Desired Features and Design Methodologies of Secure Authenticated Key Exchange Protocols in the Public-Key Infrastructure Setting

by

Hao-Hsien Bobby Wang

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2004

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand my thesis will be made electronically available to the public.

Abstract

The importance of an authenticated key exchange (AKE) protocol has long been known in the field of cryptography. Two of the questions still being asked today are (1) what properties or features does a secure AKE protocol possess, and (2) How does one, in a step by step fashion, create a secure AKE protocol? This thesis aims to answer these two questions.

The thesis contains two parts: one is a survey of previous works on the desired features of the Station-to-Station (STS) protocol, and the other is a study of a previously proposed design methodology in designing secure AKE protocols, as well as contributing an original idea of such methodologies. Descriptions and comparisons of the two design methodologies are included.

The thesis surveys the literature and conducts a case study of the STS protocol, analyzes various attacks on STS through some known attacks to it, and extracts the desired properties and features of a secure AKE protocol via the case study. This part of the thesis does not propose any new result, but summarizes a complete list of issues one should take consideration of while designing an AKE protocol. We also show that at the end of this part, a secure version of STS which possesses the desired features of an AKE protocol.

The other major part of the thesis surveys one design methodology of creating a secure AKE protocol by Bellare, Canetti, and Krawczyk; it is based on having a secure key exchange protocol then adding (mutual) authentication to it. The thesis then proposes another original design methodology; it starts with a secure mutual authentication protocol, then adds the secure key exchange feature without modifying overheads and number of flows of the original mutual authentication protocol. We show in this part the "secure" AKE protocol developed through these two design approaches is identical to the secure version of STS described in the other part, and thus possesses the desired features of a secure AKE protocol. We also give a proof of security of the secure AKE protocol developed under our design methodology.

Key words: authentication, authenticated key exchange, Diffie-Hellman, Station-to-Station protocol, STS.

Acknowledgments

Throughout my education, I have never felt so excited about studying cryptography. Not only the hackers' stories interest me, but the realization of the level of importance this field plays in the modern society. Throughout the entire process of completing this thesis, I have learned not only the research skills and original ways of thinking, but also a deeper understanding to my personal values and beliefs. I feel lucky that before I enter the "real world", I have the chance to study and produce work in the "ideal setting". If there is any value to this thesis, it is not just me, but also a lot of people's help and support.

I wish to thank my supervisor Doug Stinson for his unconditional help in knowledge, information, suggestions, and motivation. His kindness and generousness not only rescued my interest towards study, but also gave me once more my passion and confidence towards cryptography. It was such a pleasure working underneath you; I wish I could always be in contact with you and know you are living well (you can start by having more vacation days).

I wish to thank my mother for her very hard work for supporting her only son to study for many years in Canada. I am aware that I do not come from a wealthy family, and studying abroad from home may cost someone as much as debt and many years of bread and instant noodles for meals. I feel both lucky and sad that I am not the one in the above situation. No words in any language nor material can return your love and sacrifice for me. I could only say "I thank you, and I love you forever, mom."

I thank all my other family members and friends for your support and caring (especially during my down times). Special thanks to Eddie, one of my best friends since first year of undergraduate studies, for walking through such a tough process with me (we almost worked on the same topic!). Special thanks to Wesley Suen, Michael Yang, and Luis Serrano, for staying beside me when I am in need. Special thanks to Lydia, for she has taught me how to sense, feel, and love, and showed me the non-geeky side of the world. Special thanks to all my Mandarin Youth Fellowship friends and advisors, for giving me support, suggestions, and a lot of love. Special thanks to all the CACR people; you made me realize I am so not alone doing cryptography. Thanks to Angela, for occupying a lot of my heart and mind in the final stages of the thesis production. Thanks to the Huang's family for not letting me worry about where to live throughout the two years. Thanks to the most two famous persons in cryptography, Alice and Bob, for letting me use their names so many times without charging me copy-right fee.

Finally, and most importantly, unlimited thanks and glory to God, for otherwise everything up to now would not happen. Thank you.

Dedication

This thesis is dedicated to the forever righteous and loving God, my mother, and my other family members.

Contents

List of Figures

Chapter 1

Introduction

In the world of secret-key cryptography, a secure communication between two parties A and B can be thought of sending messages in an encrypted form using a *secret* commonly known to only A and B (so no one else should know the secret). This secret is referred to as a *key*, or in this case, a *secret key*. This simple technique of encryption using a secret key can be dated back to ancient times. In the old times, the two parties A and B agreed on a common key by meeting in person and coming up with the secret. However, in today's world of computing, secure communications are often needed between parties, even though they may never have seen each other before (and they may live on opposite sides of the planet). Suppose two people, Alice (A) and Bob (B), want to talk to each other securely but they have never met each other in person. The immediate problems in conducting a secure communication will be: 1) how does A know that she is really talking to B, and 2) how does A establish a key *sk* with B without seeing B in person? With the introduction of a trusted third party, certificates, and public key cryptography, the two problems seem to be solved easily. However, due to the much heavier computation of public key techniques than secret key ones, the practical solution of secure communication is to establish a common secret key *sk* using public key techniques, then encrypt messages using secret key cryptography with *sk*. Therefore, the above two problems still need to be solved. The first problem is usually referred to as the problem of *authentication* and the second as *key exchange*. It is often desirable that the solution should be a protocol achieving both authentication and key exchange at the same time. Such a protocol is called an *authenticated key exchange* (AKE) protocol.

A lot of research in key exchange has taken place in order to establish secure AKE protocols, discuss and debate desired features of AKE protocols, prove

the protocols are indeed secure, and ensure that the models resemble real life situation as a foundation of proof of security. In the traditional Public-Key-Infrustructure (PKI) setting, there are protocols such as Lim-Lee ([LL95]), the United Model Protocol ([IEEE00]), MQV ([MQV95, LMQSV03]), STS ([DOW92]), Yacobi ([Yac91]), Ateniese-Steiner-Tsudik ([AST98, AST00]), Oakley ([Orm98]), SKEME ([Kra96]), IKE ([HC98]), etc. One example of an industrial standard for authenticated key exchange is ISO/IEC 11770-3. Since 1995, research articles such as [BCK98, BR93A, Sho99] have developed different (yet somewhat similar) models and given proofs of security for different protocols. There are also survey articles (e.g., [BWJM97] and [BW99]) on key exchange protocols. [BW99] also discusses the design methodology and some goals or desired properties of a secure AKE protocol. Another approach to authenticated key exchange, namely ID- or password-based AKE protocols, has been under development since its introduction in the year 2000. For example, Katz, Ostrovsky, and Yung (in [KOY02]) have researched, in ID-based setting, secure AKE protocols (the KOY protocol), proofs of security and the corresponding model, and some desired features an AKE protocol should have (such as forward secrecy).

This thesis is a survey of the desired properties and design methodologies of authenticated key exchange protocols in the PKI setting. Unlike in [BW99, BWJM97], we specifically focus on the desired properties (or goals) and design methodologies and gives more details and analysis rather than just giving a description of the surveyed items. The motivations for doing so are as follows:

1. The desired features, or the goals of a secure AKE protocol are really the driving factor of a security model (which thus impacts the proof of security of a protocol). Without proper inspection of the desired features, a security model could be too strict (giving too much power to the adversary or requiring something that is not important) or too loose (missing essential or critical features/goals in the security model). If a model is too strict, then a "secure" protocol may be hard to design in practice; if a model is too loose, then a protocol designed under the model is simply not secure enough in the real world.

2. It's been noticed that many existing protocols, whether provably secure or not, are originally designed almost out of "thin air". That is, there is no standard process of creating an AKE protocol that can be proven secure. This ad-hoc approach to designing protocols may involve repetitively creating identical security models and giving proof of security. Having a design methodology that guarantees outcome protocols to be secure can simplify the above work and may re-use existing work for new development.

3. Although ID-based cryptography seems more convenient and more closely resembles the current network setting, the majority of applications (such as granting permission of access, electronic banking, electronic voting, etc.) are still being operated under the traditional PKI setting. Before the next working infrastructure replaces the current PKI, work on AKE protocols in the PKI setting can help people immediately.

The desired features of an authenticated key exchange protocol are demonstrated through a case study of the Station-to-Station (STS) protocol. Since the first introduction of STS in 1992 ([DOW92]), flaws have been found (also in other protocols, too) and fixes have been proposed, thus creating several variations of STS. In 1998, [BCK98] investigated a modular approach to the methodology of creating authenticated key exchange protocols and suggested and proved security of a variation of the STS protocol, generally referred to as the BCK protocol (see later chapters); this brings up the discussion of different methodologies of designing a secure AKE protocol. Other than the ad-hoc approach and Bellare, Canetti, and Krawczyk's modular approach, this thesis further discusses a third approach based on secure mutual authentication protocols and the security of Diffie-Hellman key exchange.

The organization of the thesis is as follows. Chapter 2 very briefly introduces the general background, common goals, and tools of cryptography. Chapter 3 gives the mathematical background essential to this thesis. For readers who are familiar with commutative groups, $\mathbb{Z}_p^*$ groups for p being a prime, elliptic curves, the discrete log problem (DLP), and the Diffie-Hellman problem (DH, including the computational Diffie-Hellman problem, CDH, and the decision Diffie-Hellman problem, DDH) can skip this section. Chapter 4 describes the formal models used to prove security of protocols. The chapter surveys three most relevant models: the Bellare-Rogaway model of the symmetric setting, the Blake-Wilson-Menezes model of the asymmetric setting, and the Bellare-Canetti-Krawczyk modularization model. The Bellare-Rogaway model, although not as relevant to this thesis, is described to explain concepts and terminologies. The other two models are explained according to their description in the original article. Chapter 5 discusses the desired features of authenticated key exchange protocols through a case study of the STS protocol. Some further background information is introduced, including key establishment protocols (key transport versus key exchange protocols), authentication and mutual authentication. A list of desired features of a secure AKE protocol is compiled, and some other examples of protocols, proofs of security, and discussions on existing standards are given. Chapter 6 discusses the different design methodologies of creating a secure AKE protocol. The ad-hoc approach (or the Bellare-Rogaway

Model Compliance approach), the modular approach in [BCK98], and Stinson's progressive approach are described and compared. The last chapter presents conclusion, explicit statements of our contribution, and suggestions of possible future work.

Chapter 2

Background and Terminology of Cryptography

2.1 Goals of Cryptography

The goal of an encryption scheme is to achieve *confidentiality* or *secrecy*. That is, the data sent from Alice to Bob cannot be understood by an unauthorized party. In the field of cryptography, there are many other goals for secure communication. This section gives some "hand-waving" definitions of goals of cryptography for readers with no prior knowledge to cryptography. The formal definitions of some of the goals crucial to this thesis are associated with different cryptographic models and will be explained in later sections. For readers who already have a good sense of what each goal means, this section can be skipped. Below is a list of the goals (these definitions are greatly inspired by and inherited from [StiA]):

- **Confidentiality:** confidentiality means that the data transferred from one party to another cannot be understood by an unauthorized party.
- **Data Integrity:** data integrity means that the data transferred cannot be modified by an unauthorized party without being detected.
- **Entity Authentication or Identification:** this means the verification of a particular entity (I am who I claim I am). In general, the term *authentication* will mean refer to entity authentication.
- **Data Origin Authentication:** this goal ensures that the data transmitted is from a particular source.

- **Mutual Authentication:** the two parties (say A and B) in a communication are confident that: 1) A is sure that she is talking to B, and B is indeed communicating with her, and 2) B is sure that he is talking to A, and A is indeed talking to him. Note that mutual authentication is not just authentication both ways, but it also means that the intended participants "match".

2.2 Cryptographic Tools

Throughout the development of cryptography, many tools or "modules" have been established to serve one or more goals of cryptography. Below is a list of some cryptographic tools associated with this thesis. Unlike the previous section, formal definitions of cryptographic tools are given in this section, as well as a brief explanation of what the tools are. One cryptographic tool this thesis does not mention here is the key establishment protocol. It is highly related to the focus of this thesis, and is left until later sections.

- **encryption schemes ([StiB]):** encryption is used to achieve confidentiality. Examples of encryption schemes are RC4, RC5, DES, 3DES, RSA, and AES. Formally speaking, an encryption scheme is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

  1. $\mathcal{P}$ is a finite set of possible *plaintexts*.
  2. $\mathcal{C}$ is a finite set of possible *ciphertexts*.
  3. $\mathcal{K}$, the *key space*, is a finite set of all possible *keys*.
  4. For each $\mathrm{K} \in \mathcal{K}$, $\exists$ an *encryption rule* $e_{\mathrm{K}} : \mathcal{P} \rightarrow \mathcal{C}$, $e_{\mathrm{K}} \in \mathcal{E}$ and a corresponding *decryption rule* $d_{\mathrm{K}} : \mathcal{C} \rightarrow \mathcal{P}$, $d_{\mathrm{K}} \in \mathcal{D}$ such that for all $x \in \mathcal{P}$, $d_{\mathrm{K}}(e_{\mathrm{K}}(x)) = x$.

- **signature schemes([StiB]):** a signature scheme can be thought of as someone (say A) signing a contract or a letter, so whoever receives this letter knows that it was signed by A and no one else. A signature scheme is used to achieve data integrity and data origin authentication. Formally, an signature scheme is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ where the following conditions hold:

  1. $\mathcal{P}$ is a finite set of all possible *messages*.
  2. $\mathcal{A}$ is a finite set of possible *signatures*.
  3. $\mathcal{K}$ is a finite set of all possible *keys*.

4. For each $\mathrm{K} \in \mathcal{K}$, $\exists$ a *signing algorithm* $\mathsf{sig}_\mathrm{K} : \mathcal{P} \to \mathcal{A}$, $\mathsf{sig}_\mathrm{K} \in \mathcal{S}$ and a corresponding *verification algorithm* $\mathsf{ver}_\mathrm{K} : (\mathcal{P} \times \mathcal{A}) \to \{true, false\}$, $\mathsf{ver}_\mathrm{K} \in \mathcal{V}$ such that for all $x \in \mathcal{P}$ and for all $y \in \mathcal{A}$,

$$\mathsf{ver}_\mathrm{K}(x, y) = \begin{cases} true & \text{if } y = \mathsf{sig}_\mathrm{K}(x) \\ false & \text{if } y \neq \mathsf{sig}_\mathrm{K}(x) \end{cases}$$

- **Hash Functions([StiB]):** A hash function is a function $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$, $\mathcal{Y}$ are sets of texts, $\mathcal{Y}$ is finite, $\mathcal{X}$ may or may not be finite, and the size or cardinality of $\mathcal{Y}$ is much smaller than the size of $\mathcal{X}$. Hash functions can be used to construct short fingerprints of some data. When a transferred message m is altered, then its hash value $h(m)$ will no longer be valid. Hash functions are often used for message digests or to provide data integrity. When a hash function takes a key as part of its input, then such a hash function is called a *keyed hash function* and can be used as a message authentication code (see below). Some famous examples of hash functions are SHA and SHA-1.

- **Message Authentication Codes(MACs)([StiB]):** A MAC is used to ensure that the message transmitted has not been modified (data integrity). Even though MACs serve similar purposes as signature schemes, the security requirements are different. An example of a MAC is HMAC, a standard proposed by NIST based on the SHA-1 algorithm.

2.3 Levels of Security

There are two types of measurement of security (in the example of encryption ciphers, we can use measurement of secrecy instead): **unconditional security** and **computational security**. In simple words, unconditional security means that the probability of an adversary achieving his goal (it can be decrypting a ciphertext, or forging a valid signature given a particular message, etc. – in any case it would be forging some string m' to be another string m) is no greater than "guessing" what m should be. Computational security, however, is when the adversary cannot achieve his goal if his computing power and time constraints are limited. Given enough computing power and time, an adversary can always break a computationally secure system. The point here is to clarify that public-key cryptosystems are "provably secure" only based on the underlying computationally hard problems, but not unconditionally secure. Formally speaking,

Definition: A cryptosystem is unconditionally secure if $\Pr[x|y] = \Pr[x]$ for all

$x \in \mathcal{P}, y \in \mathcal{C}$. That is, the a posteriori probability that after observing ciphertext y, the plaintext is x, is identical to the a priori probability that the plaintext is x.

That is, the adversary cannot do better than guessing what the plaintext x is, even when he has seen the ciphertext y and regardless how much time or computing power he has. One example of unconditionally secure cryptosystems is the One-Time Pad. For description and proof that it is unconditionally secure, the readers are referred to [StiB].

Some cryptosystems, such as the RSA encryption scheme, relies on some "difficult" mathematical problems (RSA relies on the difficulty of factoring products of two or more large primes). It is easy to see that given limited computing power but enough time, a composite number can always be factored. Such cryptosystems are said to be only computationally secure.

The examples of One-Time Pad and RSA demonstrate the difference between unconditional security and computational security. In any public-key cryptosystems, unconditional security is never possible; the term "secure" would really mean computationally secure. In later chapters, the term "provable security" of an AKE protocol does not mean "unconditional security" since AKE protocols discussed in this thesis are all in the public-key setting. Rather, the term "provable security" would mean the security of a protocol is at the same level of security as the underlying mathematical problems.

Chapter 3

Mathematical Background

In any public-key cryptosystem, part of the user A's "secret" is always known to the public; given enough time and computing power, one can always retrieve the value of A's secret. Therefore, "security" in a public-key setting always refers to *computational security*, and such security is based on very difficult mathematical problems, such as the integer factorization problem or the discrete log problem (DLP). The integer factorization problem is described as follows: *given an integer n, find the unique factorization $n = p_1^{e_1} \times p_2^{e_2} \times ... \times p_k^{e_k}$, where $p_1, p_2, ..., p_k$ are distinct primes and $e_1, e_2, ..., e_k$ are non-negative integers.* The discrete log problem is described as: *Given a commutative group G, a generator α of a subgroup $\langle\alpha\rangle \subseteq G$, and an element $y \in \langle\alpha\rangle$, find the smallest non-negative integer x such that $y = \alpha^x$.*

3.1 Commutative Groups

A group is an ordered pair $(G, \cdot)$, where G is a set and $\cdot$ is a binary operation on G. If one element $a \in G$ operates another element b, we write $a \cdot b$. G is called a group if it satisfies the following four properties:

1. For all $a, b \in G,\ a \cdot b \in G$.
2. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in G$.
3. There exists an element $e \in G$ such that, for all element a belong to G, $a \cdot e = a$.

4. For all $a \in G$, there exists another element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

A commutative group is a group that satisfies the fifth property:

5. $a \cdot b = b \cdot a$ for all $a, b \in G$

As an example, the set of integers is a commutative group under multiplication since for all integers a, b, $a \cdot b = b \cdot a$. However, the set of 2×2 matrices, each entry being integers, under the operation of matrix multiplication, is not.

3.2 The Group of $\mathbb{Z}_p^*$ and Elliptic Curves

Generally speaking, many difficult mathematical problems used in cryptosystems are defined in some group. A cryptosystem can be implemented in different ways just by changing the underlying group from one to another. Regarding the purpose of this thesis, two relevant commutative groups, namely $\mathbb{Z}_p^*$ and Elliptic Curves (EC) are introduced. Protocols such as MQV are originally proposed with underlying group EC, but can be described as having $\mathbb{Z}_p^*$ as its underlying group. For simplicity and consistency, protocols described in this thesis can all be, and are assumed to be, based in the group of $\mathbb{Z}_p^*$.

3.2.1 The group of $\mathbb{Z}_p^*$

By the formal definition of a group, the group $\mathbb{Z}_p^*$ should really be written as $(\mathbb{Z}_p^*, \cdot)$. The set $\mathbb{Z}_p^*$ is defined as $\mathbb{Z}_p^* = \mathbb{Z}_p \backslash \{0\} = \{1, 2, \ldots, p-1\}$, and the binary operation $\cdot$ is multiplication modulo p, where p is a prime number. For an element $\alpha \in \mathbb{Z}_p^*$, the operation $\alpha \cdot \alpha$ can be represented as α^2, and the number 2 is called the *exponent* of the expression α^2. Similarly, if there are k copies of α (that is, $\underbrace{\alpha \cdot \alpha \cdot \ldots \cdot \alpha}_{k \text{ copies}}$), then it can be expressed as α^k, and the exponent of α^k is k.

In the field of cryptography, there are many relevant theorems and concepts associated with this group (such as the Euclidean algorithm, the Chinese Remainder Theorem, quadratic residues, the Lagrange Theorem, etc.). The readers can refer to any undergraduate-level group theory textbook for more information.

3.2.2 Elliptic Curves

An elliptic curve is denoted by $(E, \oplus)$, where E is a special point ϑ (called the point at infinity) and a set of points $\{(x, y)\}$ which are solutions of a bivariate cubic equation over a field $\mathbb{F}$ (for more information on fields, please also refer to an undergraduate textbook). The binary operation $\oplus$ of an elliptic curve is called point addition. The bivariate cubic equations are of the form:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1, a_2, \ldots, a_6 \in \mathbb{F}$.

If the characteristic of $\mathbb{F}$ is not 2 or 3, then the above equation can be written as

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{F}$.

If the field $\mathbb{F}$ has characteristic 2, then the elliptic curve equation can be simplified to the form

$$y^2 + xy = x^3 + ax + b$$

where $a, b \in \mathbb{F} = \mathbb{Z}_2 = \{0, 1\}$.

