problem, does not require secure channels for key distribution, and eliminates third-party queries to the CA on a particular user's certificate status.

2.5 Certificateless Public-Key Cryptography

We now discuss *certificateless public-key cryptography (CL-PKC)*, which was introduced by Al-Riyami and Paterson [1, 2] in 2003. As with the certificate-based variety, the idea of certificateless cryptography is to combine traditional PKI and ID-based systems in a way that preserves the advantages of each. That is, certificateless schemes also avoid the key escrow problem and the high management costs of certificate distribution, storage, verification, and revocation present in PKI. CL-PKC and CB-PKC, while differing in their treatment of KGC-provided information, appear to be closely related, although the exact nature of the connection is unclear [2, 51, 22]. We give an informal description of CL-PKC below; since certificateless schemes receive particular attention in later chapters of this thesis, we give a formal definition of certificateless encryption (CL-PKE) in Appendix A.

In CL-PKC, a user generates his private key by combining a secret value with a partial private key provided by the Key Generation Center. In this way, the KGC has no access to users' private keys. Similarly, to generate a public key, the user combines his secret value with public information from the KGC. Of course, this implies that public keys are no longer easily computable by third parties, as in

ID-PKC. Public keys must be made available in some other way, such as including them in flows as part of the protocol run or storing them in a public directory. However, this system has the added advantage of flexibility—the user can generate his public key before *or* after receiving his partial private key. For instance, a user Bob might conceivably use Alice's public key combined with some string x, where x is some requirement that needs to be checked before the KGC releases the corresponding private key. For example, x might say that "Alice must be at least 18 years of age."

Since public keys are not validated as in ID-based schemes or traditional PKI, we must assume that an adversary can replace public keys at will, and this attack must be incorporated into the security model. Now, if a KGC replaces public keys, it will be able to impersonate any user, since it can easily compute the corresponding private key. Thus, for all certificateless schemes, the KGC can launch a man-in-the-middle attack (Section 3.5). For this reason, security models for certificateless schemes generally assume that the KGC never replaces public keys. Al-Riyami and Paterson [2] argue that this amounts to roughly the same amount of trust that is invested in a CA in a traditional PKI. They make the point that, while often not stated explicitly, we usually trust that CAs do not produce certificates binding arbitrary public keys to a given identity. In any case, CL-PKC amounts to less trust than in an ID-based scheme, where the KGC has access to users' private keys by definition.

One way to avoid the issue of the KGC replacing public keys is to bind a user's public and private keys, a method from [1] we describe in Section 3.6.4. This technique requires the user to send his fixed public key to the KGC, which is then incorporated into the partial private key. The result is that there can be only one working public key per user, so the existence of more than one implies that the KGC created more than one partial private key binding the user to different public keys. In fact, with this binding technique, there should be no need for partial private keys to be kept secret. The corresponding unique public key was computed with the user's secret value—this value is necessary to compute the full private key and cannot be determined from the exposed partial private key. We note the similarity between CL-PKC with binding and CB-PKC above.

2.6 Use of Trusted Third Parties

We have seen trusted third parties at work in all of the above types of public-key cryptography, whether we have referred to this party as a certification authority (CA) or key generation center (KGC). For the purposes of this section we use the term KGC. We formalize the level of trust we place in this entity below, according to Girault's trust model [19].

Definition 2.6.1. We say that a cryptographic primitive achieves *trust level 1* if the KGC knows (or can trivially compute) users' private keys and can thereby impersonate users without fear of discovery.

Definition 2.6.2. We say that a cryptographic primitive achieves *trust level 2* if the KGC does not know (and cannot trivially compute) users' private keys, but can impersonate users by creating false public keys and/or certificates.

Definition 2.6.3. We say that a cryptographic primitive achieves *trust level 3* if the KGC does not know (and cannot trivially compute) users' private keys and is incapable of impersonating users by creating false public keys and/or certificates without detection.

Using this model, we see that ID-based schemes by definition can achieve at most trust level 1. Given the most general definition for certificateless cryptography, we see that these schemes generally achieve trust level 2, unless care is taken to bind users to their public keys. This follows, as without such a binding scheme, the KGC will be able to replace public keys without detection. Similarly, well-designed CB-PKC and self-certified PKC should achieve trust level 3.

Chapter 3

Key Agreement Protocols

A *key establishment protocol* is a protocol in which two or more parties gain access to a shared secret. Depending on the number of parties gaining access, we refer to the protocol as a *two-party*, *tripartite*, or (if there are more than three parties) *group* or *conference* key establishment protocol. We distinguish between *interactive* and *non-interactive* protocols, i.e., those protocols in which information is passed between participants and those which involve no communication.

If only one party is involved in choosing and/or obtaining the secret and securely transports this value to the other(s), we say the protocol is a *key transport* or *key distribution* protocol. If the shared secret is a function of information provided by and/or associated with each party, we say the protocol is a *key agreement (KAP)* protocol. We concern ourselves mainly with the latter, especially the two-party *dynamic* key agreement setting, wherein the established key varies with each execution.

We introduce some terminology common to these protocols. We refer to a protocol run as a *session*, and each message transmitted from one party to another as a *flow*. The shared secret resulting from a session is generally called (or used to determine) a *session key*. Protocols generally assume users (or pairs of users) have *long-lived keys (LL-keys)*, which are static secrets that are usually precomputed and stored securely. These are often used in conjunction with randomized secret input, which we refer to as *ephemeral* or *short-term* keys. Many key agreement protocols also assume the existence of a centralized server, often referred to as a *trusted authority (TA)*, *certification authority (CA)*, or *Key Generation Center (KGC)*, depending on the role played. We refer to this entity's secret information as the *master secret key*. We assume that the KGC communicates with users via a secure channel, whereas protocol flows are sent via an open channel. That is, eavesdroppers have no access to KGC/user communication, but can easily read anything sent between protocol participants.

There are two flavors of key agreement protocols, namely secret-key based and public-key based. These are, as one might expect, analogous to symmetric and public-key cryptosystems, respectively. That is, the former type of scheme requires each pair of users to have access to a shared LL-key, and in the latter, each user has his own LL-key pair.

3.1 Basic Example: Diffie-Hellman

The classic example of a two-party key agreement protocol is the Diffie-Hellman protocol [15], which we present below. We have two parties, Alice and Bob, whom we denote by A and B, respectively.

Let G be a cyclic group of prime order q with generator g. Alice chooses a number $a \in_R \mathbb{Z}_q^*$ and sends $T_A = g^a$ to Bob. Similarly, Bob chooses a number $b \in_R \mathbb{Z}_q^*$ and sends $T_B = g^b$ to Alice. Alice computes $K_A = T_B^a$ and Bob computes $K_B = T_A^b$. The session key is $K_A = K_B = g^{ab}$. (See Figure 3.1 below.)

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = g^a & & b \in_R \mathbb{Z}_q^*;\ T_B = g^b \\
 & \xrightarrow{T_A} & \\
 & \xleftarrow{T_B} & \\
K_A = T_B^a & & K_B = T_A^b.
\end{array}$$

Figure 3.1: Diffie-Hellman KAP

Now that we have seen an example, we should consider the basic goals of a key agreement protocol. For instance, in the Diffie-Hellman key agreement scheme, do Alice and Bob know who has access to the session key? Are Alice and Bob even sure they are communicating with each other?

Suppose there is a curious third party Eve who wishes to eavesdrop on all communications between Alice and Bob. What can she learn? Ostensibly she has access to both g^a and g^b and wishes to compute g^{ab}. This is exactly the computational Diffie-Hellman problem discussed in Chapter 1, which is believed to be infeasible, so in this case Eve cannot compute the session key. However, what if Eve not only listens passively, but also substitutes her own messages for those of Alice and Bob? If she does so, Eve can establish (different) keys with both Alice and Bob, with Alice and Bob none the wiser. We formalize these notions in the next session.

3.2 Objectives

There are several properties that key agreement protocols might, and depending on the application, should possess:

Definition 3.2.1. In a protocol with *(implicit) key authentication*, one party is certain that only an intended, specifically identified second party has the ability to access the session key.

Definition 3.2.2. In a protocol with *key confirmation*, one party may be certain that the second (perhaps unidentified) party actually possesses the session key.

Definition 3.2.3. A protocol in which both the implicit key authentication and key confirmation properties hold is said to provide *explicit key authentication*.

We also refer to a key agreement protocol with key authentication as an *authenticated key agreement (AK) protocol*. We emphasize that in such a protocol, neither party is sure that the other has actually computed the session key. A key agreement protocol that provides explicit key authentication is called an *authenticated key agreement protocol with key confirmation (AKC)*. As noted in [30], when combining entity authentication and key agreement, it is important to ensure that the identified party is also the party involved in the key agreement phase. Otherwise, the adversary might remain passive while the two parties successfully identify each other, i.e., achieve *mutual authentication*, but launch an active attack during key establishment.

3.3 Performance Considerations

In designing a protocol, it is necessary to try to achieve *low communication* and *computation* overhead. With this in mind, it is desirable for a protocol to use the minimal number of flows, which is (for obvious reasons) two for an AK protocol and three for an AKC protocol. Although we want computation complexity to be low for efficiency reasons, costly computations also leave protocols vulnerable to denial-of-service attacks, i.e., an attacker could flood the target with requests that require heavy computation, thereby leaving it incapable of participating in valid sessions. Many protocols attempt to satisfy this requirement by using as much *precomputation*, or offline computation, as possible, in order to reduce the amount required for online scenarios.

3.4 Security Attributes

We have the following desirable security attributes.

- *Key-compromise impersonation (KCI) security*: If the LL-key of user A is compromised, the attacker should not be able to impersonate another user B to A. Obviously, if an LL-key of A is compromised, we wish to replace this key as soon as possible, as the attacker can certainly impersonate A to any other user. This property of KCI security, also called the KCI resilience property, is nevertheless important in the sense that it minimizes the damage until the user can detect that his key has been compromised. A standard example of a KCI attack is if A is a customer of a bank B. If a third party armed with A's LL-key can successfully impersonate B to A, he can potentially get A to release information one would normally only give to one's bank, for example a PIN number or other sensitive account information.

Any protocol that uses static shared secrets computable without any user interaction, such as that provided by Sakai, Ohgishi, and Kasahara [35], is open to this type of attack. This is because an attacker armed with A's LL-key has the same ability as A to compute the static shared secret. An example of a protocol by Ryu et al. exhibiting this vulnerability can be found in [32] and a corresponding attack in [47]. Wang, Cao, and Wang's protocol [49] suffers from the same weakness, as we show in Section 6.5. To avoid this attack, the use of ephemeral keys and validity checks must be carefully considered.

- *Known session-specific temporary information security*: If the attacker has access to the ephemeral keys of a given protocol run, he should be unable to determine the corresponding session key. For *weak known session-specific temporary information security*, it is further assumed that the TA/KGC cannot access the ephemeral keys. As argued by Mandt [27], practical implementations of protocols often involve precomputing and/or storing ephemeral keys insecurely, so the secrecy of the established session key should not rely solely on the choice of these short-term keys.

- *Known session key security*: Key agreement protocols should be dynamic: each protocol run should result in a unique session key. An attacker who learns a given number of session keys should not be able to discover other session keys.

- *Forward secrecy*: Given the LL-keys of one or more users, it is clearly desirable that an attacker not be able to determine previously established session keys. *Perfect forward secrecy* implies an attacker, even armed with all participants' LL-keys, cannot determine old session keys. *Partial forward secrecy* implies an attacker armed with some, but not all, participants' LL-keys cannot determine old session keys. Similarly, *TA* or *KGC forward secrecy* deals with the case in which the attacker has the master secret key. We also define *weak perfect forward secrecy*, in which it is assumed that all LL-keys are known, but the attacker was not actively involved in choosing ephemeral keys during the sessions of interest. It has been shown by Krawczyk [23] that no 2-flow AK protocol can do better than this weaker version of forward secrecy.

- *Unknown key-share*: It should be impossible to coerce A into thinking he is sharing a key with B, when he is actually sharing a key with another (honest) user C. That is, it should not be possible for A to believe he is sharing a key with $B \neq C$, while C correctly thinks the key is shared with A. The standard real-world example of an unknown key-share attack (first mentioned by Diffie, Oorschot, and Wiener [16]) is the following. Suppose A is a bank and C and B are customers, and customer certificates (issues by bank headquarters) contain the holder's account information. Suppose electronic deposits work by exchanging a key with the bank via an authenticated key agreement protocol. That is, as soon as the bank authenticates B, encrypted funds are deposited according to the account number provided in the associated certificate. If there is no separate authentication of the encrypted deposit message, the above unknown key share attack will result in the bank believing he is communicating with B instead of C, so the funds (belonging to C) will be deposited to B's account. A standard method to prevent this attack is to include the identities of the parties in the key derivation function.

- *Key control*: Participants should have the same amount of control in determining the session key. In practice it is difficult to achieve *perfect* key control, since it is necessary for one party to initiate the protocol run and choose his ephemeral key first, so the responding party has the ability to estimate some of the bits of the session key through different choices of ephemeral keys. However, we desire this trial-and-error method to be the best available technique for one party to have influence over the key choice.

- *Deniability*: In some applications, it is desirable for protocol participants to be able to deny taking part in a given protocol run.

- *Message independence*: Flows of a protocol run should be unrelated. Of course, this property makes the most sense in the context of an AK protocol, as key confirmation inherently involves some sort of message dependence.