Elliptic Curve Point Addition

The operation on elliptic curve points follows the rules below:

1. For all points $P \in$ E, $\vartheta \oplus P = P \oplus \vartheta = P$, where ϑ is the point at infinity (in other words, ϑ is the identity of the group).
2. Let $P = (x_1, y_1) \neq \vartheta,\ x_1, y_1 \in \mathbb{F}$. The inverse of $P = (x_1, -y_1)$ is denoted by $-P$.
3. Let P, Q be two points in the set E. The operation $P - Q$ means $P \oplus (-Q)$.
4. $P \oplus P$ can be denoted as $2P$. Similarly, if k copies of P are added, the result is denoted as kP.

Let the points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, where $x_1, y_1, x_2, y_2 \in \mathbb{F}$, and $P, Q \neq \vartheta$, $Q \neq -P$.

1. If $\mathbb{F}$ has characteristic $\neq 2$ or 3:
   Let $P + Q = (x_3, y_3)$. The inverse of P is $-P = (x_1, -y_1)$.
   If $P \neq Q$, then

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

   where $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$.
   If $P = Q$, then

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

   where $\lambda = \frac{3x_1^2 + a}{2y_1}$.

2. If $\mathbb{F}$ has characteristic 2, then let $P + Q = (x_3, y_3)$. The inverse of P is $(x_1, x_1 + y_1)$.
   If $P \neq Q$, then

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \end{aligned}$$

   where $\lambda = \frac{y_1 + y_2}{x_1 + x_2}$.
   If $P = Q$ then

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a \\ y_3 &= x_1^2 + (\lambda + 1)x_3 \end{aligned}$$

   where $\lambda = x_1 + \frac{y_1}{x_1}$.

3.3 The Discrete Logarithm Problem and the Diffie-Hellman Problems

As mentioned in previous sections, public-key cryptosystems can only be computationally secure and not unconditionally secure. In this section, we will introduce three hard problems relevant to the rest of the thesis that are based on the groups described in the previous section. These are the underlying problems used to secure the public-key cryptosystems used in practice today; once these problems can be easily solved, the current public-key cryptosystems are pretty much broken.

3.3.1 The Discrete Logarithm Problem (DLP)

The formal description of DLP is as follows. Given the description of a finite field $\mathbb{F}_q$, q being the *order* of the field, a generator $\alpha \in \mathbb{F}_q^*$, and another element h in the cyclic subgroup generated by the generator α, $\langle\alpha\rangle$, find the exponent k such that $h = \alpha^k$.

Note that the previous two groups, $\mathbb{Z}_p^*$ and elliptic curves, both satisfy the conditions for inputs of DLP.

Specifically, the $\mathbb{Z}_p^*$ version of DLP would be as follows: given the finite field $\mathbb{Z}_p^*$, where p is a prime number, a generator $\alpha \in \mathbb{Z}_p^*$, and an element $h \in \langle\alpha\rangle$, find the exponent k such that $h = \alpha^k$.

Unlike the similar notation of general fields and $\mathbb{Z}_p^*$, the elliptic curve version of DLP may be confusing. The description of the problem is as follows: given an elliptic curve E, a generator $\alpha = (x_\alpha, y_\alpha) \in$ E, and an element $H \in \langle\alpha\rangle = \{i\alpha : i \in \mathbb{Z}\}$, find the exponent k such that $H = k\alpha$.

In general, when discussing the security of a public-key cryptosystem, one of the assumptions is that the *discrete logarithm assumption* must hold, for otherwise the cryptosystem will be insecure. In simple words, the discrete logarithm assumption (DL assumption) states that the DL problem is intractable given reasonable computing power and time.

3.3.2 The Computational and Decision Diffie-Hellman Problems (CDH and DDH)

The Computational and Decisional Diffie-Hellman problems are a consequence of the DLP and Diffie-Hellman key exchange. The two problems are usually thought to be representing two different levels of security. In the following, the two problems are stated and will be compared. The readers should keep in mind that the notations will be in generic form. Therefore, the expression α^k in $\mathbb{Z}_p^*$ is written as is, but in elliptic curve notation it is written as $k\alpha$. In both cases, the term *exponent* means k.

Before describing CDH and DDH, we will first describe the Diffie-Hellman key exchange.

The Diffie-Hellman key exchange is described as follows: There are two participants A and B involved in this process. Without loss of generality, suppose A is the initiator (who sends the first message to the other participant) and B is the

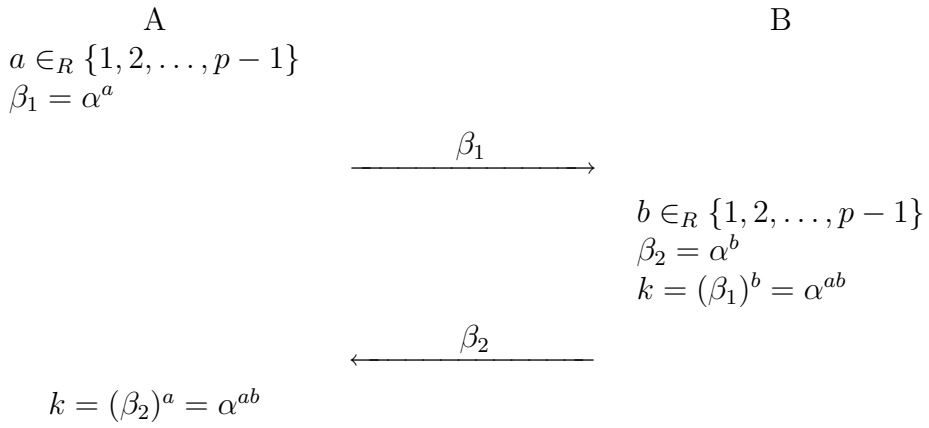

Figure 3.1: The Diffie-Hellman Key Exchange

responder (who receives the first message and responds to it). Let p be a prime and α a generator of $\mathbb{Z}_p^*$, the first step 1) is: A uniformly randomly selects an exponent $a \in \{1, 2, \ldots, p-1\}$ and sends $\beta_1 = \alpha^a$ to B. 2) Upon receiving β_1, B uniformly randomly selects another exponent $b \in \{1, 2, \ldots, p-1\}$ and computs the key $k = \beta_1^b = \alpha^{ab}$. B then sends the value $\beta_2 = \alpha^b$ to A. 3) Upon receiving β_2, A computes $k = \beta_2^a = \alpha^{ab}$. Therefore both A and B have computed the same key $k = \alpha^{ab}$. This completes the Diffie-Hellman key exchange.

The computational Diffie-Hellman problem (CDH) is stated as follows: given a description of a finite field $\mathbb{F}_q$, a generator $\alpha \in \mathbb{F}_q^*$, two elements $\beta_1 = \alpha^a$ and $\beta_2 = \alpha^b$, where $a, b \in \mathbb{Z}$ and $0 < a, b < q$, find α^{ab}.

The decision Diffie-Hellman problem is the following: given a description of a finite field $\mathbb{F}_q$, a generator $\alpha \in \mathbb{F}_q^*$, three elements $\beta_1 = \alpha^a$, $\beta_2 = \alpha^b$, and $\beta_3 \in \langle\alpha\rangle$, output *'yes'* if $\beta_3 = \alpha^{ab}$, output *'no'* otherwise.

It is easily seen that DDH polynomial-time-reduces to CDH, and CDH can be reduced in polynomial time to DLP. Therefore, it is generally believed that the *DDH assumption* represents a lower standard of security than the CDH and DL assumptions.

Note: Similar to the DL assumption, CDH assumption means that the computational Diffie-Hellman problem is intractable given reasonable computing power and time. The DDH assumption means that the decisional Diffie-Hellman problem is intractable given reasonable computing power and time.

Chapter 4

Formal Models of Authenticated Key Exchange

Key exchange, as explained before, is the process of establishing a shared secret between two parties for subsequent cryptographic usage. Authenticated key exchange, is the process of key exchange plus each entity identifying him/herself to another. In analyzing security of authenticated key exchange (AKE) protocols, one of the core task is to answer the questions "What is a secure AKE protocol?" and "How do I show that a particular AKE protocol is secure?"

4.1 Notion of Security

For a long time in the development of cryptography, the notion of security has been approached in an ad-hoc fashion (or the attack-response approach): a cryptosystem is developed, and an attack comes up, a new cryptosystem is developed to fix the problem, and another attack comes up, etc. With such an approach, no guarantees at all are given to a particular cryptosystem; the best statement that can be made with this approach is that a cryptosystem has not been broken yet and the system has not been fully analyzed. Therefore, a better assurance of security is needed.

The two most commonly used tools for formally analyzing the security of a protocol are provable security and formal methods. In particular, provable security tends to formalize the communication model, the attacker's capabilities, and to give a proof based on definition of "security" associated with the model. On the other hand, formal methods tend to analyze a protocol in a systematic way to find flaws in a protocol. The immediate implication is that the usages of the two

methods can be complementary to each other, but formal methods, although they may spot flaws in a protocol very effectively, may not guarantee the security of a system. Therefore, in terms of demonstrating the security of a cryptosystem, provable security is usually preferred. In this thesis, we will analyze protocols in a provable-security fashion.

A formal model treatment of authenticated key exchange protocols not only gives a tool for analyzing the protocol, but suggests an infrastructure for proving security, as well as a formal definition of security. Therefore, before discussing the desired features of an AKE protocol, it is helpful to survey previous work on formal models.

The rest of this section is organized as follows: the first subsection will include informal definition of secure authenticated key exchange; the rest of this section describes three formal models used to treat AKE protocols. The first model is by Bellare and Rogaway in the symmetric, or private-key setting. The second model is the Blake-Wilson-Menezes model in the asymmetric, or public-key setting. The third model is by Bellare, Canetti, and Krawczyk in the asymmetric setting. The notion of security and assumptions used in the rest of this paper will be mostly based on the Blake-Wilson-Menezes model. However, it will be easier to explain the link between real-life situations and the formalization by firstly introducing the Bellare and Rogaway model. The third formalization, the Bellare-Canetti-Krawczyk modularization model, is especially relevant to the design methodology of authenticated key exchange protocols that we study in later chapters. Note that other than the three models described in this section, there is only another one by Shoup ([Sho99]). However, most of the material in [Sho99] is not directly relevant to the issues we are considering. Therefore, we will not discuss Shoup's model in this thesis.

4.2 Informal Definition of Security

The term authenticated key exchange did not appear in the literature at the beginning of cryptography. For a long period of time, people either did not focus on this issue, or they stressed authentication and key exchange as separated issues. It is now believed that in terms of designing a secure system, these two issues should really be considered as one. In particular, [Mao] suggests that if authentication is separated from key exchange, some work will be redundant and the overall security may not be easy to achieve.

After surveying several articles related to this topic, two (informal) attributes are thought to be essential to authenticated key exchange:

1. **Secrecy of the session key.** If the adversary can learn any information about the session key developed by two parties A and B, then obviously any subsequent communication using the session key will be vulnerable to attack. Therefore, the adversary should not learn anything about the exchanged session key.

2. **Mutual authentication** of the two entities in the protocol run. If A wishes to establish a key with B via an authenticated key exchange protocol, not only will she want to be sure that she has exchanged the key with B at the end of the protocol run, but she also wants B to be sure that he has exchanged a session key with her, too.

These two principles seem to be very simple, but their formalization is somewhat difficult. The following subsections on formal models describe different attempts to capture the real security of an AKE protocol formally. Although no formalism can capture every single real-life scenario, these formalizations still model the essential parts of communication.

4.3 General Background of Provable Security

The need for provable security was firstly noticed during the 1980s. By 1985, provable security had been achieved in many areas including by Yao [Yao82] and Blum and Micali [BM84] for pseudo-random number generators; Goldwasser and Micali [GM84] for probabilistic encryption; and Goldwasser, Micali, and Rivest [GMR88] for digital signatures. However, not until 1992 were authentication and authenticated key exchanges looked at in a systematic way. Diffie, van Oorchot, and Wiener [DOW92] introduced the attributes desirable in an AKE protocol and showed security by demonstrating attacks on the protocol with the removal of a certain feature. In 1993, [BR93B] first put different desirable attributes into a *model* covering not just the desired attributes of a protocol, but also the communication and adversarial models. Since then, more research had been carried out in this field and it became clear which features of a system should be looked at in order to analyze the security.

In general, the analysis of a protocol must take the following items into account:

- The Communication Model: the modelling of analysis should resemble real life computing and communication. Issues regarding this item include the distributed computing environment; public-key versus secret-key setting; whether (in the public-key setting) certificates are present and how they are distributed to users; number of users; single or multiple sessions, etc.

- The Attack Model: this simulates the capability of the adversary and should reflect the real-life situation as much as possible. If the adversary's ability is over-estimated in the analysis, then it may be unreasonably difficult to design a secure protocol to comply with the security requirements. If, on the other hand, the attack model proposes weaker abilities of the adversary than a real life one, then the protocol designed according to this attack model simply is not secure. The common abilities of an adversary include eavesdropping, delaying or dropping messages, injecting and modifying messages. Notice that the attack model may vary slightly depending on the goals of the adversary and the protocol. There is also a difference between passive and active attacks.

- The goal of the cryptosystem and the goal of the adversary: depending on the purpose of the cryptosystem, the goal of the adversary may be different. These two goals contribute to the definition of a secure cryptosystem. If any of the goals are inappropriate, the resulting cryptosystem may be insecure in real life, and may damage other parts of a larger system that uses this particular cryptosystem.

- Definition of security.

In order to prove the security of a protocol, [BWM97] combines the above features along with the description of protocol. It suggests five steps in proving security of a protocol:

1. Specification of the model (including a formalization of the communication model and the attack model)

2. Definition of goals: including the goals of the protocol and the adversarial goals. This further clarifies the definition of security.

3. Statement of assumptions: assumptions about the underlying mathematical problems, certain constraints of user interactions, etc.

4. Description of the Protocol

5. Proof that the protocol meets its goals within the assumed model.

The five steps give a very clear way to prove security. In different approaches and design methodologies, the steps may be organized differently or the steps may be mixed up. However the above five steps are essential to consider when proving the security of an AKE protocol.

4.4 The Bellare-Rogaway Model of Authenticated Key Exchange

[BR93B] is the first treatment that formalizes authenticated key exchange protocols. A unified model is given that captures not just the way of communication, but also the adversary's ability in the communication model. Also, many real-life constraints are included as part of the model, or treated as assumptions. Associated with the formal model is a formal definition, using attributes derived from the model, of authenticated key exchange.

4.4.1 Connection: Real Life to Formalization

Before describing the model, it is easier to explain the connections between real life scenarios and their formalized forms. By doing so, readers can get a better understanding why a model is constructed in a certain way, and may relate more closely to real-life situations when understanding the model. Although one may be concerned that not all the subsequent research in this field follows the model of [BR93B], in all the subsequent work the concept of transferring a real-life scenario into a formalized form is very similar, if not the same, to the model proposed by Bellare and Rogaway. Therefore, it is very important to mention Bellare and Rogaway's contribution converting real events into a formalization.

This subsection will start by listing real life situations and then explain in plain English the way and reason of modelling the situation into a certain formalism. The idea is to explain the communication model and then explain how and why it is transformed into the formalization of Bellare and Rogaway's model. The adversarial model, the goals of an authenticated key exchange protocol, etc., are discussed.

The Communication Model

The communication model used in Bellare and Rogaway's model is in the secret-key cryptography setting. That is, before people wish to communicate with one

another, they must have an a priori shared secret. Theoretically, the number of participants in the communication model can be infinite, although in practice this will never be the case. Users who wish to establish the pre-determined secret should use a perfect random number generator, although in practice it is usually a pseudo-random number generator good enough that we assume acts close to perfectly random. There is a protocol that is associated with the communication model. Specific to our purpose, a protocol is run between two users only, but the two users may run the protocol many times for key exchange. At the end of each protocol run, one side of the protocol (a participant) may *accept* or *reject* the other party's identity and the established key. For simplicity, each protocol run is called a *session*, and the key developed at the end of the protocol run is referred to as the *session key*. One session involves at least one message transfer from one party to another; this is called a *flow*. In a session, a user either receives a message from the previous flow and then sends its output to the other user; it initializes a session (if it is first flow of the protocol); or it receives the last message of the session and decides whether it accepts or not. Of course, in order to make such decisions, each user itself may have a certain rule to decide what to do next in the session and it may require a random number generator or probabilistic algorithm. Each user uses a computer that performs polynomially many operations only.

The Adversarial Model

The adversary is thought of as existing in environment such as the internet. Its capabilities include eavesdropping on messages; mixing up the order of sent messages; dropping one or more messages in the communication; injecting or modifying messages in the wire; impersonating a legitimate user; participating in the communication model itself as a legitimate user; or requiring one or more particular sessions to expose their session keys (the known session key attack). However, the adversary's computing power is also polynomially bounded.

The Goal of Authenticated Key Exchange

Defining the security of authenticated key exchange is difficult; it is hard to specify what requirements a protocol must and must not meet in order to be a secure AKE protocol. Bellare and Rogaway in [BR93B] gave their definition as follows:

A protocol Π is a *secure* authenticated key exchange protocol, then it must be 1) a mutually authenticated protocol, 2) if the adversary does not alter any messages (i.e. we have a passive attacker), then both parties of the protocol will accept (if

they are honest), and 3) the adversary should not know anything that enables him to retrieve the session key with a greater chance of success than guessing its value.

Consequently, their definition of mutual authentication is as the follows:

A protocol is called a secure mutual authentication protocol if for the polynomial-computing-powered adversary,

1. Assume all users keep their records of conversation. If there is a session between users i and j, and they are both aware of this session plus they think they are talking to each other in this session, both of the two users and they should both accept.

2. There should not be a case where some legitimate user i has a conversation record of some session π, i accepts, but there are no other legitimate users holding a corresponding matching record of conversation.

Formalization

We now explain how the real-life scenario is converted into formalization.

Modelling the execution of authenticated key exchange protocols can be thought of as an experiment with different components. The secret key setting, where every user shares a secret with another user to start with, is captured by a *key generator* $\mathcal{G}$. In real life, each user pair may determine the shared secret by themselves, and each pair's secret may be different. The formalization models this as $\mathcal{G}$ choosing the secret for each pair based on a random input given to $\mathcal{G}$ at the start of the experiment. Also, every pair of users get the same secret a. It may be due to the fact that Bellare and Rogaway originally developed the model for a potential possibility to distribute keys to multiple users in a user group sharing the same secret. The key generation process is that a central key generation algorithm starts with a random input and is also in charge of distributing the keys secretly, since all users are supposed to choose their secret keys using a perfect random number generator and agree upon the secret key in a secure manner. However, the identical secret for every pair seems unrealistic. This event is not only rare in real life, but it eliminates the possibility of modelling another real-life scenario: that if a person learns the secret of a pair of users i and j, then a secure system should not allow the adversary to be able to know the secret key between any other user and i or j. However this is not the case in this model.

All the users in the "experiment" are thought of as a set, denoted by I. For simplicity each user is identified by a natural number i. The user set I is finite. The

"sessions" between each user pair are captured by two instances of the protocol. For simplicity we will call these "sessional protocols". If i runs a session with j, then it is modelled that there are two sessional protocols $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, where $\Pi^s_{i,j}$ represents i's point of view of this session: i believes that it is talking to j for the sth time since the start of the experiment. More generally, s is the session ID i keeps to itself as a reference. Similarly, $\Pi^t_{j,i}$ represents that j believes that it is talking to i with session ID t. This type of modelling captures that the key exchange protocol is only between two users i and j, but each of them may have several sessions with the other user. Notice that the two sessional protocols' session identifier may or may not be the same. This is due to the adversary's ability of trying to pretend to be one entity and talking to the other. Also, between each pair i, j there may be an arbitrary but finite number of such sessional protocols.

Note: in the original paper the term "sessional protocol" does not exist. The author has created this term to distinguish a protocol Π, which possesses a set of rules, and each session of the protocol (which is denoted in the form $\Pi^s_{i,j}$).

Due to the modelling of a session, the messages transferred in a session are then captured as inputs to and outputs from a sessional protocol. The sessional protocol $\Pi^s_{i,j}$ takes an incoming message sent from j to i, perhaps performing some rules of calculation and decision making (this is captured by calling the protocol Π), then outputs a message sent back to j. The outputs of $\Pi^s_{i,j}$ not only include the outgoing message to the other party, but also its decision of whether to accept or reject as well as the agreed session key. This type of formalism also include the uncertainty that a party does not necessarily receive messages sent by the other party.

The adversary, with the ability of eavesdropping, mixing the order in which messages are delivered, requiring the session key from a particular session, participating as a legitimate user, dropping messages, injecting messages, impersonating a legitimate user (with polynomially bounded computing power) is modelled as a separate entity in the model. The adversary is said to be a polynomial time probabilistic algorithm (to capture that the adversary's computing power is limited). The adversary does not belong to the user set (although in reality the adversary may be a legitimate user itself). The adversary also has a secret key and all other information a legitimate user of the protocol may have. Such information is again distributed by a central algorithm such as $\mathcal{G}$. All its capabilities are captured by the whole model being at its control with a certain level of limitations. Since a session is captured by two sessional protocols $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, the input to the sessional protocol is controlled by the adversary (capturing message-injection, dropping, and mixing order of messages), and the output, except for the computed secret key, is known to the adversary (capturing eavesdropping). To capture known session key

attacks, another type of query, called reveal, is introduced. The query to a sessional protocol $\Pi^s_{i,j}$ returns the resulting session key if there is one, and it returns the null value otherwise. Together with the adversary having the same type of information as the rest of users, it allows the adversary to play in the protocol as a legitimate user as well.

4.4.2 The Model

After the transition from plain English description to this formalism capturing methodology, this subsection is the formal description of Bellare and Rogaway's model. The organization is somewhat different from the original paper, but the author believes that the way described in this thesis may be more clear. The description will start with the preliminary notations, terminologies, assumptions, and details to make the whole model well-defined. Then the model is described from the "big picture" down to details, where details may or may not be mentioned in the preliminary part.

Preliminaries

- The set of finite binary strings is denoted by $\{0,1\}^*$. The set of finite binary strings of length at most k is denoted as $\{0,1\}^k$. The set of infinite binary strings is denoted as $\{0,1\}^\infty$.
- The concatenation of two strings str_1 and str_2 is written as $str_1||str_2$.
- A function $\epsilon(k)$ is said to be *negligible* if for all $c > 0$, there exists a $k_c > 0$ such that $\epsilon(k) < k^{-c}$ for all $k > k_c$.
- **The security parameter:** it is often associated with every cryptosystem, and is denoted by k. In unary form, the security parameter is represented as 1^k.
- **The set of users:** as described before, it is denoted by I. Users are also referred to as the *participants* or the *players* of a protocol. In the two-user setting, the identity of the *sender* or the *initiator* of the protocol is usually denoted by A(or Alice), and identity of the (intended) partner or *responder* is denoted by B(or Bob). In a more general setting (i.e. two users or more), the identity of the sender is denoted by i and the identity of the intended partner denoted by j. Sometimes the notation i, j and A, B are interchanged

for simplicity of explanation. Each individual user's identity is represented by a finite binary string in $\{0,1\}^k$ of length at most k (k is the security parameter).

- **The Long-Lived Key (LL-Key):** also referred to as the *secret* of the sender or the *private key* of the sender. In the symmetric setting of the Bellare-Rogaway model, all users $i \in I$ receive the same LL-key, denoted by a. the secret a belongs to the set $\{0,1\}^*$. The LL-keys in Bellare and Rogaway's model are generated and distributed by the LL-Key Generator $\mathcal{G}$.

Running the Protocol

Running the protocol in this case means the execution of the "experiment" discussed above. Again, a protocol run in the usually sense is referred to as a "session". After describing the preliminary items, the formal description of the "experiment" is given as follows:

The model consists of a key exchange protocol Π, an adversary O, a LL-key generator $\mathcal{G}$, and a specified security parameter k. The protocol is an algorithm, running in polynomial time, that takes the input (i, j, a, κ, r) and gives the output (m, δ, α). i represents the identity of the sender, j represents the identity of the intended receiver, a represents the secret key of the sender, κ is a record of conversation, and r is a random input. $i, j \in I$, $a \in \{0,1\}^*$, $\kappa \in \{0,1\}^*$, $r \in \{0,1\}^\infty$. In the output, m is the "next message" to send out, δ is the decision of whether the sender accepts or not, and α is the "private output" given to the sender only. $m \in \{0,1\}^* \cup \{*\}$, where $*$ means the sender sends no message. $\delta \in \{\mathsf{A}, \mathsf{R}, *\}$, where A means *accept*, R means *reject*, and $*$ means undecided. $\alpha \in \{0,1\}^* \cup \{*\}$, where $*$ means "no change in internal state".

The LL-key generator $\mathcal{G}$ is a polynomial time algorithm which takes the input $(1^k, i, r_G)$ and output and delivers a private key $a_i = \mathcal{G}(1^k, i, r_G)$ to user i in a secure manner. In the input, 1^k is the security parameter, and r_G is a random input. [BR93B] assumes that the secret value $a_i = a_j$ for all $i, j \in I$. The LL-key generator also generates the key for the adversary, but it is specified to be $a_{\mathrm{O}} = \mathcal{G}(1^k, i, r_G) = \lambda$, where λ represents the empty string. In Bellare and Rogaway's model the value of each secret key is just a prefix of the random input string r_G, and the length of the secret key varies with each individual protocol specification.

Each sessional protocol is denoted by $\Pi^s_{i,j}$, where i, j, s is as described above. Each sessional protocol in the formalization is activated (or called) by the adversary for some input queries. Upon receiving queries, the sessional protocol itself calls the

protocol Π and gets the output returned by Π. After receiving the output (which may include private information updates, the sessional protocol updates its internal state as described by the output of Π, then returns to the adversary the appropriate information retrieved from Π. Each $\Pi^s_{i,j}$ is also associated with a unique record of conversation, denoted by $\kappa^s_{i,j}$, indicating message sender i's view of the protocol flow so far.

The adversary, denoted by O and sometimes referred to as Oscar, is a polynomial-time probabilistic machine that has access to infinite number of sessional protocols $\Pi^s_{i,j}$. Before the adversary starts running it takes the input parameters $(1^k, a_{\mathrm{O}}, r_{\mathrm{O}})$, where 1^k is the security parameter, a_{O} is the private key generated by the LL-key generator for the adversary, and $r_{\mathrm{O}} \in \{0,1\}^\infty$ is a random input. The adversary is capable of sending queries to each sessional protocol representing passive and active attacks. If the adversary is deterministic and faithfully delivering messages to the corresponding protocols, then such an adversary is called a *benign adversary.*

As described before, the adversary in the formal model has the control of what to send to each sessional oracles. This is done via *queries.* To capture the ability of the adversary, as well as to complete the modelling, the adversary is equipped with three types of queries. They are the send query, the reveal query, and the test query. Each type of query is explained in the following:

- **The Send Query:** this query takes the input (i, j, s, x), representing O sending message x to entity i, claiming it is from j in session ID s, and the query is answered by the sessional protocol $\Pi^s_{i,j}$.

  To respond, the sessional protocol $\Pi^s_{i,j}$ calls the protocol Π with input

  $$(1^k, i, j, a_i, \kappa^s_{i,j} || x, r^s_{i,j}),$$

  where $r^s_{i,j}$ is the random input given to protocol $\Pi^s_{i,j}$ at the beginning of a protocol run (this will be discussed later). Π returns (m, δ, α) to $\Pi^s_{i,j}$. Then $\Pi^s_{i,j}$ updates $\kappa^s_{i,j}$ to be $\kappa^s_{i,j}||x$, and returns (m, δ) to the adversary.

  Notice that the term send *query* did not appear in the original paper. However, it is more convenient to distinguish this original query type with the other two. Also, in later work on authenticated key exchange modelling, many researchers have used such a term, thus we name it send at this point.

- **The reveal Query:** this query enables the adversary to ask any sessional protocol to return its session key. To simplify the notation, if the sessional

protocol is $\Pi^s_{i,j}$, then the corresponding session key will be $sk^s_{i,j}$. The format of the reveal query is $(i, j, s, \textbf{reveal})$, and $\Pi^s_{i,j}$ returns $sk^s_{i,j}$ to adversary.

During the protocol run, if a reveal query is sent, then $\Pi^s_{i,j}$ is said to be *opened*. Otherwise, it is said to be *unopened*.

- **The test Query:** it is to test whether the adversary is successful in terms of attacking the key exchange protocol. More will be said about this type of query later.

Finally, running the protocol will mean the following:

1. The initialization phase: the random numbers $r_G \in \{0,1\}^\infty$ (for the LL-key generator), $r_{\mathrm{O}} \in \{0,1\}^\infty$ (for the adversary), and $r^s_{i,j} \in \{0,1\}^\infty$ (for each sessional protocol $\Pi^s_{i,j}$) are generated. The LL-key generator $\mathcal{G}$ then generates the secret keys $a_i = \mathcal{G}(1^k, i, r_G)$ and distributes them securely to each entity $i \in I$. $\mathcal{G}$ also generates the secret information a_{O} and sends it (securely) to O. All the records of conversation $\kappa^s_{i,j}$ are initialized to λ.

2. The adversary now issues Send and Reveal queries according to his own (probabilistic) rules.

3. After sending a certain number of queries, the adversary issues the Test query with input (i, j, s, Test) to a certain sessional protocol. The protocol "tosses a coin" and gets a random bit $b \in_R \{0,1\}$. If $b = 0$, $\Pi^s_{i,j}$ returns its session key $sk^s_{i,j}$. If not, then $\Pi^s_{i,j}$ returns a random string from the key space. The adversary receives the output string from the sessional oracle, and he guesses what the random bit b is. The adversary's success is then measured on the probability of him guessing correctly the bit b (the conditions specifying which sessional protocol can be sent a Test query is discussed later).

4.4.3 Definition of Security

The notion of a secure key exchange protocol in [BR93B] is explicitly defined using events of the protocol. In particular, [BR93B] first introduced the concepts of *matching conversation* and *indistinguishability* of session keys to denote security. The following subsections will discuss these concepts, the security of mutual authentication, and links to running the model.

Matching Conversation and Mutual Authentication

Recall that the informal definition of a mutual authentication protocol between participants A and Bis: *at the end of the protocol run, if* A*accepts, then she believes that her intended participant is* B*, and she believes that* B*'s intended participant is* A*, in this protocol run.* So, what does that mean in a formal model?

In [BR93B], Bellare and Rogaway assert that if A is sure that she is talking to B in a particular session, and B is also sure that he is talking to A in the same protocol; both of them accept, then there must be two sessional protocols $\Pi^s_{\mathrm{A,B}}$ and $\Pi^t_{\mathrm{B,A}}$ such that their record of conversation *match* each other's and they should both accept. In other word, they base the notion of mutual authentication on the distribution of records of conversation and decisions.

Suppose the record of conversation of a particular sessional protocol $\Pi^s_{i,j}$ is written in the following form:

$$\kappa^s_{i,j} = (t_1, \alpha_1, \beta_1)||(t_2, \alpha_2, \beta_2)|| \ldots ||(t_w, \alpha_w, \beta_w)$$

where t_l denotes the time it receives the query, α_l is the input of the query, and β_l is output of the query, $l = 1, \ldots, w$. If $\alpha_1 = \lambda$ then i is called the initiator, otherwise it is called the responder.

The formal definition of matching conversation is as follows:

Definition: Without loss of generality, let the number of flows R of a protocol Π be odd, and let A be the initiator and B be the responder. Running Π with adversary O and consider the two sessional protocols $\Pi^s_{\mathrm{A,B}}$ and $\Pi^t_{\mathrm{B,A}}$ with corresponding record of conversation being denoted as $\kappa^s_{\mathrm{A,B}}$ and $\kappa^t_{\mathrm{B,A}}$.

1. $\kappa^t_{\mathrm{B,A}}$ is a *matching conversation* to $\kappa^s_{\mathrm{A,B}}$ if there exist times $t_0 < t_1 < \ldots < t_R$ and $\alpha_1, \beta_1, \ldots, \alpha_\rho, \beta_\rho$ such that $\kappa^s_{\mathrm{A,B}}$ is prefixed by

   $$(t_0, \lambda, \alpha_1)||(t_2, \beta_1, \alpha_2)||(t_4, \beta_2, \alpha_3)|| \ldots ||(t_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1})||(t_{2\rho-3}, \beta_{\rho-1}, \alpha_\rho)$$

   and $\kappa^t_{\mathrm{B,A}}$ is prefixed by

   $$(t_1, \alpha_1, \beta_1)||(t_3, \alpha_2, \beta_2)||(t_5, \alpha_3, \beta_3)|| \ldots ||(t_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}).$$

2. $\kappa^s_{\mathrm{A,B}}$ is a *matching conversation* to $\kappa^t_{\mathrm{B,A}}$ if there exist times $t_0 < t_1 < \ldots < t_R$ and $\alpha_1, \beta_1, \ldots, \alpha_\rho, \beta_\rho$ such that $\kappa^t_{\mathrm{B,A}}$ is prefixed by

   $$(t_1, \alpha_1, \beta_1)||(t_3, \alpha_2, \beta_2)||(t_5, \alpha_3, \beta_3)|| \ldots ||(t_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1})||(t_{2\rho-1}, \alpha_\rho, *)$$

   and $\kappa^s_{\mathrm{A,B}}$ is prefixed by

   $$(t_0, \lambda, \alpha_1)||(t_2, \beta_1, \alpha_2)||(t_4, \beta_2, \alpha_3)|| \ldots ||(t_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1})||(t_{2\rho-3}, \beta_{\rho-1}, \alpha_\rho).$$

In simple words, if two parties have a matching conversation, with A being the initiator and B being the responder, then every message sent out by $\Pi^s_{\mathrm{A,B}}$, except possibly the last message, is faithfully delivered to $\Pi^t_{\mathrm{B,A}}$, with the response to this message being returned to $\Pi^s_{\mathrm{A,B}}$ as its own output.

To demonstrate this concept, consider a very simple 3-flow protocol. Without loss of generality, let Alice be the initiator and Bob the responder. In the first flow, Alice sends Bob message m_1, and Bob sends Alice the second flow message m_2. Then Alice sends the third flow message m_3 to Bob, the protocol terminates. Also assume that Alice thinks she is talking to Bob for the s^{th} time, and Bob thinks he is talking to Alice for the t^{th} time. Alice sends out the first message at time 0, and Bob receives it at time 1. Bob then sends the second flow message at time 1, Alice receives it at time 2, etc. The record of the first message then will be $(0, \lambda, m_1)$. If Alice and Bob have a matching conversation, then Bob will have received the message m_1 and sent out m_2 in the second flow. Therefore, Bob will have the record $(1, m_1, m_2)$, indicating at time 1 he receives message m_1 and sends out message m_2. Alice receives the second message at time 2, sends out the third message m_3, and thus have a record $(2, m_2, m_3)$. Bob receives the last message of the protocol, m_3, and has the record $(3, m_3, *)$. Therefore, Alice's record of conversation $\kappa^s_{\mathrm{A,B}}$ looks like

$$\kappa^s_{\mathrm{A,B}} = (0, \lambda, m_1)||(2, m_2, m_3)$$

and Bob's record of conversation $\kappa^t_{\mathrm{B,A}}$ looks like

$$\kappa^t_{\mathrm{B,A}} = (1, m_1, m_2)||(3, m_3, *)$$

The two records are matching conversations to each other.

Figure 4.1 gives an illustration and explanation of the concept of matching conversations. Notice, however, the record of conversation of each sessional protocol is somewhat simplified to illustrate the concept. In Bellare and Rogaway's model, a sessional protocol's input and output usually contain more than just the messages themselves in the protocol flows. In general, if $\Pi^s_{\mathrm{A,B}}$ and $\Pi^t_{\mathrm{B,A}}$ have a matching conversation, then their record of conversation are in the form of

$$(t_0, \lambda, \alpha_1)||(t_2, \beta_1, \alpha_2)||(t_4, \beta_2, \alpha_3)|| \ldots ||(t_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1})||(t_{2\rho-3}, \beta_{\rho-1}, \alpha_\rho)$$

and

$$(t_1, \alpha_1, \beta_1)||(t_3, \alpha_2, \beta_2)||(t_5, \alpha_3, \beta_3)|| \ldots ||(t_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1})||(t_{2\rho-1}, \alpha_\rho, *)$$

where α_i and βi are the input/output being recorded. Figure 4.2 shows how it works in the general case.

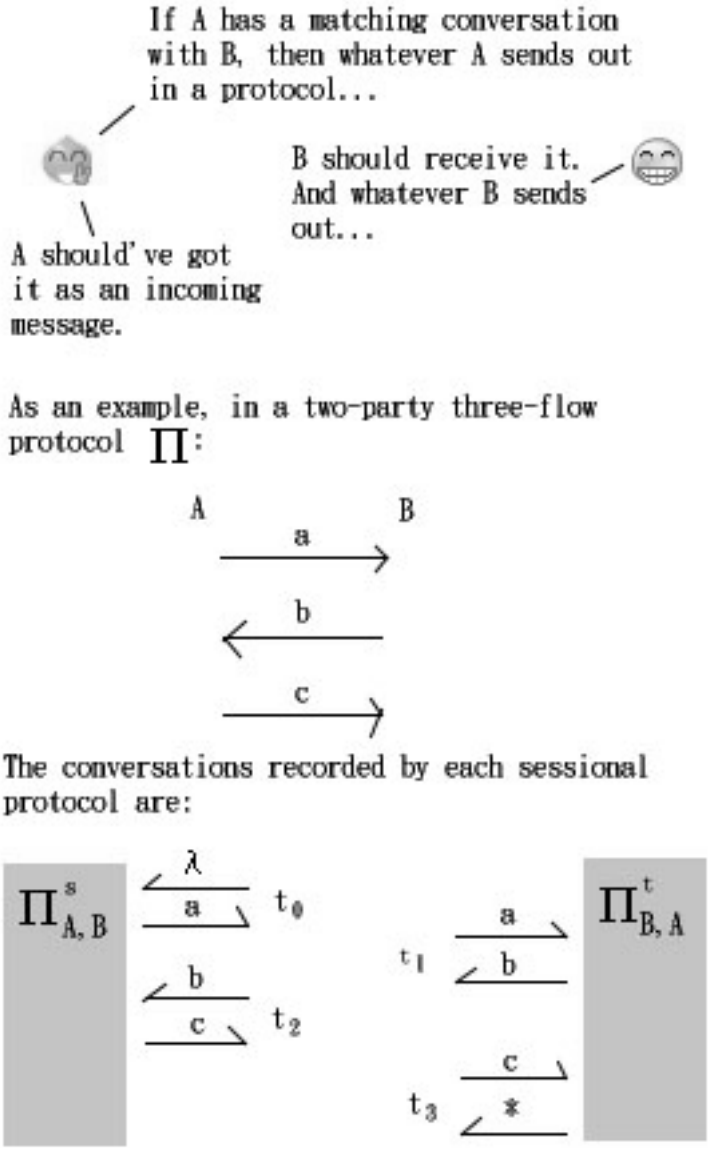

Figure 4.1: Illustration of the Concepts of Matching Conversations

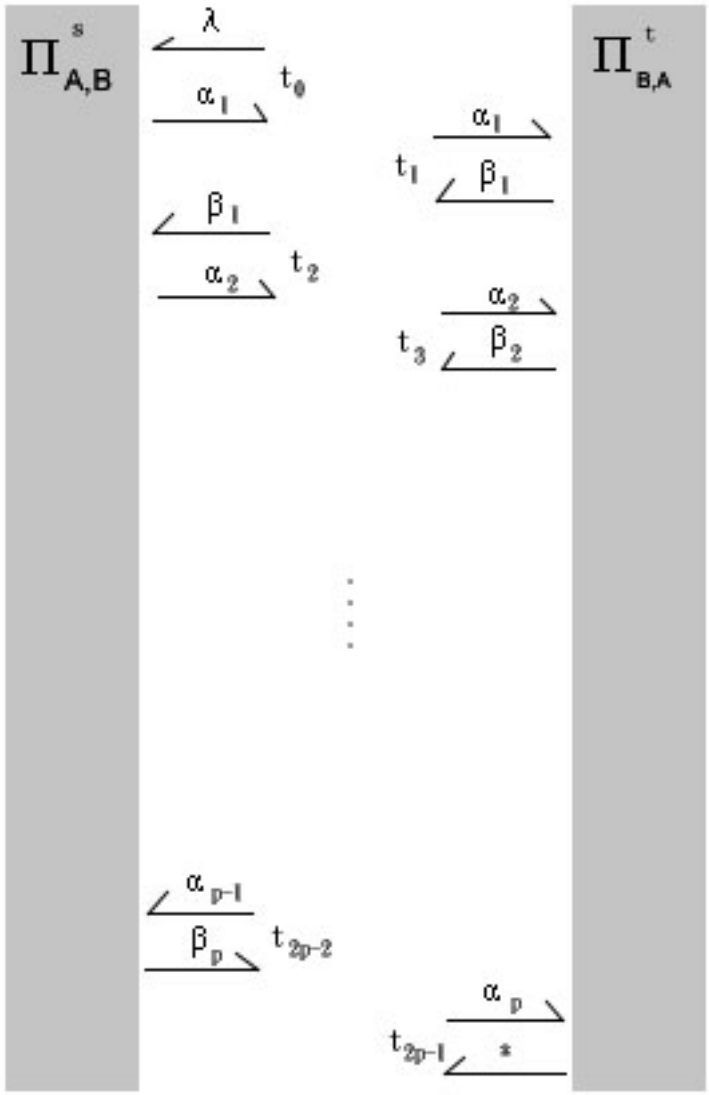

Figure 4.2: Illustration of the Concepts of Matching Conversations in General

In terms of mutual authentication, [BR93B] requires the assumption that the protocol must involve at least 3 protocol flows. Bellare and Rogaway suggest that in a mutually authentication protocol, when a party i accepts with conversation record K, then there must exist another party with an engaged matching conversation K'. Also, each party should accept in the 'absence' of the adversary.

Before describing the formal definition of secure mutual authentication in [BR93B], the term $\mathsf{No-Matching}^{\mathrm{O}}(k)$ must be introduced. $\mathsf{No-Matching}^{\mathrm{O}}(k)$ is the event where with a polynomial time adversary O and security parameter k, there exists a sessional protocol $\Pi^{s}_{i,j}$ which accepts, but there is no corresponding oracle $\Pi^{t}_{j,i}$ engaging a matching conversation with it. Then [BR93B]'s definition of mutual authentication is given as follows:

Definition: A protocol Π is a secure mutual authentication protocol if for any polynomial time adversary O,

1. If two sessional protocols $\Pi^{s}_{\mathrm{A,B}}$ and $\Pi^{t}_{\mathrm{B,A}}$ have a matching conversation, then both sessional protocols accept.
2. The probability of event $\mathsf{No-Matching}^{\mathrm{O}}(k)$ happening is negligible.

[BR93B] also mentioned the uniqueness of matching conversation partners. The event that more than one sessional protocol $\Pi^{t}_{\mathrm{B,A}}$ engaging matching conversations with $\Pi^{s}_{\mathrm{A,B}}$ is called the event of *multiple match*, denoted by $\mathsf{Multiple-Match}^{\mathrm{O}}(k)$. [BR93B] showed that if two parties are mutually authenticated, then event $\mathsf{Multiple-Match}^{\mathrm{O}}(k)$ should not happen.