3.5 Summary of Attacks

Generic protocol attacks fall into two categories: *passive* attacks, in which the adversary merely eavesdrops, or observes flows between honest entities, and *active* attacks, in which the adversary deletes, inserts, replays, or otherwise alters flows. We have already mentioned *key compromise impersonation attacks*, *unknown key share attacks*, and some types of *known key attacks* in the discussion of desirable security attributes in Section 3.4. We provide a list of general protocol attacks below, with the caveat that given any particular protocol (and implementation of said protocol), the following list is likely not complete (and in some cases not applicable). We also wish to make the point that listing these attacks separately is somewhat misleading. In reality adversaries can (and will) combine various methods of attack. In this way, it is possible for a protocol to withstand any given subset of attacks separately, yet not be able to protect against some combination of them. This leads to the notion of provable security, which we discuss in Chapter 4.

- *Source substitution attack*: In this type of attack, the adversary claims the public key of the user A as his own. This is possible if protocol participants do not have any way to check that the other user actually has the private key corresponding to the claimed public key. A typical example of this attack can be found in [30] relating to the MTI/AO key agreement protocol, in which the adversary is unable to compute the session key, but fools B into believing he has shared a key with the adversary rather than with A.

- *Time-memory trade-off attack*: If the protocol involves a flow containing the hash of the session key K, the adversary can try to determine the key by precomputing a given number of hash values, say 2^r for some r, and compare these to the hash provided in the protocol run. If K is k bits long, then the probability of success is 2^{r-k}.

- *Known key attacks*: This class of attacks involves the adversary having access to one or more pieces of secret information and using this to obtain the session key of another protocol run.

  - *key reveal attack*: In this type of attack, the adversary takes advantage of any algebraic relationship between session keys. Formally, the adversary is armed with a key reveal oracle that outputs old session keys, and uses this to gain information about the unknown session key.
  - *key-replication attack*: In this scenario, the adversary finds another session that generates a key identical to that of the desired session and uses this to try to determine the session key.

- *Known session-specific temporary information attack*: The adversary is armed with the ephemeral keys of a particular session and tries to determine the session key.

- *Known LL-key attack*: The adversary knows the LL-key of a protocol participant and uses this to gain information about previous or future session keys. (Note that this attack addresses key compromise impersonation as well as the issue of forward secrecy.)

- *Forgery attack*: The adversary attempts to forge a session key by observing flows and using the public parameters of the protocol. This attack is not necessarily thwarted by the use of a collision-free hash function for key derivation.

- *Degenerate message attack*: Care should be taken to verify that ephemeral and public keys do not result in a key that is trivial to compute. For instance, it is generally wise to check that ephemeral and public keys are not the identity element in their respective groups, as this often makes the key itself trivial. We will see examples of protocols which do not have sufficient validity checks on ephemeral/public keys in Chapter 6.

- *Interleaving attack*: The adversary uses flows from previous and/or ongoing sessions in another protocol run in order to impersonate/deceive another user. Examples of this type of attack include:
  - *replay attack*: The adversary modifies a protocol run by inserting all or part of a message from a previous and/or ongoing protocol run. This type of attack is often used in conjunction with other attack elements.
  - *reflection attack*: In the basic reflection attack, the adversary observes an ongoing protocol run π between A and B (where A is the initiator), and initiates another session π' with A in which he replays A's messages from π. He uses A's responses in π' to impersonate B in π.
- *Man-in-the-middle attack*: The adversary intercepts session messages between A and B, and replaces them with messages of his own choice in order to establish separate keys K_A and K_B with A and B, respectively, while they believe they have established a joint key with each other. The adversary can then use these keys to spy on all communication between A and B by decrypting and re-encrypting appropriately. For example, if A sends a message to B, the adversary will use K_A to decrypt, re-encrypt under K_B, and send the message on to B, with A and B none the wiser. This attack must by definition be impossible in an AK protocol. KGCs in certificateless protocols, as noted in Section 2.5, can always mount man-in-the-middle attacks by replacing public keys, so in this case we must trust the KGC to not replace any public keys.

3.6 Examples

We provide representative examples of key agreement protocols from traditional PKI, identity-based, certificateless, and self-certified cryptography. We discuss the main features of each, paying particular attention to the level of trust invested in the KGC.

3.6.1 Traditional PKI: Unified Model KAP

We present the Unified Model (UM) with key confirmation, which was introduced by Blake-Wilson, Johnson, and Menezes [6], as an example of an AKC scheme in the traditional PKI setting. There are several variants of the unified model; we have chosen to provide the 3-pass version that is included in the NIST SP 800-56A standard, the security of which Menezes and Ustaoğlu [29] consider in detail. We feel this protocol is illustrative of key agreement supported by traditional PKI. Moreover, the UM protocol is an authenticated key agreement protocol that is believed to possess forward secrecy as well as resistance to unknown key-share, known-session key, and known session-specific temporary information attacks. It has been shown, however, to be vulnerable to KCI attacks, which is evident from the protocol specification.

We describe the protocol below. (Refer to Figure 3.2.) To make the protocol more readable, we omit certain details, such as the inclusion of session identifiers, which are needed to achieve full security and thwart more complicated interleaving attacks, but are not essential to understanding the protocol structure. As usual in the traditional PKI setting, we assume that our two parties, Alice (A) and Bob (B), have access to authentic copies of each other's public keys through the exchange of certificates issued by a certification authority.

Assume $G = \langle g \rangle$, $|G| = q$, where q is prime, and the discrete logarithm problem is hard in G. We identify each user i by the string ID_i, with associated private key $x_i \in_R \mathbb{Z}_q^*$ and public key $P_i = g^{x_i}$. We make use of key derivation functions H and H', as well as a message authentication code algorithm, MAC; $\mathcal{R}$ denotes the fixed string "KC_2_U" and $\mathcal{I}$ the fixed string "KC_2_V".

1. Alice picks $a \in_R \mathbb{Z}_q^*$ and computes $T_A = g^a$. She then sends $\langle \mathrm{ID}_B, \mathrm{ID}_A, T_A \rangle$ to Bob.

2. Bob checks that $T_A \in G^*$. If this is the case, he picks $b \in_R \mathbb{Z}_q^*$ and computes $T_B = g^b$. He sets $\sigma_e = T_A^b$, $\sigma_s = P_A^{x_B}$, and $k' = H'(\sigma_e, \sigma_s, \mathrm{ID}_A, \mathrm{ID}_B)$. He then destroys σ_e, σ_s, and b, and sets $t_B = \mathrm{MAC}_{k'}(\mathcal{R}, \mathrm{ID}_B, \mathrm{ID}_A, T_B, T_A)$. He then sends $\langle \mathrm{ID}_A, \mathrm{ID}_B, T_A, T_B, t_B \rangle$ to Alice.

3. Alice checks that $T_B \in G^*$. If this is the case, she sets $\sigma_e = T_B^a$ and $\sigma_s = P_B^{x_A}$. She computes $k' = H'(\sigma_e, \sigma_s, \mathrm{ID}_A, \mathrm{ID}_B)$ and verifies that $t_B = \mathrm{MAC}_{k'}(\mathcal{R}, \mathrm{ID}_B, \mathrm{ID}_A, T_B, T_A)$. She computes $t_A = \mathrm{MAC}_{k'}(\mathcal{I}, \mathrm{ID}_A, \mathrm{ID}_B, T_A, T_B)$ and destroys k'. She then sends $\langle \mathrm{ID}_B, \mathrm{ID}_A, T_A, T_B, t_B, t_A \rangle$ to Bob.

4. Bob verifies that $t_A = \mathrm{MAC}_{k'}(\mathcal{I}, \mathrm{ID}_A, \mathrm{ID}_B, T_A, T_B)$ and destroys k'.

5. Alice and Bob both accept $k = H(\sigma_e, \sigma_s, \mathrm{ID}_A, \mathrm{ID}_B)$ as the session key.

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = g^a & & b \in_R \mathbb{Z}_q^*;\ T_B = g^b \\
 & \xrightarrow{\langle \mathrm{ID}_B, \mathrm{ID}_A, T_A \rangle} & \\
 & \xleftarrow{\langle \mathrm{ID}_A, \mathrm{ID}_B, T_A, T_B, t_B \rangle} & \\
 & \xrightarrow{\langle \mathrm{ID}_B, \mathrm{ID}_A, T_A, T_B, t_B, t_A \rangle} & \\
H(T_B^a, P_B^{x_A}, \mathrm{ID}_A, \mathrm{ID}_B) & & H(T_A^b, P_A^{x_B}, \mathrm{ID}_A, \mathrm{ID}_B)
\end{array}$$

Figure 3.2: UM AKC protocol

An attacker Eve armed with Alice's private key x_A can clearly launch a successful KCI attack. She merely intercepts Alice's messages to Bob, picks her own ephemeral key $b' \in_R \mathbb{Z}_q^*$, and follows the protocol in Bob's stead. We also comment on the use of message authentication code algorithms in this protocol—their primary use is to provide key confirmation. A user's ability to use the correct key in the MAC assures the other party that k' has been computed, thereby providing evidence of the user's ability to compute the session key k.

3.6.2 ID-based PKC: Smart's KAP

We give one of the first ID-based key agreement schemes, introduced by Smart [40].

Let q be a prime, G_1 be a cyclic additive group of order q generated by P, and G_T a multiplicative group of order q. Let $e\colon G_1 \times G_1 \to G_T$ be an admissible pairing and h a hash function satisfying $h\colon \{0,1\}^* \to G_1^*$. We assume the discrete logarithm problem (DLP) is hard in both G_1 and G_T. The KGC chooses a master secret key $s \in_R \mathbb{Z}_q^*$, sets $P_{\text{KGC}} = sP$ as its public key, and publishes the system parameters $\langle G_1, G_T, e, q, P, P_{\text{KGC}}, h\rangle$. Each user i has public key $Q_i = h(\text{ID}_i)$ and private key $S_i = sQ_i$, which is distributed securely by the KGC.

Alice (A) and Bob (B) agree on a session key as follows. (See Figure 3.3.) Alice picks $a \in_R \mathbb{Z}_q^*$ and computes $T_A = aP$. She sends T_A to Bob. Similarly, Bob picks $b \in_R \mathbb{Z}_q^*$ and computes $T_B = bP$. He sends T_B to Alice. Alice computes $K_A = e(Q_B, P_{\text{KGC}})^a e(S_A, T_B)$ and Bob computes $K_B = e(S_B, T_A)e(Q_A, P_{\text{KGC}})^b$. They use $K_A = e(Q_B, P_{\text{KGC}})^a e(S_A, T_B) = e(sQ_B, aP)e(sQ_A, bP) = e(S_B, T_A)e(Q_A, P_{\text{KGC}})^b = K_B$, which we denote by K, as the session key.

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = aP & & b \in_R \mathbb{Z}_q^*;\ T_B = bP \\
 & \xrightarrow{T_A} & \\
 & \xleftarrow{T_B} & \\
K_A = e(Q_B, P_{\text{KGC}})^a e(S_A, T_B) & & K_B = e(S_B, T_A)e(Q_A, P_{\text{KGC}})^b
\end{array}$$

Figure 3.3: Smart KAP

In its original form, the CA can easily compute all session keys, as the session key is $K = e(S_B, T_A)e(S_A, T_B)$. By the same argument, the protocol fails to have (weak) perfect forward secrecy. Chen and Kudla [14] modify the protocol by making use of a hash function $H\colon G_T \times G_1 \to \{0,1\}^k$ for some $k \in \mathbb{N}$, and using $H(K\|abP)$ as the session key. This version achieves weak perfect forward secrecy—so long as the attacker was not actively involved in the session, knowledge of the participants' LL-keys is not sufficient to compute the session key. Note that this implies that the KGC, so long as it remained passive during the session, is also unable to compute the key, i.e., this scheme is escrowless. However, we do want to emphasize that even in escrowless variants of ID-based schemes, the KGC can mount a man-in-the-middle attack undetected. It is easy to see that both versions are vulnerable to a known session-specific temporary information attack.

3.6.3 Certificate-based PKC: Wang-Cao KAP

Although it seems that not much attention has been paid to certificate-based primitives, especially key agreement protocols, we provide a scheme by Wang and Cao [46] that is based on Gentry's certificate-based encryption scheme (Section 2.4) and Smart's AK protocol (Section 3.6.2). We do not believe this protocol has received much attention from the cryptographic community, but as it is closely related to Mandt's scheme, which we cryptanalyze in Chapter 6, we provide it below.

Let q be a prime, G_1 be a cyclic additive group of order q generated by P, and G_T a multiplicative group of order q. Let $e\colon G_1 \times G_1 \to G_T$ be an admissible pairing. We assume the discrete logarithm problem (DLP) is hard in both G_1 and G_T. We also make use of hash functions $H\colon \{0,1\}^* \to G_1^*$, and a key derivation function, denoted kdf.

The KGC chooses a master secret key $s \in_R \mathbb{Z}_q^*$, sets $P_{\text{KGC}} = sP$ as its public key, and publishes the system parameters $\langle G_1, G_T, e, q, P, P_{\text{KGC}}, H \rangle$.

The user i chooses $x_i \in_R \mathbb{Z}_q^*$ as his secret value and sets $P_i = x_iP$ as his public key. He requests a certificate from the CA by sending a data string data_i, which includes his public key P_i and his identifying information ID_i. The CA computes $Q_i = H(P_{\text{KGC}}, \text{data}_i) \in G_1$ and sends the certificate $\text{Cert}_i = sQ_i$ to the user. The user will then use this certificate as a partial private key, computing $S_i = \text{Cert}_i + x_iQ_i = (s + x_i)Q_i$ for his full private key.

Alice (A) and Bob (B) agree on a session key as follows. (Refer to Figure 3.4.)