Session Key Indistinguishability and Key Exchange

In [BCK98], Bellare and Rogaway treated the problem of authenticated key exchange as two sub-problems: mutual authentication and key exchange. After looking at the mutual authentication problem, the security of key exchange is treated with the concept of *indistinguishability* of session keys. Basically, Bellare and Rogaway suggested that the key exchange component of an AKE protocol should be in charge of protecting the session key information being leaked. The way to determine this is by asking if the adversary can distinguish the session key from any other random key.

As explained before, a secure key exchange process should prohibit the adversary knowing the session key with probability higher than guessing. However, if the adversary is able to perform reveal queries, obviously it can retrieve any session key

by just offering reveal. Therefore, the target of attack should be restricted such that the adversary cannot perform reveal on it. Bellare and Rogaway set the restriction as the following:

A sessional protocol $\Pi^s_{i,j}$ is called *fresh* if (1) it accepts, (2) it is unopened, and (3) there is no open sessional protocol $\Pi^t_{j,i}$ engaging a matching conversation with $\Pi^s_{i,j}$. If $\Pi^s_{i,j}$ is fresh, then the corresponding session key $sk^s_{i,j}$ is also said to be fresh.

Condition (1) captures the idea that there is no point for an adversary to ask a sessional protocol for its session key when it does not even accept. Condition (2) says that if the adversary is trying to attack a sessional protocol for which it already issued a reveal query, then it already has the session key and does not need to ask for it again. Condition (3) says that if the adversary knows the session key of the sessional protocol $\Pi^t_{j,i}$, which has developed a session key with $\Pi^s_{i,j}$, then attacking $\Pi^s_{i,j}$ will be trivial, too. Therefore, the above three cases cannot be considered as targets of the adversary's attack. This implies the condition on issuing the test query and definition of another event, $\mathsf{Good} - \mathsf{Guess}^{\mathrm{O}}(k)$:

Suppose the protocol is run; the adversary performs several send and reveal queries, then at the end it issues the test query to a sessional protocol, say $\Pi^s_{i,j}$. $\Pi^s_{i,j}$ must satisfy the above three conditions, and then the adversary is allowed to attack it.

After the test query has been sent, the issued protocol $\Pi^s_{i,j}$ performs the following steps:

1. a random bit $b \in_R \{0, 1\}$ is generated.
2. if $b = 0$ then the session key $sk^s_{i,j}$ is returned to the adversary.
3. Otherwise, a random value from the key space is returned to the adversary.

Seeing the returned string, the adversary must reply what the value b (which is not revealed to it) is. If the adversary successfully guesses the random bit b, it is referred to as the $\mathsf{Good} - \mathsf{Guess}^{\mathrm{O}}(k)$ event, with the presence of adversary O and security parameter k.

The concept of security is then defined as follows: the adversary should not be able to compute in polynomial time any information about the session key, and thus should not be able to tell the difference between the session key and any other random key from the key space.

Note that in pure guessing, the adversary is still able to get the value of b with probability of $1/2$. So if the adversary does better than guessing, it has some *advantage*. The function $\mathsf{adv}^{\mathrm{O}}(k)$ is defined as

$$\mathsf{adv}^{\mathrm{O}}(k) = \max\left\{0, \Pr[\mathsf{Good} - \mathsf{Guess}^{\mathrm{O}}(k)] - \frac{1}{2}\right\}$$

So the formal definition of a secure authenticated key exchange protocols is as follows:

Definition: A message-driven protocol Π is a secure message authentication protocol if

1. Π is a secure mutual authentication protocol.
2. If O is a benign adversary and $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ are two sessional protocols O runs experiments on, then both $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ accept, and $sk^s_{i,j} = sk^t_{j,i}$.
3. $\mathsf{adv}^{\mathrm{O}}(k)$ is negligible.

It can be easily seen that condition (3) is ensuring that the adversary cannot distinguish the session key with any other random string from the key space. Condition (2) ensures that the keys are distributed (in this case exchanged) properly if both parties of the protocol are honest and the adversary is a passive attacker.

4.4.4 Summary of the Model and Comments

[BR93B] was the first to realize the importance of formalizing notions of key distribution and key exchange. In order to define security of mutual authentication and key exchange, the concepts of matching conversation and indistinguishability of session keys were introduced, which greatly impacted the direction of defining security notions in the later work. In the definition of mutual authentication, the concept of matching intended participants was introduced.

[BR93B] informally described a design methodology to secure authenticated key exchange, but the proposed protocols in the paper are in the secret-key setting with the intended purpose of (possibly, group) key distribution, which is really an authenticated key *transport* protocol in today's terminology. However, the idea of taking a secure mutual authentication protocol and converting it into a secure authenticated key exchange can be adapted to other settings.

Although the model is intended for an extension to multiple-party key distribution, the paper discusses the case of two entities only, and if the setting has changed to just key establishment between any two entities of a large group, then obviously each pair of user's secret key should be different from another pair. In this case, the model may need to be modified to better reflect the real-life scenario.

4.5 The Blake-Wilson-Menezes Model

After the work by Bellare and Rogaway in [BR93B] and [BR93A], in 1997 Blake-Wilson and Menezes adapted the model in [BR93B] and established another model for authentication and authenticated key *transport* in the asymmetric setting. Another paper, published by Blake-Wilson, Johnson, and Menezes ([BWJM97]), discusses authenticated key exchange in the asymmetric setting. The model used in [BWJM97], though, is directly from [BWM97]. Therefore, the concepts used to describe the model are from [BWM97], but the relevant security definitions of authenticated key exchange come from both papers. Discussions and comments on the two papers will be given at appropriate places throughout this section.

Having viewed the above model, it will be easier to explain this model; some of the settings and assumptions are identical to the previous model, and the link from real-life scenario to formalization should be more understandable to the reader. Another significance of this paper is that some of the assumptions specified in this model are used as standard assumptions in the rest of this thesis. Therefore, it is worthwhile to discuss this model in detail, and then readers can understand the modelling in an easier way.

The organization of this subsection is as follows: the link connecting real life situations to formalization is omitted since most of the connections are described in the previous model, except for minor changes. Therefore, when there is a difference in modelling real life situations, explanations will be given. Otherwise, it is assumed that readers are familiar with the link between reality and formalization from the previous section. First, the formal model will be described, then the definition of a secure protocol is given. This section ends with some comments about this model, including some comparisons with the Bellare and Rogaway model.

4.5.1 The Formal Model

Preliminaries

- The setting is asymmetric; each user owns a public-private key pair for encryption, and another pair for signatures. This contrasts with the symmetric setting in the Bellare and Rogaway model, where each pair of users share a secret beforehand. It is also assumed in this model that keys and certificates of each user are generated and distributed before the protocol starts.
- Unlike the Bellare and Rogaway model, Blake-Wilson and Menezes focus specifically on key transport from one user to another, instead of a more

general key distribution between potentially many users. [BWJM97] focuses on key exchange in the asymmetric setting.

- The security parameter is denoted by a natural number k.
- $\{0,1\}^*$ denotes the set of finite binary strings (same as in the previous model).
- λ denotes the empty string.
- The set of users I is defined as $\{1, 2, \ldots, N_1\}$. The adversary O is not included in I. The size of I is constrained to be polynomial in k.
- A real-valued function $\epsilon(k)$ is *negligible* if for all constants $c > 0$, $\exists$ a $k_c > 0$ such that $\epsilon(k) < k^{-c}$ for all $k > k_c$.
- A protocol is defined to be a pair $P = (\Pi, \mathcal{G})$ of probabilistic polynomial-time computable functions.
- The long term keys are now defined in the public key setting. Each user has a public/private key pair for encryption, and another pair for signatures.
- The *benign adversary* is informally defined as an adversary that only (faithfully) relays messages in the protocol. It is sometimes also referred to as the *passive attacker*.

Running the Protocol

The usual meaning of a *protocol run* means a generic execution of a protocol between two users. However such a term in the model will be referred to as a *session*. Running the protocol will mean execution of the formal model, or sometimes referred to as the *experiment* done by the adversary.

The long-term key generator $\mathcal{G}$ in the Blake-Wilson and Menezes model generates key-pairs in the public-key setting. It uses two additional algorithms, $\mathcal{G}_{enc}$ to generate encryption key pairs, and $\mathcal{G}_{sig}$ to generate signature key pairs. The public/private key pair for encryption is denoted as (PEK, SEK), and the public/private key pair for signature is denoted as (PSK, SSK).

At the beginning of running the protocol, $\mathcal{G}$ takes a security parameter 1^k and a random input. It generates the (PEK, SEK) and (PSK, SSK) key pairs for each user, and distributes the keys to each user securely. (That is, the total keying information for each user i, denoted by K_i is equal to $((psk_i, ssk_i), (pek_i, sek_i)))$. The key generator also generates keys for the adversary. $\mathcal{G}$ also publishes a directory

called *public-info*. This directory contains triples of user i's public information $PK_i = (i, PEK_i, PSK_i)$, where PEK_i is i's public encryption key, and PSK_i is i's public signing key. Note that each $\mathcal{G}_{enc}$ and $\mathcal{G}_{sig}$ also may or may not need random numbers as input.

Π specifies the behaviour of each honest user of the protocol. Π takes the inputs $(1^k, i, j, K_{i,j}, \kappa)$, where

- 1^k is the security parameter.
- $i, j \in I$ represent the identity of the sender and the intended responder.
- $K_{i,j}$ is i's keying information (meaning the key pairs (PEK_i, SEK_i) and (PSK_i, SSK_i), denoted by K_i) together with j's public keying information $PK_j = (j, PEK_j, PSK_j)$.
- κ is the record of the ordered messages sent and received by i so far in the protocol.

The output of $\Pi(1^k, i, j, K_{i,j}, \kappa)$ is a triple $((m, a), \delta, sk)$, where

- $m \in \{0,1\}^* \cup \{*\}$ is the message to be sent from i to j
- a is the *appendix* of m. In the Blake-Wilson and Menezes model it specifically represents the signature on m.
- δ takes on three values: A as in *accept*, R as in *reject*, and the empty output $*$ as in *undecided*. It represents i's current decision.
- sk is the value of the exchanged key. It holds the value *null* before i accepts and will not be used in (entity) authentication-only protocols. (Note that the *null* value in the Bellare and Rogaway model is specified as $*$.)

Let $\Pi^{(m,a),\delta}$ denote the first two components of the output, and let Π^{sk} denote the third. That is, $\Pi^{(m,a),\delta}(1^k, i, j, K_{i,j}, \kappa) = ((m, a), \delta)$, and $\Pi^{sk}(1^k, i, j, K_{i,j}, \kappa) = sk$.

The sessional protocols are denoted by $\Pi^s_{i,j}$, where $i, j \in I$, and $s \in \mathbb{N}$. This notation represents entity i conducting a protocol run (or a session) to entity j for the sth time. Note that in Bellare and Rogaway's model, the value s is merely a natural number representing a session ID. However, s in Blake-Wilson and Menezes's model actually serves also as a counter of the number of times i believes it has talked to j. Each sessional protocol is associated with a record of conversation, denoted by

$\kappa^s_{i,j}$ for the sessional protocol $\Pi^s_{i,j}$. At the start of the protocol run, each oracle $\Pi^s_{i,j}$ is initialized with the security parameter 1^k and the keying information of user i (namely k_i), the conversation log $\kappa^s_{i,j}$, and the *public-info* directory. In which, the conversation log at the initialization phase is set to be the empty string λ.

The adversary is denoted by the name Oscar, and symbolized as O. It is modelled to have control of all communications, and it can reveal one or more particular session keys, reveal long-term private keys, and can ask different sessions to start at will. To formalize, the adversary is able to send four types of queries:

- **The Send Query:** this query is almost the same as the send query in the Bellare and Rogaway model. This query models the adversary sending a message to i pretending to be j, and it receives a reply from i. The query takes inputs $\Pi^s_{i,j}$ and $(m', a'))$ for some messages m' and secret information a', and it returns $\Pi^{(m,a),\delta}(1^k, i, j, sk_{ij}, \kappa||(m', a'))$, and updates κ to become $\kappa||(m', a')||(m, a)$.

- **The Reveal Query:** this type of queries asks a user to reveal its session key modelled by oracle $\Pi^s_{i,j}$. This type of query models the known-session key attack. The format of the query is of the form $\mathsf{Reveal}(\Pi^s_{i,j})$ and it returns $\Pi^k(1^k, i, j, sk_{ij}, \kappa)$. No oracle updates take place.

- **The Corrupt Query:** the query Corrupt tends to model two real-life scenarios: 1) the adversary O completely takes over the identity of a user $i \neq$ O, replacing i's private and public keys with the adversary's own choices, 2) the adversary may register a new public key for i without knowing i's private key. Also, after the adversary sends this query, he retrieves the value of i's keying information, k_i. Formally speaking, the Corrupt query takes two inputs i and K, and the oracle replies with k_i, user i's keying information, and the oracle updates k_i to become K. In case of the adversary only replacing a user's public keying information, the user's private keying information becomes the empty string λ.

- **The Test Query:** it is the challenge the adversary faces at the end of its control over the communication. It must decide the secret bit generated by the sessional protocol that it attacks.

The number of sessional protocols being controlled by the adversary is limited to a polynomial in the security parameter k. As a comparison, the Bellare and Rogaway's model allows the adversary to have control over as many sessional protocols as possible.

Running the protocol P with the presence of the adversary O and security parameter k takes the following steps:

1. Generate random inputs for $\mathcal{G}$, $\mathcal{G}_{enc}$, $\mathcal{G}_{sig}$, the adversary O, and each sessional protocol $\Pi^s_{i,j}$.
2. Run $\mathcal{G}$ with the security parameter 1^k and the random input to generate public/private key pairs (of encryption and signature schemes) for each entity $i \in I$ and the adversary. $\mathcal{G}$ also distributes the key pairs securely. That is, a user's private keying information is not exposed to the adversary. Then $\mathcal{G}$ compiles the public directory *public-info* and publishes it. Note that this type of setting only works in the scenario where all users must have certified and published their certificates and keying information before the protocol starts; registering certificates/keying information during a protocol run (or a session) is not allowed. Comparitively, in real life certificates are exchanged only when a session starts, a user i does not know j's certificate nor public keying information until it first communicates with it. The inclusion of *public-info* means that everyone knows each other's public keying information to start with, which is unrealistic.
3. Initialize each sessional protocol $\Pi^s_{i,j}$. Note again that the number of sessional protocols is polynomial in the security parameter k.
4. Start the adversary O on the security parameter 1^k, the random input, and the public directory *public-info*.
5. O starts its experiment at will. That is, it sends queries (either Send, Reveal, or Corrupt) to sessional protocols. In other words, the adversary can at any time ask a user i to reveal its session key sk_{ij} with user j, can ask a user i to reveal its long-term keying information K_i and replace i's long-term keying information.
6. After sending queries for a while, the adversary issues the test query to a sessional protocol that is fresh. (For definition of freshness and more on the Test query, please see below.) If the adversary successfully passes the Test query, then it succeeds; otherwise it fails.

4.5.2 Definition of a Secure Protocol

This subsection discusses the definition of a secure authenticated key exchange protocol only. However, some relevant side security issues, such as the security of a

signature scheme, are widely applied in authenticated key exchange protocols but not discussed in [BWJM97] (instead they are discussed in [BWM97]). Therefore, the security definition of signature schemes from [BWM97] and the security of MAC from [BWJM97] are discussed.

Similar to Bellare and Rogaway's model, Blake-Wilson and Menezes' model looks at the security of authenticated key exchange in two separate steps: (mutual entity) authentication and key exchange. The security of authentication is based on matching conversations, and the security of key exchange is based on session key indistinguishability.

Before going further, some terminology should be introduced.

- If there has been a $\mathsf{Corrupt}(i, \cdot)$ query,then $\Pi^s_{i,j}$ is said to be *corrupted*, for all $i, j \in I$, $s \in \mathbb{N}$.
- If there there has been a $\mathsf{Reveal}(\Pi^s_{i,j})$ query, then $\Pi^s_{i,j}$ is said to be *opened* for all i, j, s.
- $\Pi^s_{i,j}$ is said to have been *accepted* if $\Pi^{\delta}(1^k, i, j, sk_{ij}, \kappa^s_{i,j}) = \mathsf{A}$.

Security of Signature Schemes and MACs

Signature schemes and MACs are mostly used for data integrity and authentication. These two cryptographic tools are also widely used in designing authenticated key exchange protocols in the asymmetric setting. Therefore, it is crucial to take the security of signature schemes and MACs into consideration when developing the formal model.

The security of a signature scheme considered in Blake-Wilson and Menezes's model is inherited from the definition by Goldwasser, Micali, and Rivest ([GMR88]). This definition will hereafter being referred to as the GMR-security. Similarly, if a protocol is said to be GMR-secure, it satisfies the definition of security in [GMR88].

Formally (the definition is from [GMR88]), a signature scheme is a triple

$$(\mathcal{G}_{sig}, [\cdot]_{sig(\cdot)}, [\cdot, \cdot]_{ver(\cdot)})$$

of polynomial-time algorithms. On input 1^k, $\mathcal{G}_{sig}$ generates a key pair (PSK, SSK). To sign a message m, the entity with key pair (PSK, SSK) computes :

$$(m, \sigma) \leftarrow [m]_{sig(SSK)}$$

where σ is called the *signature* of m, and (m, σ) is called a *signed message*. To verify the validity of message m, one computes:

$$[m, \sigma]_{ver(PSK)} \in \{\mathsf{true}, \mathsf{false}\}$$

where the condition below must hold:

$$[m, \sigma]_{ver(PSK)} = \mathsf{true} \ \forall \ (m, \sigma) \in \{[m]_{sig(SSK)}\}.$$

The adversary O_{sig} of the signature scheme is defined as a probabilistic polynomial-time algorithm with access to a signature oracle. O_{sig} takes the input of some public signing key PSK, chosen by $\mathcal{G}_{sig}$, and outputs a pair (m, σ) where O_{sig} did not query the signing oracle on m. [GMR88] asserts that a signature scheme is secure if the probability of the adversary forging a signature on a new message is negligible. Formally, the definition of signature scheme's security (again from [GMR88]) is given by:

Definition: a signature scheme is secure if, for every adversary O_{sig} (of the signature scheme), the function $\epsilon(k)$ defined by

$$\epsilon(k) = \Pr[(PSK, SSK) \leftarrow \mathcal{G}_{sig}(1^k) | (m, \sigma) \leftarrow \mathrm{O}_{sig}(PSK) : [m, \sigma]_{ver(PSK)} = \mathsf{true}]$$

is negligible.

Note that in the definition above, the adversary can take a given message m, ask the signing oracle to give it a signed message (m, σ), and then compute σ' such that σ' is also the signature of m. This would cause incompatibility with the original definition of matching conversations.

The security of a MAC is based on [BRK96] by Bellare, Canetti, and Krawczyk. For the rest of this thesis, the security of MACs will be based on the definition in [BRK96].

Definition: (from [BRK96]) A message authentication code is a deterministic polynomial-time algorithm $\mathtt{MAC}_{(\cdot)}(\cdot)$. To authenticate a message m, an entity with key K computes

$$(m, a) = \mathtt{MAC}_K(m)$$

Here a is called the *tag* on message m, and (m, a) is called an *authenticated message*. To verify the validity of message m, any entity with the key K can check that $\mathtt{MAC}_K(m)$ is indeed equal to (m, a).

The adversary O_{MAC} (of a MAC) is defined as a polynomial-time probabilistic algorithm which has access to an oracle that computes MACs under a randomly

chosen but fixed key K. The output of O_{MAC} is a pair (m, a) such that O_{MAC} did not query m to the MAC oracle.

The security of a MAC, informally, is defined such that the probability of O_{MAC} forging a valid tag on any message that has not yet been authenticated using a call to the MAC oracle is negligible. This is referred to as a chosen-message attack. The formal definition is given as follows:

Definition: (from [BRK96]) The MAC algorithm is secure if, for every adversary O_{MAC} of the MAC, the function $\epsilon(k)$, defined by

$$\epsilon(k) = \Pr[K \leftarrow \{0,1\}^k; (m, a) \leftarrow \mathrm{O}_{MAC} | (m, a) = \mathtt{MAC}_K(m)]$$

is negligible.

Matching Conversation and (Entity) Authentication

Similar to Bellare and Rogaway's model, [BWM97] and [BWJM97] both define mutual authentication using the concept of matching conversations. However, the matching conversation described in [BWM97] is slightly modified due to the incompatibility of the GMR-security of signature schemes and the original definition of matching conversation. The definition of a secure MAC, however, is compatible with the original definition of matching conversations, and hence it was not changed in [BWJM97].

Recall that with the GMR definition of secure signature schemes, the adversary can take a given message m, ask the signing oracle to give it a signed message (m, σ), and then (perhaps) compute σ' such that σ' is also the signature of m. [BWM97] explained that such an ability causes the incompatibility of the standard definition of matching conversation and security of signature schemes, but the reason for why this is the case is still unknown up to this point.

For the description of the original definition of matching conversation, please refer to the previous model. The modified matching conversation is described as follows:

The record of conversation of a particular sessional protocol $\Pi^s_{i,j}$ should take the form of

$$\begin{aligned} \kappa^s_{i,j} \quad = \quad & (t_1, (m_1, a_1), (\mu_1, \alpha_1)) || \\ & (t_2, (m_2, a_2), (\mu_2, \alpha_2)) || \ldots \\ & (t_w, (m_w, a_w), (\mu_w, \alpha_w)) \end{aligned}$$

where t_l denotes the time the sessional protocol receives the query, (m_l, a_l) is the input message and signature of the message input to $\Pi^s_{i,j}$ at time t_l. (μ_l, α_l) is the output message and signature of the message from $\Pi^s_{i,j}$. If $(m_1, a_1) = \lambda$ then $\Pi^s_{i,j}$ is referred to as the *initiator*; otherwise it is referred to as the *responder*.

Definition: (from [BWM97]) Without loss of generality, assume the number of flows of a protocol, R, is odd. Protocol $P = (\Pi, \mathcal{G})$ is run with adversary O. Consider an initiator sessional protocol $\Pi^s_{i,j}$ and a responder sessional oracle $\Pi^t_{j,i}$ that O experiments on. Let the corresponding record of conversation be denoted as $\kappa^s_{i,j}$ and $\kappa^t_{j,i}$, respectively.

1. $\kappa^t_{j,i}$ is said to be a *matching conversation* to $\kappa^s_{i,j}$ if there exists $t_0 < t_1 < \ldots < t_{R-1}$ such that $\kappa^s_{i,j}$ is prefixed by

   $$(t_0, \lambda, (m_1, a_1))||(t_2, (\mu_1, \alpha'_1), (m_2, a_2))|| \ldots ||(t_{2\rho-2}, (\mu_{\rho-1}, \alpha'_{\rho-1}), (m_\rho, a_\rho))$$

   and $\kappa^t_{j,i}$ is prefixed by

   $$(t_1, (m_1, a'_1), (\mu_1, \alpha_1))||(t_3, (m_2, a'_2), (\mu_2, \alpha_2))|| \ldots || \\ (t_{2\rho-3}, (m_{\rho-1}, a'_{\rho-1}), (\mu_{\rho-1}, \alpha_{\rho-1})).$$

   $\kappa^t_{j,i}$ is said to be a *matching conversation including appendices* to $\kappa^s_{i,j}$ if additionally $a_1 = a'_1, \ldots, a_{\rho-1} = a'_{\rho-1}$ and $\alpha_1 = \alpha'_1, \ldots, \alpha_{\rho-1} = \alpha'_{\rho-1}$.

2. $\kappa^s_{i,j}$ is said to be a *matching conversation* to $\kappa^t_{j,i}$ if there exists $t_0 < t_1 < \ldots < t_{R-1}$ such that $\kappa^t_{j,i}$ is prefixed by

   $$(t_1, (m_1, a'_1), (\mu_1, \alpha_1))||(t_3, (m_2, a'_2), (\mu_2, \alpha_2))|| \ldots \\ ||(t_{2\rho-3}, (m_{\rho-1}, a'_{\rho-1}), (\mu_{\rho-1}, \alpha_{\rho-1}))||(t_{2\rho-1}, (m_\rho, a'_\rho), *)$$

   and $\kappa^s_{i,j}$ is prefixed by

   $$(t_0, \lambda, (m_1, a_1))||(t_2, (\mu_1, \alpha'_1), (m_2, a_2))|| \ldots ||(t_{2\rho-2}, (\mu_{\rho-1}, \alpha'_{\rho-1}), (m_\rho, a_\rho)).$$

   $\kappa^s_{i,j}$ is said to be a *matching conversation including appendices* to $\kappa^t_{j,i}$ if additionally $a_1 = a'_1, \ldots, a_{\rho-1} = a'_{\rho-1}$ and $\alpha_1 = \alpha'_1, \ldots, \alpha_{\rho-1} = \alpha'_{\rho-1}$.

The above definition is referred to as *matching conversation including appendices*. The concept is that a matching conversation in the asymmetric setting should not just have the adversary faithfully delivering the messages, but also the appendices (such as signatures) between the two sessional protocols.

Let $\mathsf{No-Match}^{\mathrm{O}}(k)$ denote the event that when protocol P is run against adversary O with security parameter k, there is an uncorrupted sessional protocol $\Pi^s_{i,j}$ which accepts, but there is no corresponding sessional protocol $\Pi^t_{j,i}$ which engages in a matching conversation to $\Pi^s_{i,j}$.

Definition: (from [BWM97]) A protocol P is a secure mutual authentication protocol if for all adversary O:

1. If two sessional protocols $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have a matching conversation including appendices, then both sessional protocols accept.
2. The probability of event $\mathsf{No-Matching}^{\mathrm{O}}(k)$ is negligible.

In [BWJM97], no such definition is given regarding the security of mutual authentication. In fact, in the two types of authenticated key agreement protocols (authenticated key exchange and authenticated key exchange with key confirmation), the term mutual (entity) authentication is not explicitly mentioned and does not appear in both protocols' definition of security. However, mutual entity authentication is an important part of the definition of a secure authenticated key transport protocol in [BWM97], and the concept of matching definition including appendices is crucial with the security of signature schemes. Thus it is very important to mention the concepts of matching conversation including appendices and mutual authentication (including appendices) in this subsection.

Secure Authenticated Key Exchange and Secure Authenticated Key Exchange with Key Confirmation

The two types of authenticated key exchange protocols mentioned in [BWJM97] are authenticated key exchange with implicit key authentication (denoted as an AKI protocol) and authenticated key exchange with explicit key authentication or *key confirmation* (denoted as an AKC protocol). Part of the notion of security in an AKI or AKC protocol is adversary's incapability of distinguishing the session key with any random key from the key space (session key indistinguishability).

Because of the necessity of using appendices in the asymmetric setting and the different notion of matching conversations, we use the term *matching conversation* and $\mathsf{No-Matching}^{\mathrm{O}}(k)$ in a context-free manner: whenever a signature is used as appendices, then the definition of matching conversation including appendices and the corresponding $\mathsf{No-Matching}$ event should be used. Otherwise, the "standard" notion of matching conversation and $\mathsf{No-Matching}$ should be used.

One assumption in treating the security of authenticated key exchange protocols is that no session key will be computed by a sessional protocol unless it has accepted. Note that the event $\mathsf{No-Matching}^{\mathrm{O}}(k)$ is defined the same way as in [BR93B].

A sessional protocol $\Pi^s_{i,j}$ is said to be *fresh* if (1) both i and j are not corrupted, (2) $\Pi^s_{i,j}$ is unopened, (3) it has accepted, and (4) there is no opened sessional protocol $\Pi^t_{j,i}$ engaging in a matching conversation with $\Pi^s_{i,j}$. As well, if a sessional protocol is fresh, then its corresponding session key is fresh, too.

The test query and the event $\mathsf{Good-Guess}^{\mathrm{O}}(k)$ is defined exactly the same as in [BWM97]. That is, at the end of the adversary's experiment, it calls the test query on some fresh sessional oracle $\Pi^s_{i,j}$. The sessional protocol 'tosses a coin' and gets a random bit $b \in \{0,1\}$. If $b = 0$ then $\Pi^s_{i,j}$ returns its session key $sk^s_{i,j}$ to the adversary; it returns a random key from the key space, otherwise. The adversary now must guess what the bit b is. If O successfully guesses the random bit, then $\mathsf{Good-Guess}^{\mathrm{O}}(k)$ happens. The advantage of the adversary, denoted by $\mathsf{adv}^{\mathrm{O}}(k)$, is defined as

$$\mathsf{adv}^{\mathrm{O}}(k) = \left| \Pr[\mathsf{Good-Guess}^{\mathrm{O}}(k)] - \frac{1}{2} \right|$$

which is equivalent to the definition in [BR93B].

The formal definition of AKI and AKC are described below:

Definition: (from [BWJM97]) A protocol $P = (\Pi, \mathcal{G})$ is a secure AKC protocol if

1. If O is a benign adversary on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, then both sessional protocols accept with the same session key sk, and the key is uniformly randomly distributed over $\{0,1\}^k$.
2. For all adversaries O, if uncorrupted sessional protocols $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have a matching conversation, then both protocols accept and hold the same session key sk.
3. The probability of $\mathsf{No-Matching}^{\mathrm{O}}(k)$ is negligible.
4. $\mathsf{adv}^{\mathrm{O}}(k)$ is negligible.

Condition (1) asserts that a passive attacker cannot have any impact on the security of the protocol. Condition (2) says that if both parties are honest and the transmissions between them are not altered, then both of them should accept and

share the same session key. The third condition says that the adversary cannot make any sessional oracle to accept once it alters the transmitted message. [BWJM97] interprets this condition as 'essentially the only way for any adversary to get an uncorrupted entity to accept in a run of the protocol with any other uncorrupted entity is by relaying communications like a wire'. However, we believe that condition (3) also serves the concept of *matching intended participants*. That is, if i's intended participant is j in a protocol run, then the security of this protocol will ensure that j is aware that it is engaged in a this particular protocol run with i. The negligible probability of $\mathsf{No-Matching}$ says that for all sessional protocols $\Pi^s_{i,j}$ that accept, there exists a corresponding sessional protocol $\Pi^t_{j,i}$ that engages in a matching conversation with it. That is, if i believes that it has involved in a communication with entity j and reaches the conclusion that they share a key sk (i has accepted), then j must be convinced that it is also involved in this protocol run with i and shares the same session key sk. However, the interpretation of [BWJM97] does not seem to suggest this idea.

Definition: (from [BWJM97]) A protocol P is a secure AKI protocol if

1. If O is a benign adversary on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, then both sessional protocols accept with the same developed session key sk, and the key is uniformly randomly distributed over $\{0,1\}^k$.

2. For all adversaries O, if uncorrupted sessional protocols $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have a matching conversation, then both protocols accept and hold the same session key sk.

3. The probability of $\mathsf{adv}^{\mathrm{O}}(k)$ is negligible.

Note that the difference between an AKI protocol and an AKC protocol is that a secure AKC protocol requires the event $\mathsf{No-Matching}$ to have negligible probability, but AKI does not. At a glance of [BWJM97], the definition of a secure AKI seems to be inadequate since one would hope to have negligible $\mathsf{No-Matching}$ probability for any secure authenticated key exchange protocol. After consulting with the author of [BWJM97], it was clarified that the definition of AKC was the proposed definition of a secure AK protocol. Therefore, the authors of [BWJM97] also agreed that the negligible $\mathsf{No-Matching}$ probability property should be preserved in the definition of a secure AKE (authenticated key exchange) protocol.

4.6 The Bellare-Canetti-Krawczyk's Modularization Model

In 1998, Bellare, Canetti, and Krawczyk ([BCK98]) introduced a different design methodology of authenticated key exchange protocols. Their model not only supports key exchange, but more general authentication and authenticated key establishment protocols. As a result, a new model was needed to provide the proof of security of their proposed protocol. They further extended the model in [CK01] to show other applications in other fields of cryptography, but this section mainly focuses on the basic model and the relevant issues regarding authenticated key exchange.

The model was developed with the specific intention of solving authenticated key exchange. The approach is to first design a key exchange protocol Π that is secure in an ideal, authenticated setting, prove its security, then transfer this protocol Π into a secure *authenticated* key exchange protocol Π' in the unauthenticated setting using an *authenticator*. The authenticator also ensures that the protocol Π in the authenticated setting has an equivalent behaviour as protocol Π' in the unauthenticated protocol.

Having this concept in mind, their formalization involves two models: the first one is an Authenticated-links Model (AM) and the second one an Unauthenticated-links Model (UM).

4.6.1 The Authenticated-Links Model (AM)

The user set I contains n users $1, 2, \ldots, n$, each running a copy of a protocol Π (where Π is defined as being controlled by the adversary and it only responds when the adversary sends a query to it). Π is called a *message driven protocol*. The adversary in this model, for ease of recognition, is denoted by O_{AM}. Similar to the previous model, the communication between each party is modelled as follows: the adversary asks a sessional protocol of the sender i $\Pi^s_{i,j}$ to initiate a protocol run (this is referred to as an *external request*) or sending a message to a sessional protocol $\Pi^s_{i,j}$ for response (this is referred to as an *activation* upon input message m). The adversary can see any public response returned by a protocol, but not the internal state change of the protocol. In the AM model, O_{AM} is restricted to delivering messages faithfully (that is, being a passive attacker). This restriction is modelled by having a temporary *set of undelivered messages* M. Whenever O_{AM} sends an external requests to a sessional protocol $\Pi^s_{i,j}$, the output message m and

the intended recipient j are included in the set M. When O_{AM} sends a query (activates) a sessional protocol $\Pi^{s'}_{j,i}$ on an input message m, m must be in set M and m's associated intended recipient must be j. However, the adversary is not required to maintain the order of messages being sent; it can send unlimited number of external requests (therefore he controls an unlimited number of such $\Pi^{s}_{i,j}$s); it is not required to maintain any fairness of activating protocols; and it is not required to deliver all messages. On top of that, the adversary can also send Corrupt queries to a user. Once a user i is corrupted, all the internal state information (including its long-lived keying information) is known to O_{AM}. The adversary can also include any message m in the set M as long as the sender is specified as the corrupted party i. From the time that i is corrupted, any messages returned by $\Pi^{s}_{i,j}$ will have a special tag specifying i has been corrupted.

The output observed by the adversary, together with its random output, is called the *adversarial view*. The concatenation of all cumulative outputs of all sessional protocols and the adversary view is called the *global output*. Let $\mathrm{ADV}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})$ denote the adversary view of adversary O_{AM} interacting with users running protocol Π on input $\overrightarrow{x} = (x_1 x_2 \dots x_n)$ and random input $\overrightarrow{r} = (r_0 r_1 r_2 \dots r_n)$ (r_0 is the random input in starting the adversary, and x_i, r_i are the input and random input to start-up the sessional protocols associated with user i). Let $\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})_i$ denote the output of user i, running Π on input $\overrightarrow{x}$ and random input $\overrightarrow{r}$, and with adversary O_{AM}. Let $\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})$ denote the global output, that is, $\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r}) = \mathrm{ADV}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})||\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})_1||$ $\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})_2|| \dots ||\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})_n$. Let $\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x})$ denote the random variable describing $\mathrm{AUTH}_{\Pi,\mathrm{O}_{\mathsf{AM}}}(\overrightarrow{x}, \overrightarrow{r})$ when $\overrightarrow{r}$ is uniformly chosen.

4.6.2 The Unauthenticated-Links Model (UM)

The computation model of the unauthenticated-links model is similar, except the power of the adversary has been strengthened. To distinguish the two types of adversaries, let O_{UM} denote the adversary in the unauthenticated-links model. Also, if we wish to address the name of the adversaries, O_{AM} will be called AM-Oscar and O_{UM} will be addressed as UM-Oscar. Unlike AM-Oscar, UM-Oscar can now send queries to any sessional protocol with any arbitrary message m, not just limited to messages in the undelivered message set.

The protocol is augmented by an *initialization function* $\mathcal{G}$ that generates and distributes public-key information and private keys for each entity i. Formally, $\mathcal{G}$ takes a random input r and outputs a vector $\mathcal{G}(r) = \mathcal{G}(r)_0 \mathcal{G}(r)_1 \dots \mathcal{G}(r)_n$, where

$\mathcal{G}(r)_0$ is the public key information and known to all entities (including the adversary). $\mathcal{G}(r)_i$ is the private key information of entity i, and this piece of information is distributed in a secure way that only i knows the value and no one else does.

Another two notations, $\mathrm{UNAUTH}_{\Pi,\mathsf{O_{UM}}}(\overrightarrow{x}, \overrightarrow{r})$ and $\mathrm{UNAUTH}_{\Pi,\mathsf{O_{UM}}}(\overrightarrow{x})$ are defined similarly to $\mathrm{AUTH}_{\Pi,\mathsf{O_{AM}}}(\overrightarrow{x}, \overrightarrow{r})$ and $\mathrm{AUTH}_{\Pi,\mathsf{O_{AM}}}(\overrightarrow{x})$

4.6.3 Emulation

The key point of Bellare, Canetti, and Krawczyk's model is the authenticators that transform a protocol Π in the ideal, authenticated setting into an equivalent protocol Π' in the realistic, unauthenticated model. By "equivalent", [BCK98] informally states it should mean "running Π' in an unauthenticated network has the same effect as running Π in an authenticated network". More formally, we have the following defintition:

DEFINITION: Let Π and Π' both be message-driven protocols. The protocol Π' in the unauthenticated network *emulates* Π in the authenticated network if for every UM-adversary O_{UM}, there exists an AM-adversary O_{AM} such that for all inputs $\overrightarrow{x}$,

$$\mathrm{AUTH}_{\Pi,\mathsf{O_{AM}}}(\overrightarrow{x}) \approx_c \mathrm{UNAUTH}_{\Pi,\mathsf{O_{UM}}}(\overrightarrow{x})$$

where $\approx_c$ denotes "computationally indistinguishable".

That is, if Π' emulates Π, then the combined distribution of the adversary and all the parties, and the identity of corrupted users on all input $\overrightarrow{x}$, should be indistinguishable for Π' and Π.

4.6.4 Authenticators and MT-Authenticators

An *authenticator* in [BCK98] is defined as an algorithm that takes as input the description of a protocol Π in the authenticated setting, and outputs the description of another protocol Π', in which Π' emulates Π in the unauthenticated setting. In other words, an authenticator can take a protocol secure in the authenticated setting, then translates it into a secure protocol (equivalent to the first one) in the unauthenticated setting.

The construction of authenticators is through a layered, two-step process. First a simpler protocol called the *MT-authenticator* is introduced. Its only purpose is

to deliver messages from one entity to another in an authenticated manner (in an unauthenticated network). Then the second step is to construct an authenticator based on the MT-authenticator. The MT-authenticator is described as follows:

Define the *MT-protocol* MT in the following way. The protocol takes empty input. Upon an external request to entity i, of the form (j, m), i sends the message (i, j, m) to user j, and outputs the message 'i sent m to j'. Upon receiving message (i, j, m), j outputs 'j received m from i'. The MT-authenticator mt is a protocol that emulates MT in unauthenticated settings.

An authenticator can be constructed from an MT-authenticator mt. Let C_{mt} be an algorithm described as follows: Given a protocol Π in the authenticated network, apply C_{mt} to Π to get $\Pi' = C_{\mathsf{mt}}(\Pi)$; Π' invokes mt. Whenever a message is to be sent in Π, Π' sends an external request to mt, asking to send the same message to the specified recipient. Whenever a message is received in protocol Π', it activates mt with this incoming message. After mt outputs 'i received m from j' for the incoming message m, Π is activated with incoming messages m from user j. In [BCK98] C_{mt} is proven to be an authenticator; that simplifies the task of creating an authenticator to creating a protocol that serves a narrower purpose, namely the MT-authenticator. The readers can refer to [BCK98] for the proof.

4.6.5 The Ideal Key Exchange Process

In [BCK98], the secure key exchange protocols are defined through a formal model as well. Basically, the adversary's power is limited to a certain extent that is the best we can expect from a key exchange protocol. Then any key exchange protocol which can emulate the ideal key exchange process is said to be secure.

Informally, the model consists of multi-users, each may have multiple key exchange sessions with another entity. The adversary is limited so that the exposure of one session key between two users should not allow the adversary to gain knowledge of any other session keys and long-term keys.

Formally, the user set I consists of n users $\{1, 2, \ldots, n\}$, the adversary in the ideal key exchange setting is referred to as the ideal-KE-adversary (also known as ideal-KE-Oscar or $\mathrm{O}_{IDEAL-KE}$). There is also a trusted party T that distributes or generates information securely without letting the adversary know the values of distribution. Again the communication is controlled by the adversary, and the adversary now has four types of queries:

1. **Invoke i (as an initiator) to exchange a new key with j:** the effect of this query is that the output of i is concatenated with a new value 'i

established key (sk, s) with j', where sk is the new session key chosen via some predetermined distribution, s is the session ID, in the format of $(i, j, c, init)$, indicating that it is the cth time i establishes a key with j, and the value $init$ indicates that i is the initiator. The adversary is able to learn the value of s but not sk, and the value s is added to the global set of incomplete sessions (denoted by IS).

2. **Invoke j (the responder) to exchange a key of session s with i:** the adversary is only allowed to send this query if s is in IS and s is of the form $(i, j, c, init)$ for some initiator i and numerical value c. The effect of this query is that: 1) the value 'j has exchanged key (sk, s') with i', where sk is the established key and s' is of the form $(i, j, c, resp)$. $resp$ indicate j is the responder of the protocol, and the other values of s' must match the corresponding values in s. Here j is thought to receive the value sk securely by an trusted party, which does not reveal the value of sk to the adversary. Also the value s is deleted from the set of incomplete sessions IS.

3. **Corrupt session s:** let s be of the form $(i, j, c, init)$ with the corresponding $s' = (i, j, c, resp)$ and suppose the the exchanged key is sk. Session s can only be corrupted if s is currently in IS or was but not now in IS. If s is currently in IS, then the value 's is corrupted' is appended to the output of the initiator, and s is not deleted from IS. If s is no longer in IS, then both 's is corrupted' and 's' is corrupted' are appended to the output of the initiator. The adversary also learns the value of sk.

4. **Corrupt party i:** the effect of this query is that all key values known to user i is now known to the adversary, and the value 'i is corrupted' is added to i's output.

Let the notation $\mathrm{ADV}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)$ represent the output of the adversary $\mathrm{O}_{\mathsf{IDEAL-KE}}$ of a run on random input $r_{\mathrm{Oscar}_{\mathsf{IDEAL-KE}}}$ and the keys are chosen by T using random input r_T. Similarly, let $\mathrm{IDEAL}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)_i$ denote the output of user i on inputs $r_{\mathrm{Oscar}_{\mathsf{IDEAL-KE}}}$ and r_T. Let $\mathrm{IDEAL}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)$ denote the concatenation of the adversary's view and all other user's view. That is,

$$\begin{aligned}\mathrm{IDEAL}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T) \;=\; & \mathrm{ADV}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)|| \\ & \mathrm{IDEAL}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)_1|| \\ & \mathrm{IDEAL}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)_2||\ldots|| \\ & \mathrm{IDEAL}_{\mathrm{O}_{\mathsf{IDEAL-KE}}}(r_{\mathrm{O}_{\mathsf{IDEAL-KE}}}, r_T)_n\end{aligned}$$

Also, let $\text{IDEAL}_{\text{O}_{\mathsf{IDEAL}}}()$ denote the random variable describing $\text{IDEAL}_{\text{O}_{\mathsf{IDEAL}}}(r_{\text{O}_{\mathsf{IDEAL}}}, r_T)$ when $r_{\text{O}_{\mathsf{IDEAL}}}$ and r_T are uniformly chosen.