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = aP & & b \in_R \mathbb{Z}_q^*;\ T_B = bP \\
 & \xrightarrow{T_A} & \\
 & \xleftarrow{T_B} & \\
K_A = e(P_{\text{KGC}} + P_B, Q_B)^a & & K_B = e(T_A, S_B) \\
K'_A = e(T_B, S_A) & & K'_B = e(P_{\text{KGC}} + P_A, Q_A)^b \\
\text{kdf}(\text{ID}_A \,||\, \text{ID}_B \,||K_A||K'_A||aT_B) & & \text{kdf}(\text{ID}_A \,||\, \text{ID}_B \,||K_B||K'_B||bT_A).
\end{array}$$

Figure 3.4: Wang-Cao KAP

1. Alice picks $a \in_R \mathbb{Z}_q^*$ and sends $T_A = aP$ to Bob.

2. Bob picks $b \in_R \mathbb{Z}_q^*$ and sends $T_B = bP$ and P_B to Alice.

3. Alice computes $Q_B = H(P_{\text{KGC}}, \text{data}_B) \in G_1$, $K_A = e(P_{\text{KGC}} + P_B, Q_B)^a$, and $K'_A = e(T_B, S_A)$. She uses $\text{kdf}(\text{ID}_A \,||\, \text{ID}_B \,||K_A||K'_A||aT_B)$ as the session key.

4. Bob computes $Q_A = H(P_{\text{KGC}}, \text{data}_A) \in G_1$, $K_B = e(T_A, S_B)$, and $K'_B = e(P_{\text{KGC}} + P_A, Q_A)^b$. He uses $\text{kdf}(\text{ID}_A \,||\, \text{ID}_B \,||K_B||K'_B||bT_A)$ as the session key.

We note that this protocol is consistent, since $K_A = e(P_{\text{KGC}} + P_B, Q_B)^a = e(sP + x_BP, Q_B)^a = e(aP, (s + x_B)Q_B) = e(T_A, S_B) = K_B$ and similarly $K'_A = e(T_B, S_A) = e(P_{\text{KGC}} + P_A, Q_A)^b = K'_B$.

The authors do not specify how protocol participants obtain the public keys of other parties, so we suggest that these be sent as part of the flows. There are no checks specified on Alice and Bob's public keys; Alice and Bob do not even ensure that the public keys are elements of G_1^*, which potentially leaves the protocol open to the small subgroup attack mentioned by Lim and Lee [25]. Also, while the authors do mention that the KGC's certificates need not be kept secret, they do not use the certificates in a useful way. If the certificates are not public information, or if users do not check that the certificates are valid, then we can assume an attacker has the freedom to replace public keys undetected. However, with the appropriate checks in place, it appears this scheme achieves trust level 3 and has most desirable security attributes. We note, though, that the scheme evidently does not have known session-specific temporary information security, which can be fixed by adding x_Ax_BP to the key derivation function and modifying kdf as appropriate.

3.6.4 Certificateless PKC: Al-Riyami Paterson KAP

The first certificateless key agreement scheme was introduced by Al-Riyami and Paterson [1]. We present this protocol below.

Let q be a prime, G_1 be a cyclic additive group of order q generated by P, and G_T a multiplicative group of order q. Let $e\colon G_1 \times G_1 \to G_T$ be an admissible pairing. We assume the discrete logarithm problem (DLP) is hard in both G_1 and G_T. We also make use of hash functions $h\colon \{0,1\}^* \to G_1^*$ and $h'\colon G_T \times G_1 \to \{0,1\}^k$ for some $k \in \mathbb{N}$.

The KGC picks $s \in_R \mathbb{Z}_q^*$ as its master secret key, sets $P_{\text{KGC}} = sP$ as its master public key, and makes the parameters $\langle G_1, G_T, e, q, P, P_{\text{KGC}}, h, h'\rangle$ public. The KGC provides each user i with a partial private key $D_i = sQ_i \in G_1$, where $Q_i = h(\text{ID}_i)$. The user verifies that $e(D_i, P) = e(Q_i, P_{\text{KGC}})$ before using D_i.

The user i also picks $x_i \in_R \mathbb{Z}_q^*$ as his secret value (which acts as an additional partial private key) and computes his full private key as $S_i = x_iD_i = x_isQ_i$. He sets his public key as $P_i = \langle X_i, Y_i\rangle$, where $X_i = x_iP$ and $Y_i = x_iP_{\text{KGC}} = x_isP$.

Users Alice (A) and Bob (B) agree on a session key as follows. (Refer to Figure 3.5.) Alice picks $a \in_R \mathbb{Z}_q^*$ and computes $T_A = aP$. She sends T_A and $P_A = \langle X_A, Y_A\rangle$ to Bob. Similarly, Bob picks $b \in_R \mathbb{Z}_q^*$ and computes $T_B = bP$. He sends T_B and $P_B = \langle X_B, Y_B\rangle$ to Alice. Alice then checks that $e(X_B, P_{\text{KGC}}) = e(Y_B, P)$ and computes $K_A = e(Q_B, Y_B)^a e(S_A, T_B)$. Bob checks that $e(X_A, P_{\text{KGC}}) = e(Y_A, P)$ and computes $K_B = e(S_B, T_A)e(Q_A, Y_A)^b$. Alice uses $h'(K_A||aT_B)$ as the session key, and Bob uses $h'(K_B||bT_A)$.

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = aP & & b \in_R \mathbb{Z}_q^*;\ T_B = bP \\
 & \xrightarrow{T_A, P_A} & \\
 & \xleftarrow{T_B, P_B} & \\
e(X_B, P_{\text{KGC}}) \overset{?}{=} e(Y_B, P) & & e(X_A, P_{\text{KGC}}) \overset{?}{=} e(Y_A, P) \\
K_A = e(Q_B, Y_B)^a e(S_A, T_B) & & K_B = e(S_B, T_A) e(Q_A, Y_A)^b \\
h'(K_A || aT_B) & & h'(K_B || bT_A).
\end{array}$$

Figure 3.5: Al-Riyami and Paterson KAP

Since

$$\begin{aligned}
K_A &= e(Q_B, Y_B)^a e(S_A, T_B) \\
&= e(Q_B, x_B sP)^a e(x_A sQ_A, bP) \\
&= e(x_B sQ_B, aP) e(Q_A, x_A sP)^b \\
&= e(S_B, T_A) e(Q_A, Y_A)^b \\
&= K_B,
\end{aligned}$$

we see that, as desired, Alice and Bob agree on the same session key, namely $h'(e(Q_B, Y_B)^a e(Q_A, Y_A)^b || abP)$.

This scheme clearly does not have the known session-specific temporary information security property, as pointed out by Mandt [27]. However, we comment that this is easy to fix by making the session key $h'(e(Q_B, Y_B)^a e(Q_A, Y_A)^b || abP || x_A x_B P)$ and modifying h' as necessary (so Alice computes $x_A X_B = x_A x_B P$ and Bob computes $x_B X_A = x_A x_B P$).

This scheme has the nice property associated with certificateless public-key systems in that there is no key escrow. While the KGC cannot compute users' secret keys, it can nevertheless impersonate users (in a man-in-the-middle attack scenario) by creating false public keys, and so we say that this scheme reaches trust level 2. However, we emphasize that the KGC cannot perform a man-in-the-middle attack without both replacing ephemeral values and public keys, whereas in ID-based systems modified to eliminate key escrow, the KGC need only replace ephemeral values. The binding technique mentioned in [1] ensures users have only one public key for which they have the associated private key. To accomplish this, a user i must first fix his public key $P_i = \langle X_i, Y_i \rangle$ and send this to the KGC. We redefine $Q_i = h(ID_i || P_i)$ and continue the protocol as normal. In this way, the user's partial private key is bound to his choice of public key. The existence of two *working* public keys exposes a cheating KGC, as it implies the existence of two partial private keys for the same user, which are only computable by the KGC. Thus we say that with this added binding technique, the scheme achieves trust level 3.

As noted by Al-Riyami and Paterson, it is still possible for the KGC to *temporarily* replace public keys. That is, the KGC could mount a man-in-the-middle attack by replacing both ephemeral values and public keys, and then reset the public keys. In this way, the KGC might obtain access to some session keys or to the plaintext versions of some ciphertexts without being detected. There seems to be no way to prevent this type of attack, but we remind the reader that a CA in traditional PKI can always launch such an attack without fear of discovery.

3.6.5 Self-certified PKC: Girault KAP

We give the first self-certified key agreement scheme of Girault [19], the security of which depends on the hardness of factoring large integers and the DLP assumption. The CA chooses an RSA modulus $n = pq$ (so p and q are large distinct primes) with corresponding encryption key e and decryption key d satisfying $ed \equiv 1 \pmod{\phi(n)}$. We let g be an integer of maximal order in $\mathbb{Z}_n^*$. To each user i there is an associated 3-tuple $(\mathrm{ID}_i, x_i, P_i)$, where ID_i, which we assume to have built-in redundancy and/or structure, identifies i, $x_i \in_R \mathbb{Z}_n$ is the user's private key, and $P_i = (g^{-x_i} - \mathrm{ID}_i)^d \pmod{n}$ is the user's public key (which doubles as a certificate). That is, the public key of a user is essentially the CA's RSA signature on $g^{-x_i} - \mathrm{ID}_i$.

In order to receive a public key, the user i sends $\nu_i = g^{-x_i} \pmod{n}$ to the CA and proves that he knows x_i. For details on this *identification protocol* see [19]. This identification protocol also serves as the method by which users certify that they have the correct public key for the other party. Public keys are self-certified in the sense that only a CA could produce a key P_i satisfying $P_i^e + \mathrm{ID}_i = \nu_i = g^{-x_i} \pmod{n}$. We observe that the redundancy/structure of ID_i plays an important role in the self-certification process, as this prevents an attacker from selecting arbitrary $x_i, P_i \in \mathbb{Z}_n$ and claiming $\mathrm{ID}_i = g^{-x_i} - P_i^e$ as his identity. If Bob wishes to authenticate Alice's public key, he computes ν_A and initiates the identification protocol, in which Alice proves she knows the discrete logarithm of ν_A with respect to g in $\mathbb{Z}_n^*$. Note that, provided the DLP assumption holds in $\mathbb{Z}_n^*$, neither the CA nor another user is capable of computing x_i from ν_i.

Two users Alice (A) and Bob (B) can easily establish a shared secret $g^{-x_A x_B} \pmod{n}$. To see this, note that Alice can compute $(P_B^e + \mathrm{ID}_B)^{x_A} = ((g^{-x_B} - \mathrm{ID}_B)^{ed} + \mathrm{ID}_B)^{x_A} = (g^{-x_B} - \mathrm{ID}_B + \mathrm{ID}_B)^{x_A} = g^{-x_A x_B}$ and similarly Bob can compute $(P_A^e + \mathrm{ID}_A)^{x_B}$. Of course, since this is a static key agreement scheme (involving only the users' long-term secret keys), it is automatically vulnerable to a KCI attack, and is completely insecure if any private information is leaked.

Girault claims that this scheme reaches trust level 3, in that the CA not only does not have access to users' secret keys, but is incapable of publishing false public keys without being detected. While the CA can certainly use knowledge of d to create a false public key for any given user, given the security of the RSA signature scheme, the CA is also the *only* party able to do so. That is, the existence of two or more public keys for the same user automatically implies that the CA has misbehaved. Saeednia [34] comments that this scheme actually only achieves trust level 1, in the sense that there is no guarantee (or verification) that the CA will pick an appropriate RSA modulus n. The CA could potentially pick n in such a way that computing discrete logarithms is feasible. For instance, if p and q are are relatively small primes, or if $p-1$ and and $q-1$ both have small prime factors, the CA can use the Pohlig-Hellman algorithm to solve the DLP (see [30] for details).

Chapter 4

Provable Security

We now turn to the notion of provable security, which originated with Goldwasser and Micali [20]. Before this, there was no formal analysis of the security of cryptographic primitives. New primitives were given the test of time: people tried to break them and the longer they lasted, the more secure they were thought to be. Of course, numerous primitives have been broken, sometimes years after they were initially presented, and so the idea of trying to *prove* security was born. The basic outline of "proving" a cryptographic primitive secure involves carefully defining the goals of the primitive (including the capabilities of the adversary) and then showing the primitive meets these goals. Usually these proofs are subject to some standard number-theoretic assumption, for example the hardness of the DH or RSA problem.

We have already mentioned the limitations of testing the security of a given protocol by using a checklist of attacks—while useful in determining whether the protocol is immediately vulnerable to a particular attack, it does not show the protocol to be resilient to some *combination* of attacks. This problem prompted designers of key establishment protocols to turn to provable security as an analytical tool. In the remainder of the chapter we present the major security models in the area of key agreement, starting with the original Bellare-Rogaway model and Blake-Wilson, Johnson, Menezes model in Section 4.1. We then present the Canetti-Krawczyk model in Section 4.2 and finally, in Section 4.3, the extended Canetti-Krawczyk model, which appears to be the favored model in the literature today. We observe that all of these models, if designed to address PKC rather than symmetric systems, are considered from the viewpoint of traditional PKI. They can be readily adapted to the ID-based setting [13, 9], but it seems that settings in which the secret keys of users involve two pieces of information, such as certificateless, require giving the adversary new powers. We consider how the eCK model, in particular, might be adapted for certificateless key agreement protocols.

4.1 Bellare-Rogaway (BR) Model

We present the Bellare-Rogaway (BR) model [4, 5] and the Blake-Wilson, Johnson, Menezes (BJM) model [6], as these provide the framework for the more current Canetti-Krawczyk models [11]. The only fundamental difference between the BR model and the BJM model is that the first addresses the setting of symmetric key cryptography, and the latter that of public-key cryptography. The underlying idea is that a protocol is considered secure if the adversary cannot distinguish between a valid session key and a randomly generated value; this idea of provable security based on *indistinguishability* originated with Goldwasser and Micali [20]. These models assume that all parties behave honestly in the key registration phase; that is, parties form their public keys correctly and register them with the CA.