4.6.6 How to Prove the Security of a Key Exchange Protocol

Unlike [BR93B] and [KM04]'s main focus on formal modelling, [BCK98] was written with the purpose of designing a secure protocol, and thus the way of proving security is slightly different from the other two. In particular, [BR93B] and [KM04] define security in terms of the probability of the adversary knowing the exchanged session key, and the probability of a mismatched conversation (see the previous sections). In [BCK98], the security of a key exchange protocol is represented as what the adversary can do under a secure situation. Then given the protocol in the authenticated and unauthenticated links model, we try to prove the adversary in both models cannot do better than the adversary in the secure situation. Formally, [BCK98] gives the following definition:

1. **Secure in the AM-Model:** If Π is a message driven key exchange protocol for n parties, then Π is a secure key exchange protocol in the authenticated links model if for all AM-adversary O_{AM}, there exists an ideal-KE-adversary $\text{O}_{\mathsf{IDEAL}}$ such that
   $\text{AUTH}_{\Pi,\text{O}_{\mathsf{AM}}}() \approx_c \text{IDEAL}_{\text{O}_{\mathsf{IDEAL}}}()$
2. **Secure in the UM-Model:** If Π is a message driven key exchange protocol for n parties, then Π is a secure key exchange protocol in the unauthenticated links model if for all UM-adversary O_{UM}, there exists an ideal-KE-adversary $\text{O}_{\mathsf{IDEAL}}$ such that
   $\text{UNAUTH}_{\Pi,\text{O}_{\mathsf{UM}}}() \approx_c \text{IDEAL}_{\text{O}_{\mathsf{IDEAL}}}()$

An illustration is given in Figure 4.3.

Then proof of security is based on two steps:

1. Given a key exchange protocol Π, associate it with an AM-adversary, and prove that this protocol is a secure key exchange protocol in the authenticated network (by showing that for any AM-adversary O_{AM} there exists an ideal-KE-adversary $\text{O}_{\mathsf{IDEAL}}$ such that $\text{AUTH}_{\Pi,\text{O}_{\mathsf{AM}}}() \approx_c \text{IDEAL}_{\text{O}_{\mathsf{IDEAL}}}()$).
2. Apply an authenticator C to Π to get another protocol $\Pi' = C(\Pi)$. By Corollary 8 of [BCK98] conclude that Π' is a secure key exchange protocol in the unauthenticated links model.

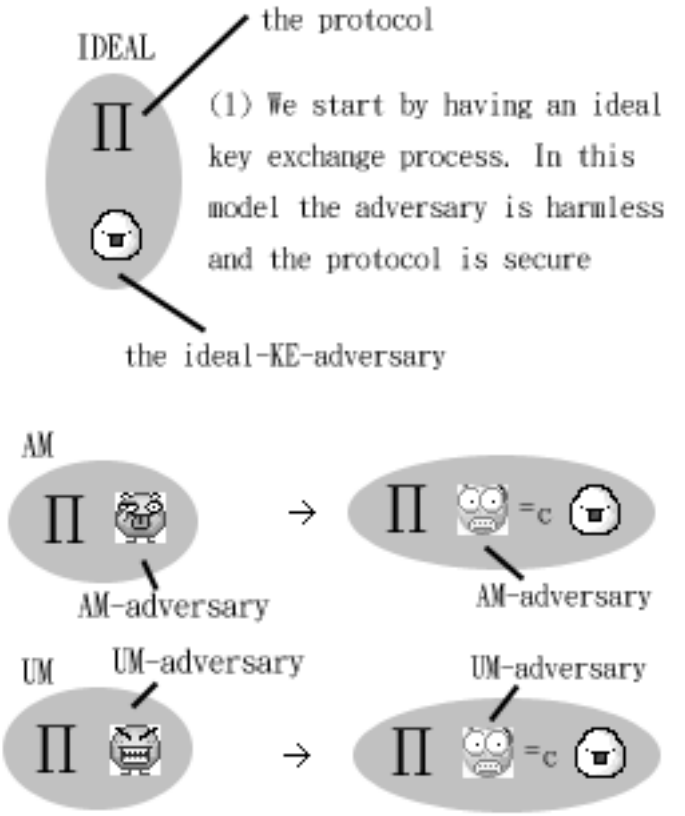

Figure 4.3: How to prove security of a key exchange protocol

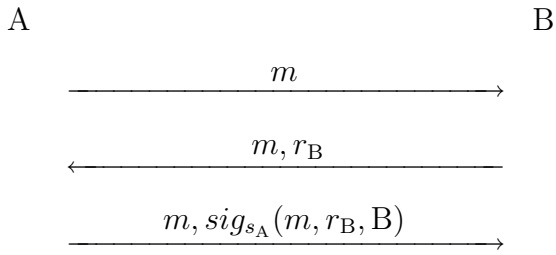

Figure 4.4: Signature based MT-authenticator

4.6.7 Examples of MT-Authenticators

[BCK98] gives two examples of MT-authenticators; one is based on signature schemes and the other on an encryption scheme and a MAC. The authentication is based on a challenge-response design. That is, when A wishes to send a message m to B, she sends m to B directly. To ensure m is really from A, B replies with a challenge to A. A must respond to the challenge along with the inclusion of her first message m, then B accepts. In the following, the two MT-Authenticators are described:

The Signature based MT-Authenticator

1. The initiator A sends message m to B (A also outputs 'A sent m to B' as the protocol specifies).
2. Upon receiving m from A, B selects a random number $r_{\mathrm{B}} \in_R \{0,1\}^k$ and sends a challenge m, r_{B} to A.
3. Upon receiving the challenge m, r_{B} from B, A computes her signature $sig_{s_{\mathrm{A}}}(m, r_{\mathrm{B}}, \mathrm{B})$ as the response and sends $m, sig_{s_{\mathrm{A}}}(m, r_{\mathrm{B}}, \mathrm{B})$ to B.
4. Upon receiving $m, sig_{s_{\mathrm{A}}}(m, r_{\mathrm{B}}, \mathrm{B})$ from A, B accepts m (that is, it outputs 'B received m from A').

As long as the signature scheme used in this MT-authenticator is secure against chosen message attacks (that is, it satisfies the GMR ([GMR84, GMR88]) security requirements), then the above protocol emulates an authenticator protocol in unauthenticated networks. An illustration is shown in Figure 4.4. (Note that $sig_{s_{\mathrm{A}}}$ means the signature of something using A's signing key s_{A}).

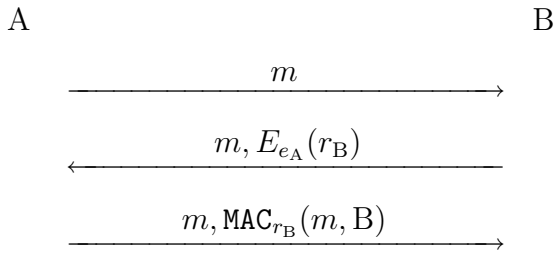

Figure 4.5: Encryption and MAC based MT-authenticator

The Encryption and MAC based MT-Authenticator

1. The initiator A sends message m to B (A also outputs 'A sent m to B' as the protocol specifies).

2. Upon receiving m from A, B selects a random number $r_{\mathrm{B}} \in_R \{0,1\}^k$ and sends $m, E_{e_{\mathrm{A}}}(r_{\mathrm{B}})$ as a challenge to A.

3. Upon receiving the challenge $m, E_{e_{\mathrm{A}}}(r_{\mathrm{B}})$ from B, A computes the MAC value of m, B using r_{B} as the MAC key. She then sends $m, \mathtt{MAC}_{r_{\mathrm{B}}}(m, \mathrm{B})$ to B.

4. Upon receiving $m, \mathtt{MAC}_{r_{\mathrm{B}}}(m, \mathrm{B})$ from A, B accepts m (that is, it outputs 'B received m from A'),

where $E_{e_{\mathrm{A}}}$ denotes the encryption of something using A's public encryption key e_{A}, and $\mathtt{MAC}_{r_{\mathrm{B}}}$ denotes the MAC value using B's random challenge as its MACing key.

This MT-authenticator emulates an authenticator protocol in unauthenticated networks as long as the encryption scheme is secure against the chosen ciphertext attacks (see [RS91]).

4.6.8 Summary and Discussion

The Bellare-Canetti-Krawcyzk's model in [BCK98] takes a different approach to secure authenticated key exchange from [BR93B] and [BWM97]. The model is closely related to a design methodology, rather than just for providing a proof of security. The model consists of three parts: the unauthenticated-links model, the authenticated-links model, and the ideal key exchange process. Each model is associated with an adversary with different abilities. To prove that a protocol

is secure, the ultimate goal is to show that the adversary in the unauthenticated-links model cannot do much more than an ideal-KE-adversary. This so-called a modular approach introduces the concept of authenticators and MT-authenticators, which can transform a secure protocol in the authenticated setting into a secure protocol in the unauthenticated setting. [BCK98] already proposes and proves two valid authenticators, then any protocol which can be proven to be a secure key exchange protocol under the authenticated links model, can then be applied with an authenticator to be secure in the unauthenticated-links model. Again, proving a protocol's security in the AM model requires showing that the AM-adversary cannot do much more than an ideal-KE-adversary.

From the above procedure of proving security, the main job of proving in [BCK98]'s model is really proving that a protocol is secure in the authenticated setting. Unlike the techniques used in [BWM97] and [BR93B], this model seems to be less work than the other two.

[BCK98] proved that the key transport protocol using encryption and the Diffie-Hellman key exchange are both secure protocols in the authenticated links model. By further applying authenticators to these two protocols, we get a secure authenticated key transport protocol and a secure key exchange protocol. This also shows that the models described in [BCK98] have a greater application in formal modelling, not just restricted to the two-party key exchange scenario.

For further discussion on this model, the readers can refer to Section 6.2 or the original paper.

Chapter 5

Desired Features of Authenticated Key Exchange Protocols through a Case Study of the Station-to-Station(STS) protocol

In Chapter 1, the importance of an AKE protocol is already emphasized. The following questions to ask would be "What are the things that needed to be included in an AKE protocol in order for it to be secure?" and "How does one go about creating a secure AKE protocol?". The following two chapters are intended to answer these two questions. The next chapter will focus on how to create a secure AKE protocol; this chapter instead focuses on the features that should be taken consideration into when one designs an AKE protocol. This is done via a case study of the Station-to-Station (STS) protocol. STS has been under development for a long period of time and has been adopted in many industrial standards. From its original form to the present accepted one, flaws of it have been found and fixes presented; it serves as a good example of some common mistakes protocol designers have made over the past years. It is not guaranteed that studying STS may reveal all the desired features in an AKE protocol. However, examining STS in detail can nevertheless help future protocol designers to understand which issues they should keep in mind.

The rest of this chapter is structured as follows: a quick introduction of different forms of key establishment is presented, followed by a brief history of STS. Then starting from the original protocol, I will demonstrate different attacks on STS and the different fixes of it. Along with the illustration of attacks and fixes, discussions

and comments will also be presented. By the end of this chapter, a secure version with all the desired properties of STS is given. The proof of security of the final version, however, is included in the next chapter together with the discussion of design methodologies.

5.1 Key Establishment Protocols

The motivation of key establishment can come from various scenarios. It can be considered in the PKI setting, where public key encryption is more computationally expensive than secret key schemes, and if two entities wish to communicate with each other, than they need to first apply public key cryptography to establish a shared key k, and then encrypt any subsequent messages using secret key encryption schemes with key k. Another scenario is the case where a client and a server share a common secret to start with (say the client's password). When the client wishes to perform banking activities at the ATM, a good practice is that once the client has authenticated itself, then a session key must be established between the ATM and the server to encrypt all transactions, instead of just using the client's password as the session key. Key establishment plays a small part of a cryptosystem, but it should certainly receive as much attention as other parts such as encryption, signatures, etc.

Depending on the setting of the computation, key establishment can exist in many forms. Key establishment can be in the symmetric setting where two parties share a common secret to start with, the asymmetric setting (like the traditional PKI which uses certificates); the key can be established between two parties or between multiple entities of a group, and key exchange can be based on each person's identity or his/her password (e.g., ID or password-based key establishment). Examples of secret-key and public-key based key establishment have been given in the previous paragraph. Key establishment can also exist in an unauthenticated (anonymity enabled) and authenticated (communication to only registered and certified members) manner. In this chapter and the rest of the thesis, the setting is restricted to the scenario where the key establishment is between only two parties, in the asymmetric setting.

Having the notion of security and the setting of computation, another problem faced in designing an authenticated key exchange protocol is the choice of features included or implemented in the actual protocol. For example, while not sacrificing the security a protocol certainly one wishes to achieve minimal network flow overhead possible. Specific to each design, a feature that can make one protocol secure

by including it may not make other protocols secure.

5.1.1 Key Transport v.s. Key Exchange Protocols

A key transport protocol is a two-user protocol, in which one entity dominates the key control ability. The other party of the communication has to rely completely on the key offering party. Many existing systems, such as SSH and Kerboros are or have key transport protocols. This technique is especially useful for the case where one party has only limited computing power and cannot afford expensive computations to participate in key establishment. Another suitable scenario is if one of the entities is a server granting access to anonymous users. Then the server may generate keys itself and force the users to use the keys in order to protect itself from possible security problems.

5.2 STS: a Brief History

The Station-to-Station AKE Protocol was proposed by [DOW92]. The original version of STS (which we will refer to it as STS-ENC) proposes the following properties:

- Implicit key authentication: at the end of a successful protocol run (in which both Alice and Bob accept), each party is sure that the other party has sufficient material to compute the same established session key sk.
- Unknown-Key-Share (UKS) Resilience using encryption. UKS is the event such that at the end of a protocol run where the adversary is actively present, without loss of generality, Alice believes that she has communicated and established a session key with Bob, but Bob instead believes that he has communicated and established with another entity, Oscar. UKS resilience is the prevention of such an event from happening.
- No cryptographic service provider on a particular secret key by each entity performing cryptographic operations on messages (signatures, encryption on signatures) containing its own input (its own random element in the group or challenge).
- Joint key control: the security of the key does not rely completely on only one participant of the protocol; both Alice and Bob have to input their "share" to establish the key.

- (Entity) authentication (preventing the man-in-the-middle attacks) by providing signatures.
- Perfect forward secrecy using ephemeral keys. Perfect forward secrecy (perfect FS) means that, the exposure of the long term keying material does not allow the adversary to retrieve any information about any previously established session keys.
- Preventing key-compromise impersonation (KCI) by signature schemes. KCI is the event when a participant A of the protocol loses her or his long-term keying information, and then others can impersonate themselves to A.
- No timestamps. Timestamps are marks or indicators specifying the "expiry time" of a particular message. A timestamp can be used to ensure message freshness (to prevent attacks using previous legitimate messages). However, due to the difficulty of synchronization between different CPU clocks, and the right duration of the "expiry time", some of researchers do not recommend the usage of timestamps in AKE protocols.
- Security of certificates: should include the Diffie-Hellman parameters - generator α and prime order p.

Ever since the publication of the protocol, much analysis has been done to discover the possible flaws; some results suggest new, improved versions of it. In particular, [DOW92] suggests that the symmetric encryption is essential in the original proposal of the STS protocol. [BM] suggests that MACs should be used instead of encryptions to provide data integrity. In [BW99] and in [BWM99], the unknown key share attacks (UKS) have been proposed to attack STS. While many argue that the UKS attack is not serious, I believe that it does model threats possible in real life. Also, in [BWM99] Blake-Wilson and Menezes proposed a variation of STS known as the STS-MAC, as well as some countermeasures to the UKS attacks. Baek and Kim in [BK00], however, showed that the countermeasures proposed in [BWM99] may not work if the underlying signature scheme is not secure. [BCK98] suggests another variation, which will be referred to as STS-BCK, and gave a security proof of it under the assumption that the signature scheme is secure.

In later sections, I will describe the three variations of STS, and demonstrate how the features of STS can achieve its desired properties by showing attacks to the protocol when certain features are removed. The following section will be on UKS, the only known attacks to STS. Two types of UKS will be described and proposed

countermeasures are surveyed. In the third section two proofs of STS's security are given: one is from [BCK98] and another one is developed independently from the proof given in [BCK98]. The last subsection will be some remarks on the STS protocol.

5.3 Case Study of STS

Before describing the protocol, it is useful to clarify the notations that will be used for the rest of this chapter. Due to complex details of the STS protocol, the assumptions of the protocols should also be mentioned before the description.

5.3.1 Notation

- Alice and Bob both have a set of keys for different purposes. For any cryptographic tools that takes Alice's key as input (e.g. keyed MAC), then the notation for it is the symbol of the cryptographic tool with subscript A. For example, applying a MAC with Alice's MAC key on a message m will be denoted as $\texttt{MAC}_{\mathrm{A}}(m)$. Similarly, $\texttt{MAC}_{\mathrm{B}}(m)$ means applying MAC with Bob's MAC key on message m.
- If Alice and Bob wish to run STS to establish a session key at the end of the protocol run, the developed session key is denoted as sk_{AB}. Note that the order of subscripts is irrelevant. That is, $sk_{\mathrm{AB}} = sk_{\mathrm{BA}}$. If by context we already know which two users are supposed to share the session key, then a simpler notation of sk is used to represent the developed session key at the end of a protocol run.
- The certificates of Alice and Bob are denoted as C_{A} and C_{B}, respectively.
- For simplicity, sometimes it is necessary to distinguish a random number generated by one party from another. The random number generated by Alice and Bob are denoted r_{A} and r_{B}, respectively.
- The generator of a group is denoted as α. The cyclic group generated by the generater is denoted as $\langle\alpha\rangle$.
- A $\rightarrow$ B denotes that Alice sends a message to Bob.
- The signature of Alice on a particular message m is $sig_{\mathrm{A}}(m)$. Similarly for Bob's case.

- Encrypting a message m using Alice's private encryption key is denoted as $E_{\mathrm{A}}(m)$. Similarly for Bob's case.
- The verification algorithm, using Alice's public verification key, is denoted as $ver_{\mathrm{A}}(m, s)$, where m is the message, and s represents the signature.
- The hash function on messages is denoted as $H(\cdot)$. However, notice that in order to produce a signature on a long message m, a hash function is usually used before the m is sent to the signature scheme. Therefore, the complete way of hashing m and signing it (with signing key k) is $sig_k(H(m))$. For simplicity of describing the protocol, whenever a hash is needed for signing purposes, we ignore writing out the hashing process unless otherwise specified. Similarly, when verifying the message m with a corresponding signature s, the verification algorithm has to take the hashed m as input instead of m itself. Again this extra notation is ignored.
- The MAC, using key k, is denoted as $\mathtt{MAC}_k(\cdot)$.

5.3.2 The Original Protocol

The STS protocol is an enhanced form of the Diffie-Hellman protocol. The original protocol in [DOW92] is as follows:

1. Alice chooses a random number r_{A} from $\mathbb{Z}_q^*$, computes $R_{\mathrm{A}} = \alpha^{r_{\mathrm{A}}}$ then Alice→Bob: $m_1 = (\alpha, p, R_{\mathrm{A}})$.
2. Bob chooses a random number r_{B} from $\mathbb{Z}_q^*$, computes $R_{\mathrm{B}} = \alpha^{r_{\mathrm{B}}}$, $sk_{\mathrm{BA}} = (R_{\mathrm{A}})^{r_{\mathrm{B}}}$, and $S_{\mathrm{B}} = sig_{\mathrm{B}}(R_{\mathrm{B}}, R_{\mathrm{A}})$ then Bob→Alice: $m_2 = (R_{\mathrm{B}}, C_{\mathrm{B}}, E_{sk}(S_{\mathrm{B}}))$.
3. Upon receiving m_2, Alice checks whether $ver_{\mathrm{TA}}(C_{\mathrm{B}}) = true$ and extracts ver_{B} from C_{B}. Alice then verifies whether $ver_{\mathrm{B}}(S_{\mathrm{B}}, (R_{\mathrm{B}}, \alpha^{r_{\mathrm{A}}})) = true$. If so, Alice accepts and computes the session key $sk_{\mathrm{AB}} = sk = (R_{\mathrm{B}})^{r_{\mathrm{A}}}$. Alice then computes $S_{\mathrm{A}} = sig_{\mathrm{A}}(R_{\mathrm{A}}, R_{\mathrm{B}}))$ and Alice→Bob: $m_3 = (C_{\mathrm{B}}, S_{\mathrm{A}})$.
4. When Bob receives m_3 from Alice, he checks whether $ver_{\mathrm{TA}}(C_{\mathrm{A}}) = true$ and extracts ver_{A} from C_{A}. Bob then verifies whether $ver_{\mathrm{A}}(S_{\mathrm{A}}, (R_{\mathrm{A}}, \alpha^{r_{\mathrm{B}}})) = true$. If so, then Bob accepts. The protocol terminates at this point.

This is illustrated in Figure 5.1.

In [DOW92], it claimed that STS achieves the above desired properties. The explanation given in the paper is described below:

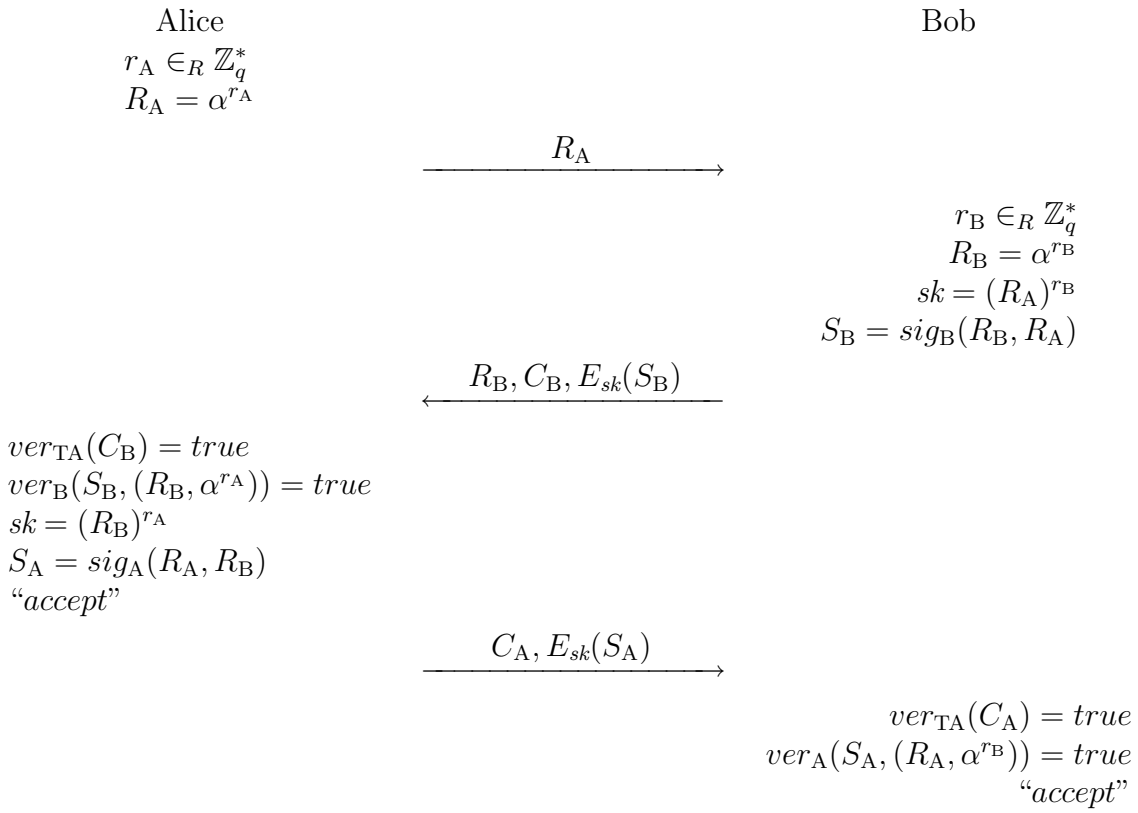

Figure 5.1: The Original STS Protocol

- No Timestamps: It is easy to see that instead of using timestamps, challenges are used for freshness.

- No Cryptographic Service Provider: the two cryptographic operations involved in this protocol are encryption and signatures. In Alice's case, she signs the content $(R_{\mathrm{A}}, R_{\mathrm{B}}) = (\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}})$. In this signature, r_{A} is in the content and is her own input, thus she is not signing something she has no control of. Furthermore, Alice encrypts her signature, which already has her own input, thus the content being encrypted is partially controlled by Alice as well. Similarly, Bob also signs and encrypts contents he has control of.

- Joint Key Control: clearly, the session key $sk_{\mathrm{AB}} = sk_{\mathrm{BA}} = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$ involves inputs from both participants in the protocol, and thus neither party can force the session key to be in a particular format. Of course, in order for an adversary to trivialize the session key, each participant should check that the value he/she receives does not equal to 0 and 1. Also, the participants should check that the values they receive is in the subgroup $\langle \alpha \rangle$. Normally in practice, α is carefully chosen so that α has order q, where q is a prime, and there is only one subgroup with order q. To verify that an element is in $\langle \alpha \rangle$, one raise the element to the power of q and see if it results in 1.

- Perfect Forward Secrecy: The long-term keying material is the secret key for each user's signature scheme. In particular, the session key $sk_{\mathrm{AB}} = sk_{\mathrm{BA}} = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$ is chosen independently from each user's secret key. Therefore, compromising the secret key of any user does not affect the security of the exchanged keys from earlier runs (due to the usage of Diffie-Hellman key exchange).

- Entity Authentication and Implicit Key Authentication: Bob ensures that, by Alice signing the random element $\alpha^{r_{\mathrm{A}}}$ in the third flow, the messages he receives in the first and third flow are from the same person. The certificate of Alice is sent in the third flow so that Bob knows it is Alice he is communicating with. This achieves authentication. Also, by seeing the encryption using the session key sk on Alice's signature, Bob ensures that Alice is able to compute the session key. The Diffie-Hellman key exchange mechanism ensures that no one other than Alice can compute the session key. From Alice's point of view, she verifies that it is Bob she is talking to by seeing Bob's certificate. Also, Bob has signed the content $(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}})$, in which $\alpha^{r_{\mathrm{A}}}$ is the value she sends in the first flow, and $\alpha^{r_{\mathrm{B}}}$ matches the value sent from Bob. This ensures that the signature from Bob is fresh, and Bob knows the session key. Again, Diffie-Hellman key exchange ensures that no one else can possibly compute the key.

Each feature in the protocol is claimed to be essential. This can be shown by removing a feature in the protocol and then demonstrating an attack on the protocol. Recall that the original STS consists of exchanging two random numbers, signatures on one's own exponential and the other party's exponential, encryption on the signatures, and authentication with key exchange. The content in the signatures is of the format

$$(< \textit{one's own exponent} >, < \textit{the other party's exponent} >).$$

An alternative to this format is to just sign the content in a particular order. For example, both Alice and Bob may sign the content $(\alpha^{r_B}, \alpha^{r_A})$ instead of one signing $(\alpha^{r_B}, \alpha^{r_A})$ and the other $(\alpha^{r_A}, \alpha^{r_B})$. I will refer to this property as the order of signing exponents. In the following paragraphs, I will remove each feature one at a time and then demonstrate an attack to emphasize the necessity of having that particular feature in the protocol.

5.3.3 Using Static Exponents Instead of Random Exponents

If the two parties use static exponents instead of random ones, then this pretty much means between Alice and Bob there is only one long-term shared key in this scenario, say $\alpha^{x_A x_B}$, where α^{x_A} is the static exponent of Alice and α^{x_B} is the static exponent of Bob. Note that x_A and x_B are long-term private keys of Alice and Bob since they serve the same purpose as if we choose any other long-term static keying materials to perform key exchange. Immediately we can see that forward secrecy is compromised. If one of x_A or x_B is known (WLOG we assume x_A is known to the adversary Oscar), then Oscar can compute the long-term shared key by computing $(R_B)^{x_A} = \alpha^{x_B x_A} = \alpha^{x_A x_B}$. Thus any previously computed shared keys, which are just $\alpha^{x_A x_B}$ are known to Oscar.

5.3.4 Removing Signing One's Own Exponential (i.e. Signing Only the Other Party's Exponential)

If each party is signing only the other party's exponential, the direct consequence is that the two participants become a cryptographic service provider since whatever is provided to the party, it is guaranteed to get her/him to sign it. [DOW92] indicates that the adversary can potentially preselect logarithms and request this

fixed quantity to be signed, but they could not suggest a general attack to this modification of protocol.

Note that in this scenario, although Bob signs whatever is sent to him, he does have control of how the signature should be encrypted by choosing his own exponents to influence the session key. Therefore, Bob's cryptographic service does not come for free. Oscar will still need to solve the Diffie-Hellman problem to be able to use the signature. Under the assumption that DHP is intractable, signing only the other party's signature with encryption under the session key does not give Oscar the ability of using the signature directly. Note that the encryption in this case helps data integrity and authentication in addition to using signatures. In today's practice, people tend to separate authentication, secrecy, and data integrity using separate different cryptographic services instead of nesting one in the other.

Although the encryption protects the direct usage of signature cryptographic service, there is still a concern about whether Oscar may use the signature indirectly to conduct a successful attack. From Bob's point of view, his goal in this protocol is to ensure that 1) the person who talks to him in the protocol is really that person, and 2) at the end of the protocol no one other than his intended participant is able to compute the session key. The initiator, Alice, sends her key material and signature for authentication in two different flows to demonstrate she knows the session key and she is the one who sends the first message. However, Alice does not get the same assurance this way. Bob sends both his key material AND signature at the same time, so Alice's only assurance is that 1) she has received an exponent from an entity, 2) she has received the certificate of Bob, 3) using the exponent she received, she can decrypt a received ciphertext and the decrypted message is Bob's signature on the exponent. What Alice is not sure is that whether Bob sends this exponent and the signature intended for her. However, since no attack taking this advantage of the protocol design has been found, such a concern is left as an open problem.

5.3.5 Removing Signing the Other Party's Exponential (i.e. Signing Only One's Own Exponential)

The immediate consequence of signing only one's own exponential is the loss of freshness and hence the protocol is vulnerable to a potential message-reply attack. However, the freshness is again ensured by including the encryption using the established session key sk so that both parties know the signature being encrypted is bounded to this particular protocol run and is not a replayed message.

[DOW92] suggested that no general attack was known during the time of the publication of [DOW92]; the only known attack was a special case when the signature scheme is RSA, the hash function (used on a message, then sends the message to the signature scheme) is the identity function, and the Diffie-Hellman key exchange is carried out over $GF(p)$, where p is a prime. Oscar can impersonate Alice in a run with Bob by using α^0 as the exponential in the key exchange. Now Oscar's exponential is $\alpha^0 = 1$, and the exchanged session key $sk = \alpha^{0*r_B} = 1$. In order to impersonate Alice, Oscar needs to compute $E_{sk}(sig_A(H(\alpha^0))) = E_1(sig_A(1))$ (since the hash function is the identity function) $= E_1(1)$ (due to RSA signature scheme). Now $E_1(1)$ is easy to compute and Oscar can perform the following attack:

1. O(A) $\rightarrow$ B: $\alpha^0 = 1$
2. Bob computes $sk = 1$ regardless what his exponential is. Hence B $\rightarrow$ O(A): $\alpha^{r_B}, C_B, E_1(sig_B(\alpha^{r_B}))$
3. O(A) $\rightarrow$ B: $(C_A, E_1(sig_A(1))) = (C_A, E_1(1))$
4. Bob accepts, Oscar has successfully impersonated Alice.

Note that, regardless of the special scenario shown in the above attack, any cryptographic protocols nowadays require that the recipient check that the exponential he/she receives is not equal to 1 and that the generator α to the power of the exponent has to lie in the cyclic subgroup $\langle\alpha\rangle$. The attack is illustrated in Figure 5.2.

5.3.6 Uncoupling Authentication from Key Exchange

If the STS protocol is modified such that each participant signs the content that is independent of the exponentials, then it is vulnerable to the man-in-the-middle attack ([RS84]). Suppose that Alice (the initiator) has to sign message M_3 in flow 3 and Bob the responder has to sign message M_2 in flow 2. Both M_2 and M_3 are independent of the exponentials. The man-in-the-middle attack works as follows:

1. A $\rightarrow$ O(B): α^{r_A}, where $r_A \in_R \mathbb{Z}_q^*$
2. O(A) $\rightarrow$ B: $\alpha^{r_{O1}}$, where r_{O1} is a number picked by Oscar himself.
3. B $\rightarrow$ O(A): $\alpha^{r_B}, C_B, E_{sk_{BO}}(sig_B(M_2))$, where $sk_{BO} = \alpha^{r_B r_{O1}}$.

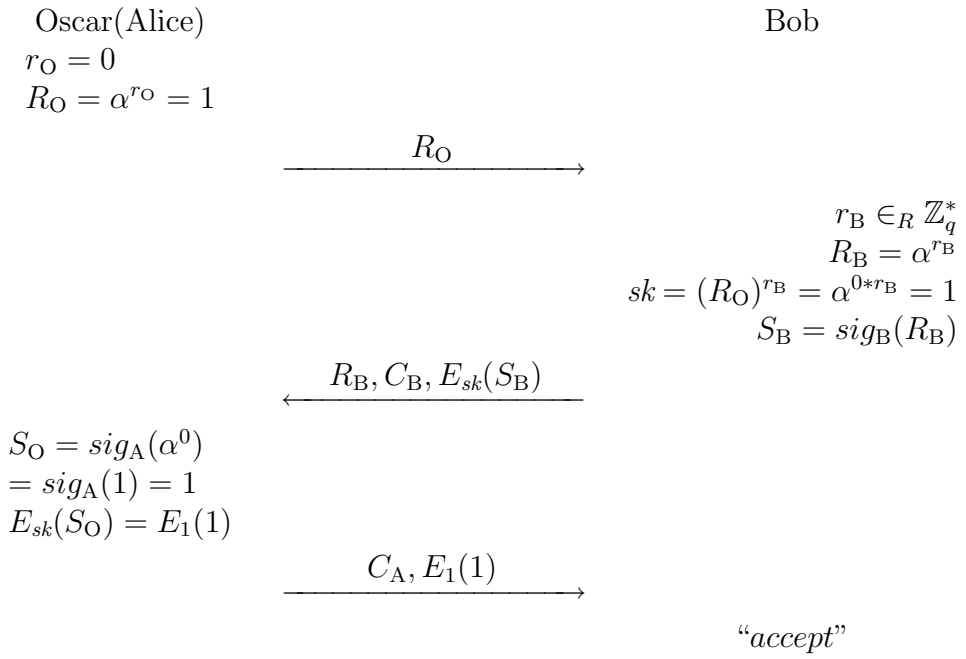

Figure 5.2: Impersonation Attack to STS with Signing One's Own Exponential

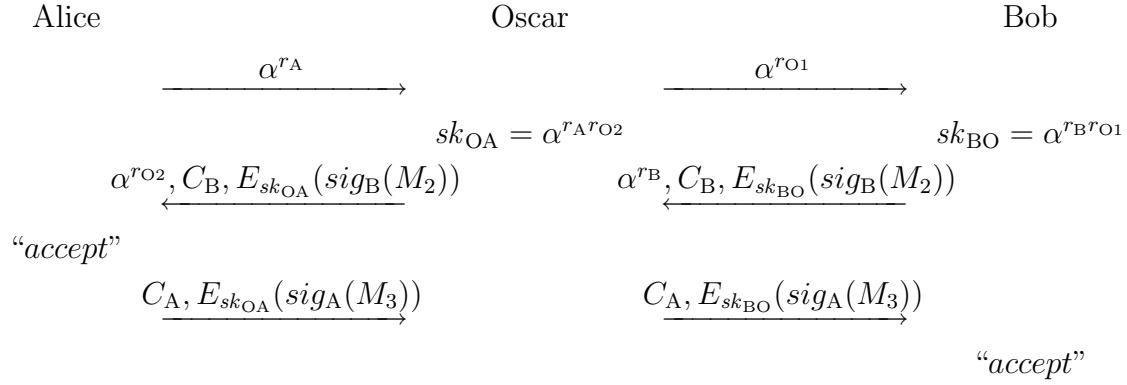

Figure 5.3: Man-in-the-Middle Attack on STS Uncoupling Authentication from Key Exchange

4. Note that Oscar can decrypt the message and retrieve the raw signature $sig_B(M_2)$ since the message M_2 is independent of the exchanged exponentials. Oscar then can encrypt the signature again using his session key with Alice. Thus, O(B) $\rightarrow$ A: $\alpha^{r_{O2}}, C_B, E_{sk_{OA}}(sig_B(M_2))$, where $sk_{OA} = \alpha^{r_A r_{O2}}$.

5. Alice accepts and then A $\rightarrow$ O(B): $C_A, E_{sk_{OA}}(sig_A(M_3))$

6. Oscar now can decrypt $E_{sk_{OA}}(sig_A(M_3))$ and retrieve the raw signature $sig_A(M_3)$. He then encrypt the signature using sk_{BO} and O(A) $\rightarrow$ A: $C_A, E_{sk_{BO}}(sig_A(M_3))$. Upon receiving this message, Bob accepts.

In the above scenario, Oscar substitutes his own exponentials for Alice's and Bob's exponentials. This action results in Oscar sharing session keys with both Alice and Bob while Alice believes that Oscar is Bob and Bob believes Oscar is Alice. Within the authentication part of the protocol, Oscar can pass on Bob's signature to Alice by decrypting Bob's signature using their shared key sk_{BO} and then encrypt the signature using the key he shares with Alice, sk_{OA}. Similarly, Oscar can decrypt Alice's signature then encrypt it again then send it to Bob.

5.3.7 Changing the Order of Signing Exponentials

This scenario was not considered in [DOW92] but was pointed out in [Sho99]. By making both parties sign the same content, the protocol is vulnerable to an

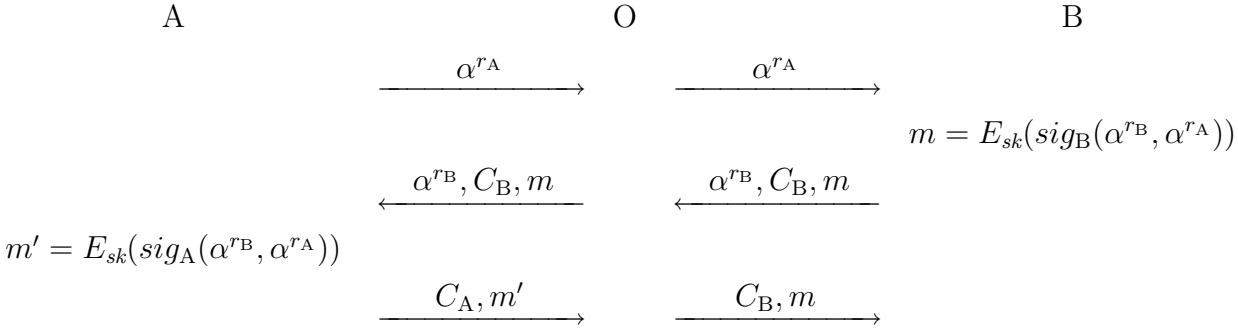

Figure 5.4: Interleaving Attack on STS with Order of Signing Exponential Change

interleaving attack. In this scenario, the attacker Oscar can take the encrypted signature output by Bob in the second flow, relay it to Alice (in the second flow) and feed it back to Bob in the third flow. Thus, both Alice and Bob accept, and Alice thinks she is talking to Bob while Bob thinks he is talking to an instance of himself. To demonstrate the attack,

1. A $\rightarrow$ O(B): α^{r_A}
2. O $\rightarrow$ B: α^{r_A}
3. O $\leftarrow$ B: $\alpha^{r_B}, C_B, m = E_{sk}(sig_B(\alpha^{r_B}, \alpha^{r_A}))$
4. A $\leftarrow$ O(B): $\alpha^{r_B}, C_B, m = E_{sk}(sig_B(\alpha^{r_B}, \alpha^{r_A}))$
5. A $\rightarrow$ O(B): $C_A, m' = E_{sk}(sig_A(\alpha^{r_B}, \alpha^{r_A}))$
6. O $\rightarrow$ B: $C_B, m = E_{sk}(sig_B(\alpha^{r_B}, \alpha^{r_A}))$

An illustration is shown in Figure 5.4.

The effect of this attack relates to unknown key share. In the above attack both Alice and Bob accepts at the end of the protocol run. Alice has the right belief she has exchanged a session key with Bob, and no one other than Bob may compute the key. Bob, however, believes that he has exchanged a session key with an instance of himself; no one but then himself can compute the key. It is an attack, although it is not clear what subsequent damage Oscar can do to benefit himself.

5.3.8 The Diversity: STS-MAC and STS-BCK

There are two other versions of STS published in the literature. One is suggested in [BWM99] and this version which uses a MAC in replacement of the encryption scheme. The other in [BCK98], by Bellare, Canatti, and Krawczyk, applies key-exchange based on an *authenticator* for authentication. These two variations are explained in the following. Note that in order to sign a particular message m, m is always hashed first before being signed. Therefore, the following protocol illustrations will omit the presence of a hash function, but the reader should keep in mind that the message being signed is hashed before applying the signature scheme.

The STS-MAC is described as follows (illustration shown in Figure 5.5):

1. A $\rightarrow$ B: $\alpha^{r_{\mathrm{A}}}$
2. A $\leftarrow$ B: $C_{\mathrm{B}}, \alpha^{r_{\mathrm{B}}}, sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$
3. A $\rightarrow$ B : $C_{\mathrm{A}}, sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

Here $sk = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$ is the session key established after a successful protocol run.

Note: In STS-MAC, each participant sends their signature without encryption and adds an additional MAC value of the signature with the established key *sk*. On the contrary, in STS-ENC the participants do not have to send an extra item in flow 2 and 3, but their signatures are encrypted with *sk*.

STS-BCK is described as follows:

1. A $\rightarrow$ B: $\alpha^{r_{\mathrm{A}}}$
2. A $\leftarrow$ B: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}, ID(\mathrm{A}))$
3. A $\rightarrow$ B: $sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}, ID(\mathrm{B}))$

An illustration of STS-BCK is given in Figure 5.6.