In the Bellare-Rogaway model, all communication is controlled by the adversary. That is, the adversary is allowed to intercept, modify, delete, and create messages of his choosing, as well as deliver messages out of order or to unintended recipients. Moreover, the adversary can control the number of sessions occurring at any given time, as well as the identities of the parties involved. Informally, the adversary plays a "game." He simulates protocol runs, subject to a certain set of constraints, and eventually tries to win the game by guessing whether a given output is a valid session key or a randomly generated value.

Formally, we call the set of possible entities I and model the adversary E by a probabilistic polynomial time Turing machine. (We note that E is not a member of I.) The protocol is viewed as a function Π that specifies the behavior of parties in an honest protocol run; we assume Π is efficiently computable in a given security parameter k. (In the BJM model, we also have an efficiently computable function $\mathcal{G}$, which generates key pairs.) The adversary has access to a collection of oracles $\{\Pi^s_{i,j} | i, j \in I, s \in \mathbb{N}\}$, where $\Pi^s_{i,j}$ denotes the s^{th} protocol run initiated by entity i in an attempt to establish a session key with entity j. Oracles can either "accept" or "reject" protocol runs, according to the rules of the protocol. Rejection can occur at any time, whereas acceptance is generally at the end of the run and indicates the run was successful. We assume the session key is an element of $\{0, 1\}^k$.

We say a *conversation* for a given oracle is the ordered concatenation of all messages (both incoming and outgoing). Moreover, two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have *matching conversations* if the outgoing messages of one are the incoming messages of the other, and vice versa. That is, the initiator oracle $\Pi^s_{i,j}$ must have a conversation matching to the responder oracle $\Pi^t_{j,i}$: each message the responder received was generated first by the initiator oracle, the response to this message was the next message of the initiator's conversation, and the initiator's subsequent response was the next message of the responder oracle. (We remark that a responder oracle's conversation matching that of an initiator oracle is a slightly different concept, in that the last message of the initiator might not be delivered correctly.) Informally, two oracles having a matching conversation when the adversary delivers all messages faithfully, in which case we call the adversary *benign*. We wish the oracles involved in a given protocol run to accept only if they have matching conversations. This leads to the following definition.

Definition 4.1.1. We say Π is a *secure mutual authentication* protocol if for any polynomial time adversary E:

1. If oracles $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$ have matching conversations, then both accept;
2. The probability of $\Pi_{i,j}^s$ accepting is negligible when there is no oracle with a matching conversation.

The power of the adversary E is modeled by three queries, namely Send, Reveal, and $\mathsf{Corrupt}$. The $\mathsf{Send}(\Pi_{i,j}^s, x)$ query translates to the adversary E sending the message x to the oracle $\Pi_{i,j}^s$. For example, E can tell entity i to initiate the s^{th} protocol run with entity j by setting $x = \lambda$, the empty string. The $\mathsf{Reveal}(\Pi_{i,j}^s)$ query tells the given oracle to reveal the session key it currently holds (if any). The $\mathsf{Corrupt}(i)$ query allows E to take over entity i. (In the BJM model, the $\mathsf{Corrupt}$ query also allows the adversary to replace i's key pair with a valid key pair of E's choice. This essentially allows the adversary to have access to i's LL-key.) To distinguish among oracles the adversary may have influenced in some way, we say that an oracle $\Pi_{i,j}^s$ is *fresh* if the oracle computes and accepts a session key, the adversary does not use the $\mathsf{Reveal}(\Pi_{i,j}^s)$ query or the Reveal query on the oracle matching to $\Pi_{i,j}^s$, and the adversary does not use the $\mathsf{Corrupt}$ query on either i or j.

Once E has finished interacting with the oracles, it picks a fresh oracle to be the *test* oracle. A bit b is then picked at random. If $b = 0$, the test oracle reveals the session key, and if $b = 1$, it generates a random value in $\{0,1\}^k$. The adversary then attempts to guess b. We define $\text{GoodGuess}^E(k)$ to be the event that E correctly guesses b, and

$$\text{Advantage}^E(k) = \max\left\{0, \left|\Pr[\text{GoodGuess}^E(k)] - \frac{1}{2}\right|\right\}.$$

This leads us the following formal security definitions:

Definition 4.1.2. We say that a key exchange protocol Π in security parameter k is a *BR-secure AK* protocol if the following hold for any polynomial time adversary E:

1. If E is a benign adversary on $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$, both oracles accept and hold the same session key, where this key is distributed uniformly at random on the key space;
2. If oracles $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$ have matching conversations and both i and j are uncorrupted, both oracles accept and hold the same session key;
3. $\text{Advantage}^E(k)$ is negligible.

Definition 4.1.3. We say that a key exchange protocol Π in security parameter k is a *BR-secure AKC* protocol if the following hold for any polynomial time adversary E:

1. If E is a benign adversary on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, both oracles accept and hold the same session key, where this key is distributed uniformly at random on the key space;
2. If oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have matching conversations and both i and j are uncorrupted, both oracles accept and hold the same session key;
3. $\text{Advantage}^E(k)$ is negligible;
4. The probability of $\Pi^s_{i,j}$ accepting is negligible when there is no oracle with a matching conversation.

Note that the latter definition is equivalent to saying that Π is a BR-secure AKC protocol if Π is a secure mutual authentication protocol and a BR-secure AK protocol.

4.2 Canetti-Krawczyk (CK01) Model

The Canetti-Krawczyk (CK01) Model [11] was designed to analyze the use of key establishment protocols in combination with symmetric encryption systems and authentication functions, something which the BR model does not consider. In particular, protocols secure under the CK01 model can be used to create "secure channels." Roughly speaking, session keys produced by CK01-secure key agreement protocols can be used in a symmetric key encryption scheme to provide "secret" and authenticated data transmission. The CK01 model uses the notion of indistinguishability from the BR model and is designed in the public-key setting.

Formally, in the CK01 model, we have a (finite) set of polynomial time machines P_i called *parties*. By a *message-driven protocol*, we refer to a collection of interactive processes run concurrently by the parties, which specify what to do with incoming messages and what to send as outgoing messages. These protocols can be initiated via an *action request* or an *incoming message* to a party. Activated parties create or update (one or more) copies of protocol runs, called *sessions*, which are technically interactive subroutines within each party. Key exchange protocols are understood in this context to be message-driven protocols in which two parties communicate and return a session key once finished. The sessions of the CK01 model are similar to the oracles of the BR model from the previous section.

The input to a key exchange protocol within party P_i, i.e., a session $\mathfrak{s}$ within the *owner* P_i, is of the form $\mathfrak{s} = (P_i, P_j, \Phi, \text{role})$, where P_j is the identity of another party, called the session *peer*, Φ is a *session identifier*, and role is either $\mathcal{I}$, for *initiator*, or $\mathcal{R}$, for *responder*. By session identifier, we mean some unique string within P_i that identifies the given session. Parties P_i and P_j of a given session s are referred to as *partners*. After receiving an activation request for a particular session, P_i will check to see that the session identifier Φ has not been used before, and only creates the session if this is the case. A party's local information specific to a particular session is stored in the *session state*, some of which is designated secret. The session $\mathfrak{s}$ outputs (P_i, P_j, Φ, κ), where κ is either the session key or, if the session was aborted, a null value. We assume that a completed session erases everything in its session state except the output. Also, a session can at any point receive the Expire command, which causes the session to delete its session key and corresponding session state. We say that two sessions of the form $(P_i, P_j, \Phi, \text{role})$ and $(P_j, P_i, \Phi, \text{role}')$ are *matching*, whether or not the roles are different. We note that if $\mathfrak{s}$ is expired, its matching session may or may not be expired. Moreover, the matching session of a completed session $\mathfrak{s}$ may or may not be completed.

As in the BR model, the CK01-adversary E has complete control of communications. Messages are sent directly to the adversary, who is in control of their delivery. E controls the creation of sessions and is free to delete, modify, create, inject, and deliver messages to the party of its choosing. The adversary is also allowed the queries $\mathsf{SessionStateReveal}$, $\mathsf{SessionKeyReveal}$, and $\mathsf{Corrupt}$. Upon receiving $\mathsf{SessionStateReveal}(\mathfrak{s})$, E knows all the contents of the session state, including any secret information. The query is noted and $\mathfrak{s}$ produces no further output. The $\mathsf{SessionKeyReveal}(\mathfrak{s})$ query enables the adversary to obtain the session key, provided $\mathfrak{s}$ actually has a session key. The $\mathsf{Corrupt}(\mathsf{P_i})$ query allows E to take over the party P_i, i.e., the adversary has access to all information in P_i's memory, including LL-keys and any session-specific information still stored. A corrupted party produces no further output. We say a session $\mathfrak{s}$ with owner P_i is *locally exposed* if the adversary has issued $\mathsf{SessionKeyReveal}(\mathfrak{s})$, $\mathsf{SessionStateReveal}(\mathfrak{s})$, or $\mathsf{Corrupt}(\mathsf{P_i})$ before $\mathfrak{s}$ is expired. We say $\mathfrak{s}$ is *exposed* if $\mathfrak{s}$ or its matching session have been locally exposed, and otherwise we say $\mathfrak{s}$ is *fresh*.

We also allow the adversary E a single $\mathsf{Test}(\mathfrak{s})$ query, which can be issued at any stage to a completed, fresh, unexpired session $\mathfrak{s}$. A bit b is then picked at random. If $b = 0$, the test oracle reveals the session key, and if $b = 1$, it generates a random value in the key space. Unlike in the BR model, E can continue to issue queries as desired, with the exception that it cannot expose the test session. At any point, the adversary can try to guess b. As before, we define $\text{GoodGuess}^E(k)$ to be the event that E correctly guesses b, and

$$\text{Advantage}^E(k) = \max\left\{0, \left|\Pr[\text{GoodGuess}^E(k)] - \frac{1}{2}\right|\right\},$$

where k is a security parameter.

This motivates the following definition of security:

Definition 4.2.1. A key exchange protocol Π in security parameter k is said to be *CK01-secure* if for any CK01-adversary E,

1. If two uncorrupted parties have completed matching sessions, these sessions produce the same key as output;
2. $\text{Advantage}^E(k)$ is negligible.

We refer the reader to [45] for a more detailed description and critique of the properties of the CK01 model, but we summarize them here. The CK01 model incorporates many of the basic security attributes and attacks discussed in Sections 3.4 and 3.5, and a CK01-secure KAP provides mutual authentication. Passive adversaries, man-in-the-middle attacks, forward secrecy, known key attacks, and unknown key share attacks are all covered by the adversarial model. Adversaries can access session-specific temporary information for a given session, and in a secure protocol, this will have no effect on the security of *other* sessions. However, the model does not allow adversaries to gain the ephemeral secrets of the test session, and so secure protocols do not necessarily have known session-specific temporary information security. Ideally, a protocol should remain secure as long as both the ephemeral and LL-keys of a given party are not compromised.

In addition, the model does not allow for key compromise impersonation attacks, since after the adversary gains the LL-key of a party through the corruption query, that party is assumed to produce no further output. Reflection attacks are also ignored, since parties cannot be both the owner and peer of a session with given identifier Φ. As adversaries are assumed not to be parties, this approach neglects malicious insiders. This implies that attackers cannot, for example, have access to a valid public key and associated certificate of their own.

4.3 Extended Canetti-Krawczyk (eCK) Model

The extended Canetti-Krawczyk (eCK) model [24] captures all of the security properties of the CK01 model, as well as resilience to KCI attacks, malicious insiders, and known session-specific temporary information attacks. The eCK model in a sense builds on the CK01 definition, but replaces the notion of matching sessions with that of matching conversations from the BR model. In addition, registration of public keys is arbitrary: there are no validity checks on the public keys (other than checking whether the public key is a non-identity element of the given group) and adversaries can register public keys on behalf of entities at any time.

As in the CK01 model, we have a finite set of parties $P_1, \ldots, P_n$ modeled by probabilistic Turing machines. The adversary, also modeled by a probabilistic Turing machine, controls all communication—parties give outgoing messages to the adversary, who has control over their delivery via the Send query. Parties are activated by Send queries, so the adversary has control over the creation of protocol runs, once again called sessions, which take place within each party. As before, we call the initiator of a session the *owner*, the responder the *peer*, and say both are *partners* of the given session. Rather than view sessions as matching according to session identifiers as in CK01, we say two sessions are *matching* if they have matching conversations in the sense of the BR model. Parties choose ephemeral keys randomly on a per-session basis, so a session's (perhaps incomplete) conversation is essentially a unique identifier, given that the probability parties will use the same ephemeral keys twice is negligible.

In addition to the Send query, the powers of the adversary are formalized through $\mathsf{EphemeralKeyReveal}$, $\mathsf{SessionKeyReveal}$, $\mathsf{StaticKeyReveal}$, and $\mathsf{Establish}$ queries. The secret information of protocols is divided into two types, ephemeral and static. Ephemeral information includes any session-specific secret information, whereas static information is compromised of the long-term secret key of the party. The query $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$ allows the adversary to obtain the ephemeral private key of the session $\mathfrak{s}$; this is not equivalent to issuing the query $\mathsf{EphemeralKeyReveal}$ on the session matching to $\mathfrak{s}$ (if it exists), as only the ephemeral information chosen by the session owner is revealed. The $\mathsf{SessionKeyReveal}(\mathfrak{s})$ query allows the adversary to obtain the session key for the specified session $\mathfrak{s}$ (so long as $\mathfrak{s}$ holds a session key). The $\mathsf{StaticKeyReveal}(\mathsf{party})$ query gives the adversary access to the LL-key of the identified party. The ability of the adversary to register a public key on behalf of a given party is modeled by $\mathsf{Establish}(\mathsf{party})$; these parties are then adversary-controlled. If the adversary has not issued an $\mathsf{Establish}$ query against a given party, we say that party is *honest*.

We also have the notion of a *fresh* session, which is considerably more complicated than in the CK01 model, in order to more appropriately describe the possible leakage of secret information to the adversary. The underlying idea is that as long as the adversary has not compromised both the LL-key and ephemeral secrets of a given involved party, the session should be secure. We provide the formal definition of a fresh session below.

Definition 4.3.1. Let $\mathfrak{s}$ be a completed session owned by party P_i with peer P_j, both of whom are honest. Let $\mathfrak{s}^*$ denote the matching session (if such a session exists). We say $\mathfrak{s}$ is *fresh* if none of the following conditions are true, where E denotes the adversary:

1. E issues a $\mathsf{SessionKeyReveal}(\mathfrak{s})$ or $\mathsf{SessionKeyReveal}(\mathfrak{s}^*)$ query (provided $\mathfrak{s}^*$ exists);
2. $\mathfrak{s}^*$ exists and E either makes queries:
   (a) both $\mathsf{StaticKeyReveal}(\mathsf{P_i})$ and $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$ or
   (b) both $\mathsf{StaticKeyReveal}(\mathsf{P_j})$ and $\mathsf{EphemeralKeyReveal}(\mathfrak{s}^*)$;
3. No matching session $\mathfrak{s}^*$ exists and E either makes queries:
   (a) both $\mathsf{StaticKeyReveal}(\mathsf{P_i})$ and $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$ or
   (b) $\mathsf{StaticKeyReveal}(\mathsf{P_j})$.

We also allow the adversary E a single $\mathsf{Test}(\mathfrak{s})$ query, which can be issued at any stage to a completed, fresh session $\mathfrak{s}$. A bit b is then picked at random. If $b = 0$, the test oracle reveals the session key, and if $b = 1$, it generates a random value in the key space. E can continue to issue queries as desired, with the requirement that the test session remain fresh. At any point, the adversary can try to guess b. Using the same notation as before, $\mathrm{GoodGuess}^E(k)$ is the event that E correctly guesses b, and

$$\mathrm{Advantage}^E(k) = \max\left\{0, \left|\Pr[\mathrm{GoodGuess}^E(k)] - \frac{1}{2}\right|\right\},$$

where k is a security parameter.

Finally, we are ready to formalize the notion of eCK security:

Definition 4.3.2. We say a key establishment protocol is *eCK-secure* if the following conditions hold:

1. If honest parties have matching sessions, these sessions output the same session key (except with negligible probability);
2. For any polynomial time adversary E, $\mathrm{Advantage}^E(k)$ is negligible.

4.4 Extending eCK

In this section, we attempt to give a framework for an eCK variant appropriate to the certificateless setting. We note that Al-Riyami and Paterson give a security definition relevant to certificateless encryption in [1, 2], but our key agreement model is based on the eCK model and is not the natural extension of their definition to key establishment protocols. We note that the treatment of the longterm secret information of the user is different in Al-Riyami and Paterson's definition, as their adversary is not allowed to know only part of a user's private key, an issue which restricts the treatment of leakage of ephemeral information. Nevertheless there are some similarities, and we owe the general adversarial model to Al-Riyami and Paterson: we consider two possible types of adversaries, namely those without the master secret key, who can replace public keys at will, and those with the master secret key, who are not allowed to replace public keys at any time. To model these, we add the following set of queries to the original eCK model:

- $\mathsf{RevealMasterKey}$: The adversary gains access to the master secret key.
- $\mathsf{ReplacePublicKey}(\mathsf{party})$: The adversary replaces the public key of the given party. Unlike in the eCK model, this does not mean the adversary has control over the party. Instead, it implies that all other parties will use the adversary's version of the party's public key, while the given party will continue to use the correct public key in any calculations.

- RevealPartialPrivateKey(party): The adversary gains access to the given party's partial private key, which is generated from the master secret key. Note that this command is redundant if the RevealMasterKey query has been issued.

- RevealSecretValue(party): The adversary gains access to the party's chosen secret value (which is used to generate the party's public key). We assume that an adversary cannot issue a RevealSecretValue query against a party which has already received the ReplacePublicKey query.

We also remove the queries Establish and StaticKeyReveal. In the original eCK model, adversary-controlled parties are those against which the adversary issued the Establish query, and we assume that all future behavior of these parties is determined by the adversary. In the certificateless setting, we consider an adversary-controlled party to be one against which the adversary has issued both the ReplacePublicKey and RevealPartialPrivateKey queries. If the RevealMasterKey query has been issued, any party issued the ReplacePublicKey query is considered to be adversary-controlled; in this way, we capture the intent of the requirement that adversaries holding the master key should not be allowed to replace public keys. As before, we say a party that is not adversary-controlled is *honest*. We have essentially replaced the StaticKeyReveal query by the RevealSecreatValue and RevealPartialPrivateKey queries. The EphemeralKeyReveal, SessionKeyReveal, Send, and Test queries remain as before.

Our notion of *matching sessions* is also slightly different. Since in general, certificateless protocols may involve participants sending their public keys as part of the message flows or publishing them to some insecure directory, we ignore any ReplacePublicKey queries in determining whether a conversation between two parties is matching. That is, a conversation is defined to be matching if it is matching once public keys, if any, have been removed from the messages.

We then modify the definition of a *fresh* session as follows:

Definition 4.4.1. Let $\mathfrak{s}$ be a completed session owned by party P_i with peer P_j, both of whom are honest. Let $\mathfrak{s}^*$ denote the matching session (if such a session exists). We say $\mathfrak{s}$ is *fresh* if none of the following conditions are true, where E denotes the adversary:

1. E issues a SessionKeyReveal($\mathfrak{s}$) or SessionKeyReveal($\mathfrak{s}^*$) query (provided $\mathfrak{s}^*$ exists);

2. $\mathfrak{s}^*$ exists and E either makes queries:

    (a) both RevealPartialPrivateKey($\mathsf{P_i}$) and RevealSecretValue($\mathsf{P_i}$) as well as EphemeralKeyReveal($\mathfrak{s}$) or

    (b) both RevealPartialPrivateKey($\mathsf{P_j}$) and RevealSecretValue($\mathsf{P_j}$) as well as EphemeralKeyReveal($\mathfrak{s}^*$);

3. No matching session $\mathfrak{s}^*$ exists and E either makes queries:
   (a) both $\mathsf{RevealPartialPrivateKey}(\mathsf{P_i})$ and $\mathsf{RevealSecretValue}(\mathsf{P_i})$ as well as $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$ or
   (b) both $\mathsf{RevealPartialPrivateKey}(\mathsf{P_j})$ and $\mathsf{RevealSecretValue}(\mathsf{P_j})$.

This definition encompasses both types of adversaries. In the case where the adversary has issued the $\mathsf{RevealMasterKey}$ query, he cannot issue replace public key queries without making the involved parties dishonest. Moreover, as this adversary automatically has access to the partial private keys of users, he is assumed unable to issue both the $\mathsf{RevealSecretValue}(\mathsf{P_i})$ and $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$ or the $\mathsf{RevealSecretValue}(\mathsf{P_j})$ and $\mathsf{EphemeralKeyReveal}(\mathfrak{s}^*)$ queries (provided $\mathfrak{s}^*$ exists). Using the same notation as in Definition 4.3.2, we have:

Definition 4.4.2. We say a certificateless key establishment protocol is *secure* if the following conditions hold:

1. If honest parties have matching sessions and no $\mathsf{ReplacePublicKey}$ queries have been issued, these sessions output the same session key (except with negligible probability).
2. For any polynomial time adversary E, $\text{Advantage}^E(k)$ is negligible.

This definition raises the question of whether such secure protocols even exist. We have not seen any examples in the literature, which may imply that, as defined, our adversary is too strong. We have already noted that the Al-Riyami Paterson KAP is vulnerable to an adversary who issues $\mathsf{EphemeralKeyReveal}$ queries on matching sessions, and we will see that even our proposed fix is not as secure as we would like. The ability to replace public keys affords enormous power, and as we show in Chapter 6, opens certificateless protocols to various types of attack. It seems that certificateless protocols that use the binding technique to defeat replacement of public keys enjoy more security, but of course lose the flexibility of the general certificateless scheme. In the most general setting, public keys can be generated before or after partial private keys, but once the binding technique is utilized, public keys must be fixed beforehand. This restriction is not particularly desirable, especially since users who wish to update their public keys are no longer free to do so at will.

Chapter 5

Related Work

The next chapter focuses heavily on the existence of key compromise impersonation attacks in key agreement, and for the purposes of our discussion it is helpful to examine relevant prior results. In particular, Strangio [42] demonstrates KCI attacks on several protocols previously thought to be resilient. His technique is similar to ours, in that he notices that it is possible to pick messages of a clever form, which, when used in place of honest messages, allow the attacker to compute the session key. The general idea is to try pick a message which will "cancel" certain terms containing the unknown ephemeral key, leaving only terms which use information gained from the target's messages and LL-key. Strangio does not consider certificateless key agreement protocols, so his attacks never involve the replacement of public keys.

Yet another example of a protocol susceptible to a clever KCI attack is the ECKE-1 protocol, introduced by Strangio [43]. The attack and proposed modifications can be found in [48] and [44]. Other work includes [47], in which a protocol of Ryu et al. provided in [32] is attacked. Ryu's protocol is vulnerable because it essentially combines a static shared secret with a Diffie-Hellman exchange, so in this example the attack is not particularly surprising. The same attack on Ryu's protocol is discussed in [50] and a revised (albeit more computationally intensive) protocol is presented. The new protocol uses asymmetry to its advantage, departing from the use of a static shared secret. In a recent publication, Lim and Lee [26] present a progression of revised versions of a deniable authentication protocol by Chou et al. Their first attempt defeats a basic KCI attack on the original protocol, but neglects to consider an insider KCI attack. The final version combats this by adding checks to ensure the correct public keys are used in computing messages and using a key derivation function (on inputs relevant to both the ephemeral and private keys) to compute the session key.

An interesting variation on the KCI attack theme in the setting of certificateless public-key cryptography is presented by Au et al [3]. In this paper, the authors remove the assumption that the KGC behaves honestly during the establishment of its own public-key pair, and thus the authors distinguish between a *malicious-but-*

passive KGC, which can be trusted to honestly generate both its own public-key pair and the partial private keys of the users and to not replace user public keys, and a more active KGC. Of course, as discussed previously, in the certificateless setting we must always rely on the KGC to not actively replace user public keys, so the authors challenge the assumption that the KGC chooses its public-key pair honestly. They analyze several types of certificateless primitives and conclude that schemes using the classic key generation techniques of Al-Riyami and Paterson are more vulnerable. They demonstrate how the KGC can target a potential user during the set-up phase of the Al-Riyami Paterson scheme (Section 3.6.4). In particular, this attack allows the KGC to determine the targeted user's secret key once the corresponding public key has been published. The attack is undetectable in the sense that the master public key of this scenario is computationally indistinguishable from an honest master public key. We present this attack in detail as it applies to the Al-Riyami Paterson key agreement protocol in Section 6.3 and suggest a fix.

We also note that work has been done on key compromise impersonation in one-pass key agreement protocols by Chalkias et al. [12], which fall somewhere between key transport and regular key agreement schemes, in that one party transfers information used to establish the session key (but does not transfer the session key itself). Since we have been concerned primarily with 2-pass key agreement protocols, we shall not concern ourselves with the details, but mention that the authors establish two classes of KCI attacks against which many one-pass protocols are vulnerable. The first, which is somewhat preventable through the use of sender verification methods, is the traditional KCI attack *without eavesdropping*. The second type of attack, for which no good solution seems to exist, involves the attacker (say, with Bob's private key) eavesdropping on a session between Alice and Bob. In this way, the attacker can either cut communications between Alice and Bob and impersonate Alice, or simply have access to the information encrypted with the given session key.

Chapter 6

Cryptanalysis of Protocols

In this chapter we explore several certificateless authenticated key agreement protocols from the literature, as well as one "self-certified" protocol (which on closer analysis actually appears to be certificateless). We analyze the security attributes of the given protocols, paying particular attention to the existence of key compromise impersonation attacks and whether or not the protocols have known temporary session-specific information security. Where possible, we demonstrate the failure of these protocols to prevent such attacks, and in one case we show the protocol to be entirely insecure.

6.1 Attack Model

We define the adversaries used in our analysis; our definitions are consistent with the adversaries the protocol designers considered in the original papers. Where possible, we also include how these attacks might be formally modeled in the suggested extension to eCK of Section 4.4.

Definition 6.1.1. We say an adversary is an *outside attacker* if the adversary does not have the master secret key. We assume an outside attacker is able to replace public keys of users.

Definition 6.1.2. We say an adversary is an *inside attacker* if the adversary has access to the master secret key. We assume an inside attacker cannot replace public keys of users.

We also consider the case of the malicious KGC mentioned in Chapter 5 and take a brief look at the related attack on the Al-Riyami Paterson KAP. We remind the reader of the definition below:

Definition 6.1.3. We say that the KGC is a *malicious KGC* if the KGC is not trusted to establish its public-key pair honestly. As with inside attackers, we assume malicious KGCs do not replace public keys of users.

6.2 Notation

We establish notation used in the following protocols.

Let q be a prime, G_1 a cyclic additive group of order q generated by P, and G_T a multiplicative group of order q. Let $e\colon G_1 \times G_1 \to G_T$ be an admissible pairing. We assume the discrete logarithm problem (DLP) is hard in both G_1 and G_T. We use various hash functions, namely $H\colon \{0,1\}^* \to \mathbb{Z}_q^*$, $H'\colon G_T \to \mathbb{Z}_q^*$, $h\colon \{0,1\}^* \to G_1$, and $h'\colon \{0,1\}^* \times G_1 \to G_1$, as well as an (appropriately defined) key derivation function, denoted kdf.

In the following protocols, we refer to the trusted third party as a KGC, with associated master secret key $s \in \mathbb{Z}_q^*$ and master public key $P_{\text{KGC}} = sP$. This notation varies somewhat from that of the original papers. In particular, we have replaced the terms 'trusted authority' and 'certification authority' in order to standardize notation.

The system parameters $\langle q, G_1, G_T, e, P, P_{\text{KGC}}, H, H', h, h' \rangle$ are made public. If the protocol flows do not include the participants' public keys, we assume the existence of a public directory for this purpose.

6.3 Al-Riyami Paterson KAP Revisited

We present the malicious KGC attack described in [3] as it applies to the Al-Riyami Paterson KAP; we refer the reader to Section 3.6.4 for the protocol specifications. This method of attack targets a specific user i before the system parameters have been established—the authors had an important personage in mind as the potential target. As usual, we assume Alice (A) is trying to agree on a key with Bob (B).

Let's say the KGC chooses to target user A. To do so, the KGC sets $P = \alpha Q_A = \alpha h(\text{ID}_A) \in G_1$ where $\alpha \in_R \mathbb{Z}_q^*$, rather than picking P to be an arbitrary generator of G_1. When A publishes his public key $\langle X_A, Y_A \rangle$, the KGC can compute S_A, the user's secret key, by the formula $S_A = \alpha^{-1} Y_A$. It is easy to see that this formula is correct, since $\alpha^{-1} Y_A = \alpha^{-1} s x_A P = \alpha^{-1} s x_A \alpha h(\text{ID}_A) = s x_A Q_A = S_A$, as desired. Thus, in this special case, the Al-Riyami Paterson KAP has key escrow.

However, our suggestion to incorporate $x_A x_B P$ into the session key appears to thwart this attack, in addition to providing known session-specific temporary information security. The KGC knows S_A, but not x_A, and so cannot compute $x_A x_B P$ unless it is allowed to replace public keys. Thus, at least in terms of key agreement protocols, the malicious KGC attack does not seem particularly strong.

Our fix is not quite as strong as we would like, however. In the context of our formal model from Section 4.4, an outside adversary (one who has not issued the RevealMasterKey query) can still mount an attack on A. He simply issues the ReplacePublicKey query on B and uses the EphemeralKeyReveal query on both the test session and its matching session. The initial vulnerability of the protocol to the leakage of ephemeral information allows the attacker to compute K_A. He can compute A's version of x_Ax_BP by using his knowledge of the replaced public key. Specifically, suppose he chooses to replace B's public key with $\langle x'_BP, x'_BsP\rangle$ for some $x'_B \in_R \mathbb{Z}_q^*$. Then A will compute x_Ax_BP as $x_A(x'_BP) = x'_BX_A$. It is clear that the adversary will be able to distinguish the session key held by A from a randomly chosen element of the keyspace.

6.4 Mandt KAP

6.4.1 Protocol Summary

We outline the basic certificateless key agreement protocol presented by Mandt [27], which relies on the difficulty of the bilinear Diffie-Hellman problem (BDH).

Each user i chooses a partial public-key pair (x_i, P_i), where $x_i \in_R \mathbb{Z}_q^*$ is the user's secret value and $P_i = x_iP$ is his public key. Each user also has $Q_i = h(\mathrm{ID}_i) \in G_1$ as an additional partial public key and associated partial private key sQ_i, which is distributed by the KGC. The user's full private key, denoted by S_i, is $S_i = sQ_i + x_iQ_i = (s + x_i)Q_i$. For ease of notation, we let $D_i = P_{\mathrm{KGC}} + P_i$. For this protocol, our key derivation function is of the form $\mathrm{kdf}\colon G_T \times G_1 \times G_1 \to \{0,1\}^k$ for some $k \in \mathbb{Z}$.

Alice (A) and Bob (B) agree on a session key as follows. (Refer to Figure 6.1.)

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = aP & & b \in_R \mathbb{Z}_q^*;\ T_B = bP \\
 & \xrightarrow{T_A,\, P_A} & \\
 & \xleftarrow{T_B,\, P_B} & \\
K_A = e(Q_B, D_B)^a e(S_A, T_B) & & K_B = e(S_B, T_A)e(Q_A, D_A)^b \\
\mathrm{kdf}(K_A||aT_B||x_AP_B) & & \mathrm{kdf}(K_B||bT_A||x_BP_A).
\end{array}$$

Figure 6.1: Mandt KAP

1. Alice picks $a \in_R \mathbb{Z}_q^*$ and sends $T_A = aP$ and P_A to Bob.
2. Bob picks $b \in_R \mathbb{Z}_q^*$ and sends $T_B = bP$ and P_B to Alice.
3. Alice checks that $P_B \in G_1^*$. She terminates the protocol run if this check fails, and otherwise computes $K_A = e(Q_B, D_B)^a e(S_A, T_B)$. She computes the session key as $\mathrm{kdf}(K_A||aT_B||x_AP_B)$.
4. Bob checks $P_A \in G_1^*$. He terminates the protocol run if this check fails, and otherwise computes $K_B = e(S_B, T_A)e(Q_A, D_A)^b$. He uses $\mathrm{kdf}(K_B||bT_A||x_BP_A)$ as the session key.

We note that

$$\begin{aligned}
K_A &= e(Q_B, D_B)^a e(S_A, T_B) \\
&= e(Q_B, P_{\text{KGC}} + P_B)^a e(S_A, T_B) \\
&= e(Q_B, (s + x_B)P)^a e((s + x_A)Q_A, bP) \\
&= e((s + x_B)Q_B, aP) e(Q_A, (s + x_A)P)^b \\
&= e(S_B, T_A) e(Q_A, P_{\text{KGC}} + P_A)^b \\
&= e(S_B, T_A) e(Q_A, D_A)^b \\
&= K_B,
\end{aligned}$$

so the protocol is consistent.

Mandt designed this protocol with the goal of improving the Al-Riyami Paterson KAP. His paper [27] places considerable emphasis on the importance of known session-specific temporary information security, which as we have discussed is not possessed by the Al-Riyami Paterson KAP. Mandt [27] argues heuristically that his KAP has this property, as well as known session key security, weak forward secrecy, and resistance to key compromise impersonation, unknown key share, and key control attacks. We show in the next section that the protocol actually admits both a key compromise impersonation and a known session-specific temporary information security attack.

In addition to the basic protocol, Mandt provides two variations, one in which the protocol participants use separate KGCs and one which provides key confirmation. The former is almost identical to the basic protocol, with the substitution of the different master keys where appropriate. The latter uses a MAC keyed under a value related to that of the session key, i.e., derived using a different key derivation function on the same inputs. Both versions are vulnerable to the attacks outlined below.

6.4.2 Analysis and Attack

We first show the protocol is vulnerable to a key compromise impersonation attack from an outside attacker. In fact, it suffices for the adversary Eve (E) to know a user's secret value x_i; she does not need the partial private key provided by the KGC. As the flows in the protocol are symmetric, it does not matter whether we attempt to impersonate the initiator or responder of the protocol run. Formally, we describe this attack sequence on a session held by Alice as $\mathsf{RevealSecretValue(A)}$ and $\mathsf{ReplacePublicKey(B)}$ where no matching session exists, i.e., the adversary uses the Send query to send messages purportedly from B to A.

Assume Eve has access to x_A. She impersonates Bob to Alice by selecting $\beta, b \in \mathbb{Z}_q^*$ and sending $P_B^* = -P_{\text{KGC}} + \beta P$ for B's public key and $T_B = bP$ as usual. Since $P_B^* \in G_1^*$, Alice computes

$$\begin{aligned} K_A &= e(Q_B, P_{\mathrm{KGC}} + P_B)^a e(S_A, T_B) \\ &= e(Q_B, P_{\mathrm{KGC}} - P_{\mathrm{KGC}} + \beta P)^a e((s + x_A)Q_A, bP) \\ &= e(Q_B, \beta P)^a e(bQ_A, (s + x_A)P) \\ &= e(\beta Q_B, aP) e(bQ_A, P_{\mathrm{KGC}} + P_A). \end{aligned}$$

Alice then derives the session key from $\mathrm{kdf}(K_A||aT_B||x_AP_B)$.

As Eve chooses both β and b, she can compute K_A and $aT_B = bT_A$. Note that we have not needed knowledge of x_A up to this point. The only reason we need x_A is to compute x_AP_B to input into the kdf, so this term is the only preventative measure against a man-in-the-middle attack similar to the KCI attack above. We see no clever substitution for P_B which allows for both the calculation of $e(Q_B, P_{\mathrm{KGC}}+P_B)^a$ and x_AP_B, however. The heuristic argument against KCI attacks given in [27] fails to consider the scenario where the public key for B is replaced. A subsequent paper [28] on the same protocol recognizes the possibility of replacing public keys, but still fails to account for the above attack.

Mandt also claims the protocol has the property of known session-specific temporary information security. However, it is an easy matter to mount an outsider man-in-the-middle attack if armed with a and b. Suppose Eve substitutes $P_A^* = \alpha P$ for P_A and $P_B^* = \beta P$ for P_B in the protocol run, where $\alpha, \beta \in_R \mathbb{Z}_q^*$. From above we have that A and B will compute $K_A = e(Q_B, P_{\mathrm{KGC}} + P_B^*)^a e(Q_A, P_{\mathrm{KGC}} + P_A)^b$ and $K_B = e(Q_B, P_{\mathrm{KGC}} + P_B)^a e(Q_A, P_{\mathrm{KGC}} + P_A^*)^b$, respectively, so Eve will have no problem calculating K_A and K_B if she knows a and b.

Moreover, we have $x_AP_B^* = x_A\beta P = \beta P_A$, so Eve will establish the session key $\mathrm{kdf}(K_A||aT_B||x_AP_B^*) = \mathrm{kdf}(K_A||abP||\beta P_A)$ with A. Similarly she will establish $\mathrm{kdf}(K_B||abP||\alpha P_B)$ as the session key with B. Thus the protocol fails to satisfy the known session-specific temporary information security property against outside attackers. If, on the other hand, the attacker is passive or cannot replace public keys, the protocol remains secure. The variants of the protocol are similarly vulnerable. Note that this is essentially the same attack mentioned in Section 6.3 on the improved version of the Al-Riyami Paterson KAP.

Interestingly, the Mandt protocol is almost identical to the Wang-Cao KAP presented in Section 3.6.3. The main difference between the protocols is that in the latter, the private keys are bound to the public keys, so the attacks presented above are not possible in an ideal protocol specification of the Wang-Cao KAP, whereas Mandt's protocol allows adversaries to easily replace public keys. We achieve the same scheme if we apply the binding technique of Al-Riyami and Paterson to Mandt's protocol, although for cheating to be evident, we must again require the partial private keys (or users' certificates) to be public.

6.5 Wang-Cao-Wang KAP

6.5.1 Protocol Summary

We outline the certificateless key agreement protocol provided by Wang, Cao, and Wang [49].

Each user i chooses a partial public-key pair (x_i, P_i), where $x_i \in_R \mathbb{Z}_q^*$ is the user's secret value and $P_i = x_i P$ is his public key. Each user also has $Q_i = h(\mathrm{ID}_i) \in G_1$ as an additional partial public key, and an associated partial private key $S_i = sQ_i$ which is distributed by the KGC. The user's full private key is the pair (x_i, S_i).

Alice (A) and Bob (B) agree on a session key as follows. (Refer to Figure 6.2.)

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = aP & & b \in_R \mathbb{Z}_q^*;\ T_B = bP \\
 & \xrightarrow{T_A} & \\
 & \xleftarrow{T_B} & \\
V_A = e(S_A, Q_B) & & V_B = e(S_B, Q_A) \\
K_A = \mathrm{kdf}(A, B, V_A, aP_B, x_A T_B) & & K_B = \mathrm{kdf}(A, B, V_B, x_B T_A, bP_A)
\end{array}$$

Figure 6.2: Wang-Cao-Wang KAP

1. Alice picks $a \in_R \mathbb{Z}_q^*$ and sends $T_A = aP$ to Bob.
2. Bob picks $b \in_R \mathbb{Z}_q^*$ and sends $T_B = bP$ to Alice.
3. Alice computes $V_A = e(S_A, Q_B)$.
4. Bob computes $V_B = e(S_B, Q_A)$.
5. Alice checks that $T_B \in G_1^*$. She terminates the protocol run if this check fails, and otherwise computes the session key $K_A = \mathrm{kdf}(A, B, V_A, aP_B, x_A T_B)$.
6. Bob checks that $T_A \in G_1^*$. He terminates the protocol run if this check fails, and otherwise computes the session key $K_B = \mathrm{kdf}(A, B, V_B, x_B T_A, bP_A)$.

Note that $V_A = e(S_A, Q_B) = e(sQ_A, Q_B) = e(Q_A, sQ_B) = e(Q_A, S_B) = V_B$. Moreover, $aP_B = ax_B P = x_B(aP) = x_B T_A$ and similarly $bP_A = x_A T_B$, so $K_A = K_B$ as desired.

The authors claim their protocol is secure against both man-in-the-middle attacks mounted by the KGC and KCI attacks. Since, as discussed in Section 3.5, all certificateless key agreement protocols are vulnerable to a KGC man-in-the-middle attack, the first claim is certainly false. We challenge the second claim in the following section.

We mention that the authors also claim their protocol has known-key security, forward secrecy, unknown key-share resistance, and key control.

6.5.2 Analysis and Attack

We first observe that use of the static shared secret $V_A = V_B$ prevents the formal attack outlined in Section 6.3, where knowledge of the matching sessions' ephemeral keys and one public key replacement allows a successful attack. This term implies a successful attacker must have access to at least one of the participating party's partial private keys. However, the protocol does not guard against an adversary who, for a test session $\mathfrak{s}$ with owner A and matching session $\mathfrak{s}^*$, issues the queries $\mathsf{RevealPartialPrivateKey(A)}$, $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$, and $\mathsf{EphemeralKeyReveal}(\mathfrak{s}^*)$. The adversary will be able to compute the session key $\mathrm{kdf}(A, B, V_A, aP_B, bP_A)$.

We mount an outsider KCI attack as follows. As with the KCI attack on the Mandt protocol, we can express this attack using the formal terminology of Section 4.4. The adversary chooses a test session owned by B and takes advantage of the Send, $\mathsf{RevealPartialPrivateKey(B)}$ and $\mathsf{ReplacePublicKey(A)}$ queries. The (informal) details follow.

Given that the users' public keys are published in a directory, we assume that a user initiating the key agreement protocol may have already looked up the public key of the other party. In this case, our attacker might not be able to modify the public key of the responder while remaining undetected, thereby thwarting the following attack. Thus we assume that Eve is attempting to impersonate Alice to Bob, where Alice is the initiator and Bob is the responder.

For a KCI attack on Bob, we would generally assume our attacker Eve has access to all of B's private key, that is, both x_B and S_B. Here we show that it is sufficient for Eve to have S_B.

Eve proceeds by sending $T_A = aP$ as usual (for her own choice of $a \in_R \mathbb{Z}_q^*$). She completes the attack by replacing A's public key entry with $P_A^* = \alpha P$ for some $\alpha \in_R \mathbb{Z}_q^*$.

Note that Eve can easily compute $V_B = e(S_B, Q_A)$, as she has access to S_B. Recall that B will use $x_B T_A$ and bP_A^* in his computation of the secret key. Since $x_B T_A = aP_B$ and $bP_A^* = b\alpha P = \alpha T_B$, Eve can compute these as well. (She has chosen a and α, P_B is public, and B sends T_B to A during the protocol run.) Thus Eve can compute $K_B = \mathrm{kdf}(A, B, V_B, x_B T_A, bP_A^*)$, as desired. It is worth stressing that Eve cannot succeed without knowledge of S_B, as without it she cannot compute V_B.

6.6 Shao KAP

6.6.1 Protocol Summary

We outline the key agreement protocol provided by Shao [37]. Shao claims his protocol is self-certified, but we disagree. The scheme is much closer to a certificateless protocol than Girault's idea of a self-certified key agreement protocol.

Each user i chooses a partial public-key pair (x_i, P_i), where $x_i \in_R \mathbb{Z}_q^*$ is the user's secret value and $P_i = x_i P$ is his public key. Each user also has $Q_i = h'(\mathrm{ID}_i, P_i) \in G_1$ as an additional partial public key, and an associated partial private key $S_i = sQ_i$ which is distributed securely by the KGC. In this scheme the user i checks that $Q_i = h'(\mathrm{ID}_i, P_i)$ and $e(S_i, P) = e(Q_i, P_{\mathrm{KGC}})$ before participating in any protocol runs. This property seems to be Shao's motivation for claiming the public keys are self-certified. However, unlike in Girault's model, the KGC has partial key escrow (in that it computes the partial private keys) and requires a secure channel between itself and the users. Also, users cannot be sure that they have the correct public key of another party, and so the adversarial model is equivalent to that of certificateless schemes.

Alice (A) and Bob (B) agree on a session key as follows. (Refer to Figure 6.3.)

$$
\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^*;\ T_A = aP_B & & b \in_R \mathbb{Z}_q^*;\ T_B = bP_A \\
 & \xrightarrow{T_A,\, \mathrm{ID}_A} & \\
 & \xleftarrow{T_B,\, \mathrm{ID}_B} & \\
Q_B = h'(\mathrm{ID}_B, P_B) & & Q_A = h'(\mathrm{ID}_A, P_A) \\
e_A = H'(e(S_A, Q_B))ax_A^{-1} \pmod q & & e_B = H'(e(Q_A, S_B))bx_B^{-1} \pmod q \\
K_A = e_A T_B & & K_B = e_B T_A \\
\mathrm{kdf}(K_A \| \mathrm{ID}_A \| \mathrm{ID}_B) & & \mathrm{kdf}(K_B \| \mathrm{ID}_A \| \mathrm{ID}_B)
\end{array}
$$

Figure 6.3: Shao KAP

1. Alice picks $a \in_R \mathbb{Z}_q^*$ and sends ID_A and $T_A = aP_B$ to Bob.

2. Bob picks $b \in_R \mathbb{Z}_q^*$ and sends ID_B and $T_B = bP_A$ to Alice.

3. Alice checks that $T_B \in G_1^*$. She terminates the protocol run if this check fails, and otherwise computes $Q_B = h'(\mathrm{ID}_B, P_B)$, $e_A = H'(e(S_A, Q_B))ax_A^{-1} \pmod q$, and $K_A = e_A T_B$. She uses $\mathrm{kdf}(K_A \| \mathrm{ID}_A \| \mathrm{ID}_B)$ as the session key.

4. Bob checks that $T_A \in G_1^*$. He terminates the protocol run if this check fails, and otherwise computes $Q_A = h'(\mathrm{ID}_A, P_A)$ and $e_B = H'(e(Q_A, S_B))bx_B^{-1} \pmod q$. He sets $K_B = e_B T_A$ and uses $\mathrm{kdf}(K_B \| \mathrm{ID}_A \| \mathrm{ID}_B)$ as the session key.

We note that

$$\begin{aligned}
K_A &= e_A T_B \\
&= H'(e(S_A, Q_B))ax_A^{-1}bP_A \\
&= H'(e(S_A, Q_B))ax_A^{-1}bx_AP \\
&= H'(e(S_A, Q_B))abP \\
&= H'(e(S_A, Q_B))bx_b^{-1}ax_BP \\
&= e_BT_A = K_B,
\end{aligned}$$

so the protocol is correct.

Shao claims that the protocol provides forward secrecy, known-key security, and resilience to a "masquerade attack," which refers to KCI attacks where not *all* of the user's private key is compromised. We launch a KCI attack in the next section and note that the author should have claimed weak forward secrecy, as discussed in Section 3.4. Moreover, although at first glance it seems that Shao's protocol has applied the binding technique of Al-Riyami and Paterson mentioned in Section 2.5, given his definition of Q_i for user i, this is not quite the case. Shao's protocol relies in an essential way on the secrecy of the partial private keys; if we make these keys public, the scheme reduces to the basic Diffie-Hellman KAP (with the extra term $H'(e(S_A, Q_B))$ that anyone can compute).

6.6.2 Analysis and Attack

We first observe that this protocol, like that of Section 6.5, does not hold up to the following formal attack. Letting $\mathfrak{s}$ denote the test session with owner A and matching session $\mathfrak{s}^*$, suppose the adversary issues the queries $\mathsf{RevealPartialPrivateKey}(\mathsf{A})$, $\mathsf{EphemeralKeyReveal}(\mathfrak{s})$, and $\mathsf{EphemeralKeyReveal}(\mathfrak{s}^*)$. The adversary will be able to compute $H'(e(S_A, Q_B))abP$, and hence the session key as well.

Let us now consider Shao's claim that the protocol is secure provided not all of the user's private key (x_i, S_i) is compromised. We show the protocol is in fact vulnerable in the scenario where S_i is known, but x_i and s remain secure.

We launch an outsider key compromise impersonation attack on Alice with the knowledge of S_A, but not x_A, as follows. Since knowing $S_A = sQ_A = sH'(\mathrm{ID}_A, x_AP)$ is not the same as knowing s, replacing public keys is permissible in this scenario. As the protocol messages are symmetric, there is a corresponding attack on Bob, and thus it does not matter whether or not the attacker initiates the protocol run. The formal queries needed for this attack are similar to the KCI attack outlined in Section 6.5, so we do not mention them here.

Our attacker Eve replaces Bob's public key with $P_B^* = \beta P$ for $\beta \in_R \mathbb{Z}_q^*$ of her choosing. She then follows the protocol and sends ID_B and $T_B = bP_A$ for some $b \in_R \mathbb{Z}_q^*$. Alice will then compute $Q_B^* = h'(\mathrm{ID}_B, P_B^*)$ and $e_A^* = H'(e(S_A, Q_B^*))ax_A^{-1}$ (mod q).

Alice calculates the session secret as

$$\begin{aligned} K_A &= e_A^* T_B \\ &= H'(e(S_A, Q_B^*)) a x_A^{-1} b P_A \\ &= H'(e(S_A, Q_B^*)) b a x_A^{-1} x_A P \\ &= H'(e(S_A, Q_B^*)) b a P. \end{aligned}$$

We see that Eve can compute $H'(e(S_A, Q_B^*))b$, as she possesses S_A and chooses b herself. Moreover, since A sends $T_A = a\beta P$ in the first round (and Eve knows β), Eve can compute aP and thus K_A.

6.7 Shi-Li KAP

6.7.1 Protocol Summary

We outline Shi and Li's [38] certificateless key agreement protocol.

We set $g = e(P, P)$ and assume our key derivation function is of the form $\mathrm{kdf}\colon G_T \to \{0,1\}^k$ for some $k \in \mathbb{Z}$.

Each user i chooses a partial public-key pair (x_i, P_i), where $x_i \in_R \mathbb{Z}_q^*$ is the user's secret value and $P_i = g^{x_i}$ is his public key. Each user also has $Q_i = \frac{1}{H(\mathrm{ID}_i)+s}P$ as an additional partial private key, which is distributed securely by the KGC, and an associated partial private key $S_i = x_i Q_i$ computed by the user. Here, by $Q_i = \frac{1}{H(\mathrm{ID}_i)+s}P$, we mean that Q_i is a fixed element of G_1 satisfying $(H(\mathrm{ID}_i) + s)Q_i = P$.

Alice (A) and Bob (B) agree on a session key as follows. (Refer to Figure 6.4.)

$$\begin{array}{ccc}
A & & B \\
a \in_R \mathbb{Z}_q^* & & b \in_R \mathbb{Z}_q^* \\
T_A = a(H(\mathrm{ID}_B)P + P_{\mathrm{KGC}}) & & T_B = b(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}}) \\
 & \xrightarrow{T_A, P_A} & \\
 & \xleftarrow{T_B, P_B} & \\
K_A = e(T_B, S_A)P_B^a & & K_B = e(T_A, S_B)P_A^b \\
\mathrm{kdf}(K_A) & & \mathrm{kdf}(K_B)
\end{array}$$

Figure 6.4: Shi-Li KAP

1. Alice picks $a \in_R \mathbb{Z}_q^*$ and sends P_A and $T_A = a(H(\mathrm{ID}_B)P + P_{\mathrm{KGC}})$ to Bob.

2. Bob picks $b \in_R \mathbb{Z}_q^*$ and sends P_B and $T_B = b(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}})$ to Alice.

3. Alice checks that $P_B \in G_T$. She terminates the protocol run if this check fails, and otherwise computes $K_A = e(T_B, S_A)P_B^a$. She uses $\mathrm{kdf}(K_A)$ as the session key.

4. Bob checks that $P_B \in G_T$. He terminates the protocol run if this check fails, and otherwise computes $K_B = e(T_A, S_B)P_A^b$. He uses $\mathrm{kdf}(K_B)$ as the session key.

We note that

$$\begin{aligned}
K_A &= e(T_B, S_A)P_B^a \\
&= e(b(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}}), x_A Q_A)P_B^a \\
&= e\left(b(H(\mathrm{ID}_A)P + sP), \frac{x_A}{H(\mathrm{ID}_A) + s}P\right) P_B^a \\
&= e(bP, x_A P)P_B^a \\
&= e(P, P)^{bx_A} g^{ax_B} \\
&= g^{bx_A} g^{ax_B} \\
&= P_A^b e(aP, x_B P) \\
&= P_A^b e\left(a(H(\mathrm{ID}_B)P + P_{\mathrm{KGC}}), \frac{x_B}{H(\mathrm{ID}_B) + s}P\right) \\
&= P_A^b e(T_A, S_B) \\
&= K_B,
\end{aligned}$$

so the protocol is correct.

We emphasize that the session key is derived directly from $g^{bx_A} g^{ax_B} = P_A^b P_B^a$, so this protocol does not have known session-specific temporary information security. We argue that, given the key relies only on poorly-verified public information and the ephemeral keys chosen by A and B, the protocol design does not efficiently make use of the users' longterm secret keys and therefore opens itself to attack.

The authors claim this protocol provides implicit key authentication, known session key security, partial forward secrecy, key compromise impersonation resistance, and unknown key share resistance. We demonstrate that the protocol is entirely insecure by mounting a man-in-the-middle attack.

6.7.2 Analysis and Attack

We show the protocol fails to provide implicit key authentication by demonstrating a man-in-the-middle attack by an outside attacker. Our attacker Eve intercepts Alice's $\langle T_A, P_A \rangle$ and instead sends $\langle T_A^*, P_A^* \rangle$ to Bob. Here $T_A^* = a^*(H(\mathrm{ID}_B)P + P_{\mathrm{KGC}})$ and $P_A^* = e(\alpha(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}}), P)$ for $a^*, \alpha \in_R \mathbb{Z}_q^*$ of Eve's choosing.

Similarly Eve replaces Bob's message $\langle T_B, P_B\rangle$ with $\langle T_B^*, P_B^*\rangle$, where $T_B^* = b^*(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}})$ and $P_B^* = e(\beta(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}}), P)$ for $b^*, \beta \in_R \mathbb{Z}_q^*$ of her choosing.

Notice that $P_A^* \in G_T$, so Bob will compute

$$\begin{aligned}
K_B &= e(T_A^*, S_B)(P_A^*)^b \\
&= e(a^*(H(\mathrm{ID}_B)P + P_{\mathrm{KGC}}), \frac{x_B}{H(\mathrm{ID}_B)+s}P)e(\alpha(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}}), P)^b \\
&= e(a^*P, x_BP)e(b(H(\mathrm{ID}_A)P + P_{\mathrm{KGC}}), \alpha P) \\
&= g^{x_Ba^*}e(T_B, \alpha P) \\
&= P_B^{a^*}e(T_B, \alpha P).
\end{aligned}$$

As Eve chooses both a^* and α, she can compute K_B. Similarly, Eve will be able to compute Alice's key $K_A = P_A^{b^*}e(T_A, \beta P)$. We have therefore shown the protocol to be insecure. The corresponding formal attack on the protocol is modeled by picking a test session with owner A and using the ReplacePublicKey(B) and Send queries to alter the messages sent by B to A. As shown above, the adversary will be able to compute the session key.

If we transform this protocol into a certificate-based protocol, whereby the public keys of users are bound to the corresponding partial private keys and the latter are used as public certificates, the scheme appears to be secure. The Shi-Li KAP provides a compelling example of the problems associated with avoiding public-key certification altogether—an adversary able to replace public keys is devastating to the security of the scheme.

Chapter 7

Conclusions and Future Work

This thesis has presented the current methods used to generate and validate public keys and issues relating to the security of key agreement protocols based on these techniques, particularly with respect to the amount of trust invested in the KGC. We have focused on certificateless key agreement schemes, as these were designed to maintain the advantages of ID-based systems while minimizing user trust in the KGC.

We have given a possible security model for certificateless key agreement protocols based on the eCK model in Chapter 4, and noted in Chapter 6 that none of the schemes we have found in the literature meet this security definition. This leads to several questions, among them:

- Is it possible to design a certificateless key agreement scheme that meets the requirements of Definition 4.4.2 or is our security definition too strict? How can we strengthen the security of the protocols mentioned in Chapter 6?
- Given that our security definition is a natural extension of the eCK model, how practical are current certificateless protocols? Most of the protocols we have analyzed falsely claim KCI resilience, which (if true) would be an improvement in comparison to, for example, the security of the commonly-used Unified Model KAP. The only scheme that appears to have KCI resilience is the Al-Riyami Paterson KAP, the original version of which fails to have known session-specific temporary-information security. Even with our fix, this protocol does not hold up against an adversary able to reveal ephemeral information and replace public keys. Overall, is the avoidance of traditional PKI worth the weakened security of these schemes?
- Is the user flexibility of certificateless schemes worth the corresponding vulnerabilities brought on by an adversary's ability to replace public keys, or is the use of Al-Riyami and Paterson's binding technique and certificate-based schemes a more promising approach to protocol design?

Appendix A

CL-PKE Algorithms

For completeness, we provide the formal definition of a CL-PKE system below [1, 2].

Definition. A CL-PKE scheme is specified by the following algorithms:

- Setup: A probabilistic algorithm that takes as input the security parameter k and generates the system parameters params and the master public/secret key pair $(\mathrm{mpk}, \mathrm{msk})$. The system parameters specify the set of possible plaintexts $\mathcal{P}$ and ciphertexts $\mathcal{C}$.

- $\mathsf{PartialPrivateKeyExtract}$: A deterministic algorithm that takes as inputs an entity i's identity $\mathrm{ID}_i \in \{0,1\}^*$, params, $(\mathrm{mpk}, \mathrm{msk})$, and outputs a user's partial private key D_i.

- $\mathsf{SetSecretValue}$: A probabilistic algorithm that takes params as inputs and generates a user i's secret value x_i.

- $\mathsf{SetPrivateKey}$: A deterministic algorithm that takes params, D_i, and x_i as input and outputs user i's full private key S_i.

- $\mathsf{SetPublicKey}$: A deterministic algorithm that takes params and x_i as input and outputs user i's public key P_i.

- $\mathsf{Encrypt}$: A probabilistic algorithm that takes params, P_i, ID_i, and $m \in \mathcal{P}$ as inputs and outputs $c \in \mathcal{C}$ or a null value, indicating failure.

- $\mathsf{Decrypt}$: A deterministic algorithm that takes params, S_i, and $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{P}$ or a null value, indicating failure.

As usual, we require $\mathsf{Decrypt}(\mathsf{params}, S_i, \mathsf{Encrypt}(\mathsf{params}, P_i\, \mathrm{ID}_i, m)) = m$. Normally the algorithms Setup and $\mathsf{PartialPrivateKeyExtract}$ are run by the KGC. The entity i usually runs $\mathsf{SetSecretValue}$, $\mathsf{SetPrivateKey}$, and $\mathsf{SetPublicKey}$ for himself; x_i and S_i are generally known only to entity i.

We note that the algorithms Setup, PartialPrivateKeyExtract, SetSecretValue, SetPrivateKey, and SetPublicKey are not specific to encryption schemes, but are applicable to all certificateless primitives. We have made the inputs and outputs of these algorithms clear in our descriptions of certificateless protocols, although we do not preserve the exact notation given above, except for the original Al-Riyami Paterson KAP presented in Section 3.6.4.

References

[1] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In C. S. Laih, editor, *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer Berlin / Heidelberg, 2003. Full version available at `http://eprint.iacr.org/2003/126`. 12, 13, 25, 26, 36, 54

[2] Sattam S. Al-Riyami and Kenneth G. Paterson. CBE from CLE-PKE: A generic construction and efficient schemes. In *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 398–415. Springer Berlin / Heidelberg, 2005. 12, 13, 36, 54

[3] Man Ho Au, Yi Mu, Jing Chen, Duncan S. Wong, Joseph K. Liu, and Guomin Yang. Malicious KGC attacks in certificateless cryptography. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, Computer and Communications security*, pages 302–311. ACM Press, 2007. 39, 42

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer Berlin / Heidelberg, 1993. 30

[5] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. In *STOC '95: Proceedings of the 27th annual ACM Symposium on Theory of Computing*, pages 57–66. ACM Press, 1995. 30

[6] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer Berlin / Heidelberg, 1997. 21, 30

[7] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001. 11

[8] Dan Boneh, Antoine Joux, and Phong Q. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In *Advances in Cryptology - ASIACRYPT 2000*, Lecture Notes in Computer Science, pages 30–43. Springer Berlin / Heidelberg, 2000. 6

[9] Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and Kenneth G. Paterson. Efficient one-round key exchange in the standard model. In *Information Security and Privacy*, volume 5107 of *Lecture Notes in Computer Science*, pages 71–84. Springer Berlin / Heidelberg, 2008. 29

[10] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 4

[11] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. Cryptology ePrint Archive, Report 2001/040, 2001. `http://eprint.iacr.org/`. 30, 32

[12] Konstantinos Chalkias, Foteini Mpaldimtsi, Dimitrios Hristu-Varsakelis, and George Stephanides. On the key-compromise impersonation vulnerability of one-pass key establishment protocols. In *International Conference on Security and Cryptography (SECRYPT 2007)*, 2007. 40

[13] L. Chen, Z. Cheng, and N.P. Smart. Identity-based key agreement protocols from pairings. Cryptology ePrint Archive, Report 2006/199, 2006. `http://eprint.iacr.org/`. 29

[14] Liqun Chen and Caroline Kudla. Identity-based authenticated key agreement protocols from pairings. In *Proc. 16th IEEE Security Foundations Workshop*, pages 219–233. IEEE Computer Society Press, 2003. 11, 23

[15] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. 16

[16] Whitfield Diffie, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992. 19

[17] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Appl. Math.*, 156(16):3113–3121, 2008. 5

[18] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2653 of *Lecture Notes in Computer Science*, pages 272–293. Springer Berlin / Heidelberg, 2003. 10, 12

[19] Marc Girault. Self-certified public keys. In *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 490–497. Springer Berlin / Heidelberg, 1991. 11, 13, 27

[20] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. System Sci.*, 28(2):270–299, 1984. 29, 30

[21] P. Gutmann. PKI: It's not dead, just resting. *IEEE Computer*, 35(8):41–49, 2002. 10

[22] Bo Gyeong Kang and Je Hong Park. Is it possible to have CBE from CL-PKE? Cryptology ePrint Archive, Report 2005/431, 2005. `http://eprint.iacr.org/`. 12

[23] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer Berlin / Heidelberg, 2005. 18

[24] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2007. 34

[25] Chae Hoon Lim and Pil Joon Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 249–263. Springer Berlin / Heidelberg, 1997. 25

[26] Meng-Hui Lim, Sanggon Lee, and Hoonjae Lee. Cryptanalysis on improved Chou et al.'s ID-based deniable authentication protocol. In *ICISS '08: Proceedings of the 2008 International Conference on Information Science and Security*, pages 87–93, Washington, DC, USA, 2008. IEEE Computer Society. 39

[27] Tarjei K. Mandt. Certificateless authenticated two-party key agreement protocols. Master's thesis, Gjøvik University College, Department of Computer Science and Media Technology, 2006. 18, 26, 43, 44, 45

[28] Tarjei K. Mandt and Chik How Tan. Certificateless authenticated two-party key agreement protocols. In *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues*, volume 4435 of *Lecture Notes in Computer Science*, pages 37–44. Springer Berlin / Heidelberg, 2008. 45

[29] Alfred Menezes and Berkant Ustaoğlu. Security arguments for the UM key agreement protocol in the NIST SP 800-56A standard. In *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, Computer and Communications security*, pages 261–270, New York, NY, USA, 2008. ACM. 21

[30] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997. 2, 5, 10, 17, 20, 28

[31] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. 6

[32] E. K. Ryu, E. J. Yoon, and K. Y. Yoo. An efficient ID-based authenticated key agreement protocol from pairings. In *NETWORKING 2004, Networking*

Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications, volume 3042 of *Lecture Notes in Computer Science*, pages 1458–1463. Springer Berlin / Heidelberg, 2004. 11, 18, 39

[33] Shahrokh Saeednia. Identity-based and self-certified key-exchange protocols. In *Information Security and Privacy*, volume 1270 of *Lecture Notes in Computer Science*, pages 303–313. Springer Berlin / Heidelberg, 1997. 11

[34] Shahrokh Saeednia. A note on Girault's self-certified model. *Inf. Process. Lett.*, 86(6):323–327, 2003. 11, 28

[35] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairings. In *Proceedings of the Symposium on Cryptography and Information Security - SCIS*, 2000. 18

[36] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Berlin / Heidelberg, 1985. 10

[37] Zu-hua Shao. Efficient authenticated key agreement protocol using self-certified public keys from pairings. *Wuhan University Journal of Natural Sciences*, 10(1):267–270, 2005. 47

[38] Yijuan Shi and Jianhua Li. Two-party authenticated key agreement in certificateless public key cryptography. *Wuhan University Journal of Natural Sciences*, 12(1):71–74, 2007. 50

[39] Joseph H. Silverman. *The Arithmetic of Elliptic Curves.* Springer Science + Business Media, New York, NY, USA, 1986. 5

[40] N. P. Smart. An identity based authenticated key agreement protocol based on the Weil pairing. *Electronics Letters*, 38:630–632, 2002. 11, 23

[41] Douglas R. Stinson. *Cryptography: Theory and Practice.* Chapman and Hall/CRC, Boca Raton, 3 edition, 2006. 2, 5

[42] Maurizio A. Strangio. On the resilience of key agreement protocols to key compromise impersonation. Cryptology ePrint Archive, Report 2006/252, 2006. `http://eprint.iacr.org/`. 39

[43] Maurizio Adriano Strangio. Efficient Diffie-Hellmann two-party key agreement protocols based on elliptic curves. In *Proceedings of the* 20^{th} *ACM Symposium of Applied Computing (SAC)*, pages 324–331, 2005. 39

[44] Maurizio Adriano Strangio. Revisiting an efficient elliptic curve key agreement protocol. Cryptology ePrint Archive, Report 2007/081, 2007. `http://eprint.iacr.org/`. 39

[45] Berkant Ustaoğlu. *Key Establishment—Security Models, Protocols, and Usage*. PhD thesis, University of Waterloo, Department of Combinatorics and Optimization, 2008. 34

[46] Shengbao Wang and Zhenfu Cao. Escrow-free certificate-based authenticated key agreement protocol from pairings. *Wuhan University Journal of Natural Sciences*, 12(1):63–66, 2007. 24

[47] Shengbao Wang, Zhenfu Cao, and Haiyong Bao. Security of an efficient ID-based authenticated key agreement protocol from pairings. In *Parallel and Distributed Processing and Applications - ISPA Workshops 2005*, volume 3759 of *Lecture Notes in Computer Science*, pages 342–349. Springer Berlin / Heidelberg, 2005. 18, 39

[48] Shengbao Wang, Zhenfu Cao, Maurizio Adriano Strangio, and Lihua Wang. Cryptanalysis and improvement of an elliptic curve Diffie-Hellman key agreement protocol. Cryptology ePrint Archive, Report 2007/026, 2007. `http://eprint.iacr.org/`. 39

[49] Shengbao Wang, Zhenfu Cao, and Licheng Wang. Efficient certificateless authenticated key agreement protocol from pairings. *Wuhan University Journal of Natural Sciences*, 11(5):1278–1282, 2006. 18, 46

[50] Quan Yuan and Songping Li. A new efficient ID-based authenticated key agreement protocol. Cryptology ePrint Archive, Report 2005/309, 2005. `http://eprint.iacr.org/`. 39

[51] Dae Hyun Yum and Pil Joong Lee. Identity-based cryptography in public key management. In *Public Key Infrastructure*, volume 3093 of *Lecture Notes in Computer Science*, pages 71–84. Springer Berlin / Heidelberg, 2004. 12