Note that STS-BCK uses only one type of cryptographic service, namely signatures, as compared to STS-ENC or STS-MAC, which use an additional encryption or MAC in the protocol. Also, the established session key in STS-BCK is never used in any protocol flow. STS-ENC and STS-MAC protocols, however, use the session key encrypt/MAC signatures as part of the outgoing message. The lack

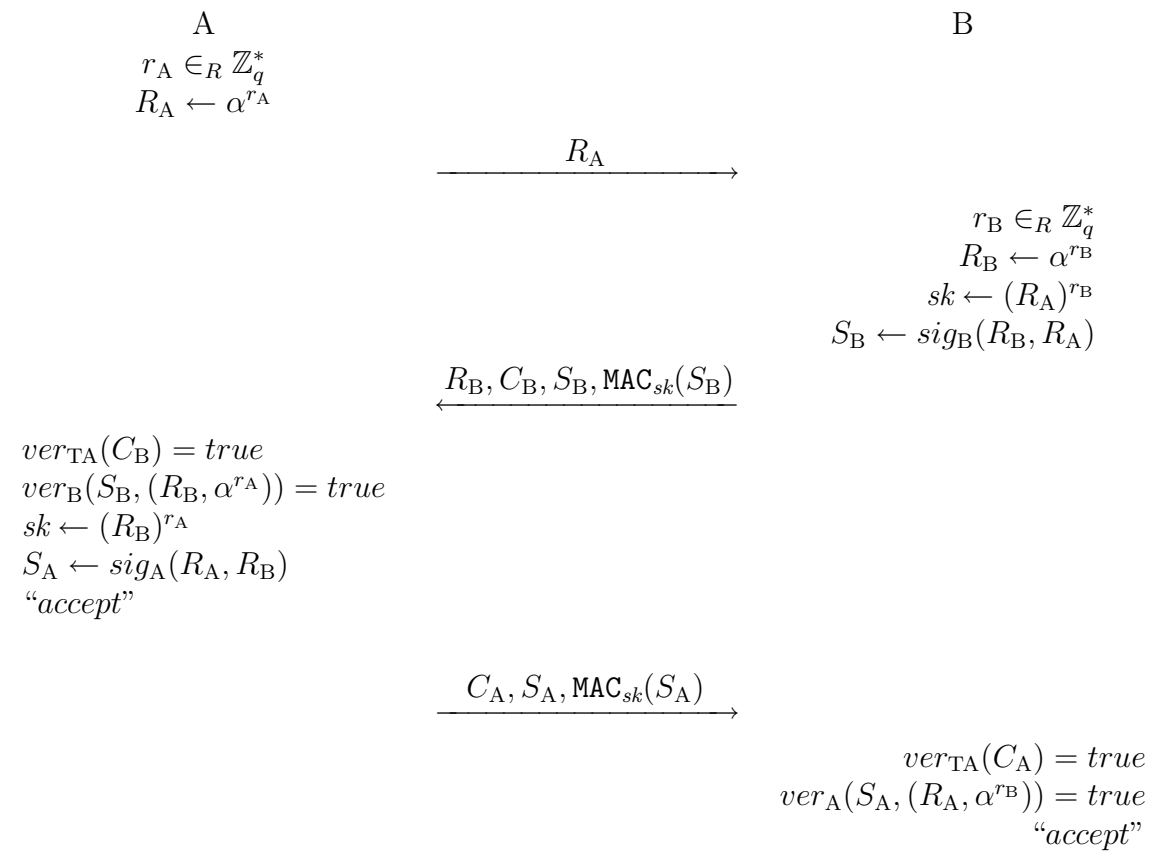

Figure 5.5: The STS-MAC Protocol

of applying an encryption or a MAC also implies that STS-BCK provides only implicit key confirmation; while STS-ENC and STS-MAC both provides explicit key confirmation. Another difference is that STS-BCK includes the identity of the intended recipient in the outgoing signature and the other two do not.

From the literature, some papers would describe the above protocols with the addition of an $ID(\mathrm{A})$ in flow 1 and $ID(\mathrm{B})$ in flow 2. In the figures the ID's are omitted and will be discussed later to emphasize the importance.

5.3.9 Lowe's Attack

Before the publication of the model by Blake-Wilson, Johnson, and Menezes, Lowe published [Lowe96] and demonstrated an attack on STS-ENC with his own defini-

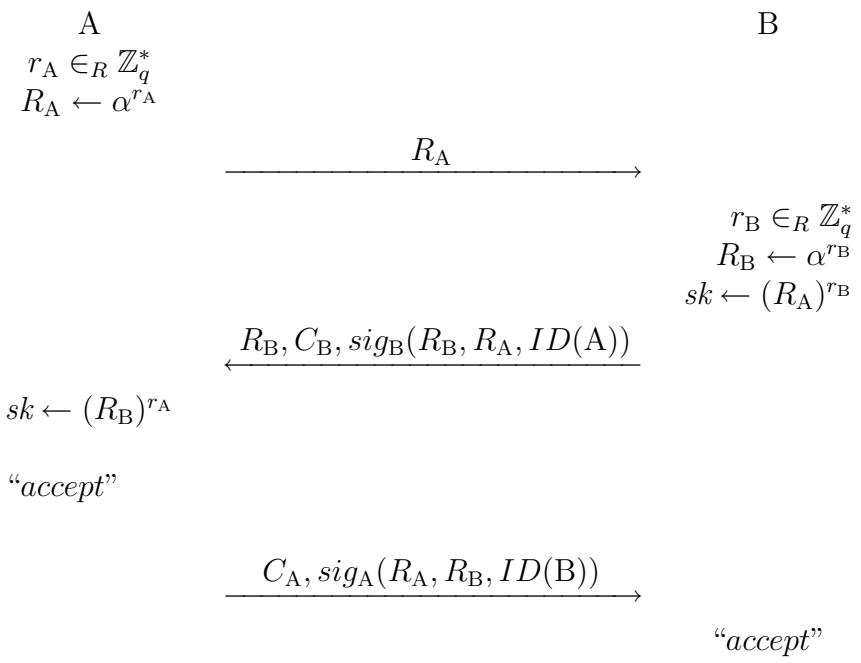

Figure 5.6: The STS-BCK Protocol

tion of authentication. Many subsequent papers in the literature have doubted the validity of Lowe's attack, but this attack foreshadows the unknown key share attack, as well as suggesting the importance of unknown key-share resilience (although I would prefer to call it mutual belief of identity).

Lowe's definition of authentication in [Lowe96] is quoted *"Whenever an agent A completes a run in the protocol, apparently with B, then B has recently been running the protocol, apparently with A, and the two agents agree upon who initiated the run, and agree upon all data values used in the run; further, there is a one-one relationship between the runs of A and the runs of B."* In other words, two points are made:

- if user A accepts at the end of the protocol, if A's intended peer in the protocol is B, then B's intended peer must be A. If A believes she is the initiator, then B must believe that A is the initiator. Similar for the case of A believing herself as the responder.
- At the end of the protocol A and B have a matching conversation.

The idea of Lowe's attack is essentially the same as unknown key share: At the end of the protocol run, Alice believes she is communicating and has established a

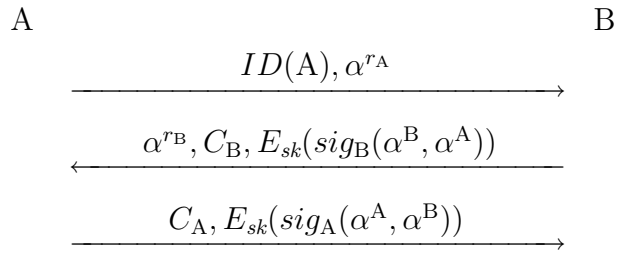

Figure 5.7: STS-ENCi, STS-ENC with ID(A) in the first flow

key with Bob, while Bob is not aware that he is talking to Alice. The controversy of Lowe's attack is that in this attack Oscar can only impersonate Bob to Alice (thus Alice accepts), but then Bob does not accept in the protocol. Alice believes that she shares a key with Bob but Bob does not have a successful run at all.

To conduct the attack, Lowe used a modified version of STS-ENC where in the first flow, Alice sends not only her randomly-chosen exponential α^{r_A} but also her identity $ID(\mathrm{A})$. For convenience this modification will be referred to as the STS-ENCi (see Figure 5.7.

The attack is described as follows:

1. A $\rightarrow$ O(B): $ID(\mathrm{A}), \alpha^{r_\mathrm{A}}$
2. O $\rightarrow$ B: $ID(\mathrm{O}), \alpha^{r_\mathrm{A}}$
3. O $\leftarrow$ B: $\alpha^{r_\mathrm{B}}, C_\mathrm{B}, E_{sk}(sig_\mathrm{Bob}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}))$
4. A $\leftarrow$ O(B): $\alpha^{r_\mathrm{B}}, C_\mathrm{B}, E_{sk}(sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}))$
5. A accepts. A $\rightarrow$ O(B): $C_\mathrm{A}, E_{sk}(sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}))$. Oscar drops the last message to Bob, and Bob never accepts.

Without the inclusion of $ID(\mathrm{A})$, this attack is essentially relaying messages back and forth (except for the last message from Alice to Bob, which Oscar just drops it). So what is the difference?

Intuitively, Alice accepts and all her beliefs are correct (1. Bob has identified himself to Alice, 2. she has established a key with her intended participant Bob, and Bob has all the information to compute the established key, 3. no one other than Bob or herself can compute the key). From Bob's point of view, however, he thinks he has an unsuccessful run with Oscar, and he does not accept at the end

of the protocol. Oscar also does not know any information about the established session key, so even though Alice thinks she is talking to Bob and in fact she is talking to Oscar, Oscar does not seem to be able to use any subsequent message from Alice (under the encryption using the established key) to benefit himself. Also, this type of scenario is still covered in the scope of a secure authenticated key exchange protocol (but not a secure authenticated key exchange protocol with key confirmation) under the definition of [BWJM97].

However, [Mao] argues that with the inclusion of $ID(\mathrm{A})$ in the first flow, at the time where Alice accepts, the recorded messages of Alice does not match the recorded messages kept by Bob. This violates the definition of authentication of [DOW92] and [Lowe96]. Also, consider the following scenario: Alice is a server providing services to requests from legitimate users. When a user Bob requests a service, Alice initiates an AKE protocol to make Bob identify himself to her and to exchange a session key. Once accepted in flow 2, Alice reserves some resources in preparation to provide service to Bob. Once the protocol is successfully completed, Alice then provides the service in encrypted form. If Oscar simply drops the last message from Alice to Bob, then Bob would send Alice a *not-acknowledged* (NAK) message and the protocol run may be unsuccessful. However, if Bob is thinking he is talking to Oscar, then Bob's NAK message would be directed to Oscar instead of Alice. However, Alice has now accepted Bob's identity and is preparing resources to service Bob; nobody will notify Alice of any abnormality. If Oscar applies this attack with multiple instances, Alice's capacity of serving other end users may drastically drop and may even power down. This is an instance of perfect denial of service attack (perfect DoS, [Mao]). The difference between a perfect DoS and the conventional DoS is that Oscar is not required to provide a certificate in a perfect DoS and the cost of Oscar is very small compared to Alice.

Side Discussion

The difference between the original STS-ENC and STS-ENCi is that in STS-ENC, Bob does not know who his intended peer is until the third flow (by looking at the certificate). If Bob does not receive the third flow, then the only way he knows who to send a NAK message to is through IP addresses or he may even not send any messages at all. In the former case, the perfect DoS is still possible since Alice will wait for a response from Bob, but in the later scenario, the perfect DoS seems not possible. Comparatively, STS-ENCi ensures that Bob knows his intended participant in the first flow, and thus knows who to send a NAK message to if he does not receive the third flow.

5.3.10 The Unknown Key Share Attack

In 1999, Blake-Wilson and Menezes introduced the unknown key share attack to STS-ENC and STS-MAC. The idea is to make one party, say Alice, believe that at the end of the protocol run, she is indeed communicating with Bob and no one other than Bob can compute the session key *sk*. However, Alice's intended participant, Bob, believes that he is communicating with the adversary Oscar and no one but Oscar can compute the session key *sk*. Note that the session key Alice computes is the same as the session key Bob has. This attack can be conducted against either the initiator or the responder. In [BWM99] Blake-Wilson and Menezes suggest several possible countermeasures including public key validation (proof of possession of corresponding private key), enforced exchange of certificates before start of protocol, and adding the identity of the intended recipient, the identity of the sender, and the flow number in the signature. In [BK00], Baek and Kim demonstrate that the UKS attack is still possible after including the ID of the intended recipient, ID of the sender, and the flow number in the signature if the signature scheme used in STS is either ElGamal or RSA. Baek and Kim's paper suggest a potential need to revise the definition of a secure signature scheme for data integrity and authentication. Similar to Lowe's attack, there are concerns about whether UKS constitutes a valid attack, how realistic it can be, and how serious the damage could be in real life.

To show the possibility of UKS damages in practice, [BWM99] suggests two hypothetical scenarios that might happen in real life:

Scenario A: Suppose that Bob is a bank server and Alice is an account holder. An electronic deposit system uses an authenticated key exchange protocol (WLOG, say STS) to establish a share key between Alice and Bob. At the end of protocol Alice then uses the shared key to encrypt the amount of money for deposit without further authentication. If Oscar conducts the UKS attack so that Bob the bank thinks his communication session is with Oscar, then when Alice transmitted the encrypted amount to Bob, the money will be deposited into Oscar's account instead of hers.

Scenario B: Suppose Bob controls access to a suite of sensitive applications (e.g. account information, salary database, etc.). Each application is associated with a password. A trusted authority TA distributes certificate and public keys to all the users who may use the applications. The TA also chooses the passwords for each application and distributes them securely to each user eligible or entitled to use a certain application. When a user Alice wishes to use an application, she starts an authenticated key exchange protocol with Bob. At the end of the protocol run, she then encrypts using the established session key the password associated with

the application she wishes to have access to. Bob decrypts the message from Alice and check if the password is valid. If so, then access is granted to Alice. Suppose Oscar performs the UKS attack so that Alice thinks she's exchanged keys with Bob and Bob thinks he's exchanged keys with Oscar, then Bob will treat Alice's subsequent encrypted message as coming from Oscar. If Alice's password is valid, then the access to this particular application will be granted to Oscar but not Alice.

There are two types of UKS attack described in [BWM99]; I will refer to them as the public key substitution UKS (PKS-UKS) and the duplicate-signature key selection UKS (DSKS-UKS) for ease of reference and consistency with [BK00]. Regarding the possibility of UKS attacks happening in real life, [BWM99] has stated the assumptions essential for UKS to be carried out.

To perform public key substitution UKS, the adversary must be able to register public keys of his own choice. Also, the TA should also allow duplicate public key values. In order to carry out DSKS-UKS, two other assumptions have to be made:

1. The signature scheme used in STS has the duplicate-signature key selection property. That is, given Alice's public key P_{A}, a message M, and the corresponding signature s_{A}, it is possible for Oscar to find a different public/private key pair $(P_{\mathrm{O}}, S_{\mathrm{O}})$, where $P_{\mathrm{O}} \neq P_{\mathrm{A}}$, such that s_{A} is also Oscar's signature on M. Protocols that propose this property include RSA, Rabin, ElGamal, DSA, and ECDSA signature schemes.

2. The adversary Oscar can get his public key certified during a protocol run. This situation is possible in situations where delays in the transmission of messages are normal, and the TA is online to provide certification services.

The idea of PKS-UKS is as follows: when Alice sends Bob the messages (ID(A), $\alpha^{r_{\mathrm{A}}}$), the adversary Oscar simply replaces ID(A) with ID(O). In the second flow, Oscar just forwards Bob's message to Alice, although Bob's intended participant is Oscar. When Alice sends the message in flow 3, Oscar replaces Alice's certificate with his own, and sends to Bob. Here Bob is the entity being led to false beliefs and the attack is said to be against the responder. If the UKS is carried out such that Alice is led to false beliefs instead of Bob, then it is an attack against the initiator.

PKS-UKS Attack on STS-ENCi Against the Responder

1. A $\rightarrow$ O(B): $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$
2. O $\rightarrow$ B: $ID(\mathrm{O}), \alpha^{r_{\mathrm{A}}}$

3. O $\leftarrow$ B: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, E_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$

4. A $\leftarrow$ O(B): $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, E_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$

5. A $\rightarrow$ O(B): $C_{\mathrm{A}}, E_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

6. O $\rightarrow$ B: $C_{\mathrm{O}}, E_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

Note that Oscar registers his public key as Alice's and thus $sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}) = sig_{\mathrm{O}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}})$. When Bob verifies the signature in flow 3, the signature is valid and represents Oscar's identity; Bob accepts.

PKS-UKS Attack on STS-ENCi Against the Initiator

In this scenario, Oscar registers his public key the same as Bob's (so to be against Alice the initiator). The attack is described below:

1. A $\rightarrow$ O: $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$

2. O(A) $\rightarrow$ B: $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$

3. O(A) $\leftarrow$ B: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, E_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$

4. A $\leftarrow$ O: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{O}}, E_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$

5. A $\rightarrow$ O: $C_{\mathrm{A}}, E_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

6. O(A) $\rightarrow$ B: $C_{\mathrm{A}}, E_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

Again, Oscar registers his public key the same as Bob's, then $sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}) = sig_{\mathrm{O}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}})$. When Alice verifies it in flow 2, Alice accepts Oscar's identity.

Similarly, the PKS-UKS attack can be applied to STS-MAC. Here we use a modification of STS-MAC, adding $ID(\mathrm{A})$ in the first flow and this modification will be referred to as STS-MACi. Notice that PKS-UKS is also possible to be applied to STS-MAC.

PKS-UKS Attack on STS-MACi Against the Responder

1. A $\rightarrow$ O(B): $ID(\mathrm{A}), \alpha^{r_\mathrm{A}}$
2. O $\rightarrow$ B: $ID(\mathrm{O}), \alpha^{r_\mathrm{A}}$
3. O $\leftarrow$ B: $\alpha^{r_\mathrm{B}}, C_\mathrm{B}, sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}), \mathtt{MAC}_{sk}(sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}))$
4. A $\leftarrow$ O(B): $\alpha^{r_\mathrm{B}}, C_\mathrm{B}, sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}), \mathtt{MAC}_{sk}(sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}))$
5. A $\rightarrow$ O(B): $C_\mathrm{A}, sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}), \mathtt{MAC}_{sk}(sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}))$
6. O $\rightarrow$ B: $C_\mathrm{O}, sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}), \mathtt{MAC}_{sk}(sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}))$

PKS-UKS Attack on STS-MACi Against the Initiator

1. A $\rightarrow$ O: $ID(\mathrm{A}), \alpha^{r_\mathrm{A}}$
2. O(A) $\rightarrow$ B: $ID(\mathrm{A}), \alpha^{r_\mathrm{A}}$
3. O(A) $\leftarrow$ B: $\alpha^{r_\mathrm{B}}, C_\mathrm{B}, sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}), \mathtt{MAC}_{sk}(sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}))$
4. A $\leftarrow$ O: $\alpha^{r_\mathrm{B}}, C_\mathrm{O}, sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}), \mathtt{MAC}_{sk}(sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}))$
5. A $\rightarrow$ O: $C_\mathrm{A}, sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}), \mathtt{MAC}_{sk}(sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}))$
6. O(A) $\rightarrow$ B: $C_\mathrm{O}, sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}), \mathtt{MAC}_{sk}(sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}))$

Note that due to this attack, both STS-ENC and STS-MAC fail to achieve the definition in [BWJM97] of a secure AKC protocol. However, both STS-ENC and STS-MAC are still secure AKI protocols under [BWJM97]'s definition.

This attack is possible due to the fact that Oscar can choose duplicate values for his public key and register it even without knowing what the corresponding value is. One way to prevent this attack from happening is to force each public key to be distinct. However this mechanism seems very complicated and does not seem to scale in terms of key management. [BWM99] then emphasize the essence of public key validation (or proof of possession) - when an user registers a public key, the TA must force the user to prove (in zero knowledge, perhaps) his/her knowledge of the corresponding private key before issuing his/her certificate.

The duplicate signature key selection UKS (DSKS-UKS) attack is based on the assumptions that the certification authority is always online and a participant can

register his/her public key during a protocol run. This type of attack bypasses the public key validation process and implies to be a stronger attack than PKS-UKS.

The idea behind DSKS-UKS is that given Alice's message m and her signature on m, Oscar can select a valid public/private key pairs such that his signature on m is the same as Alice's. Furthermore, [BK00] illustrated that in some circumstances, DSKS-UKS can be carried out even in situations where Oscar can freely choose a fixed static message m' that may or may not be different from Alice's message m, and produces a valid signature $sig_{\mathrm{O}}(m')$ that is equal to Alice's signature on m, $sig_{\mathrm{A}}(m)$.

DSKS-UKS Attack on STS-MACi Against the Initiator

1. A $\rightarrow$ O: $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$
2. O(A) $\rightarrow$ B: $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$
3. O(A) $\leftarrow$ B: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$
4. Now Oscar selects a public/private key pair $(y_{\mathrm{O}}, x_{\mathrm{O}})$ so that the signature $sig_{\mathrm{O}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}})$ is the same as $sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}})$.
   A $\leftarrow$ O: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{O}}, sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$
5. A $\rightarrow$ O: $C_{\mathrm{A}}, sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$
6. O(A) $\rightarrow$ B: $C_{\mathrm{O}}, sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

DSKS-UKS Attack on STS-MACi Against the Responder

1. A $\rightarrow$ O(B): $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$
2. O $\rightarrow$ B: $ID(\mathrm{O}), \alpha^{r_{\mathrm{A}}}$
3. O $\leftarrow$ B: $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$
4. A $\leftarrow$ O(B): $\alpha^{r_{\mathrm{B}}}, C_{\mathrm{B}}, sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}))$
5. A $\rightarrow$ O(B): $C_{\mathrm{A}}, sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$
6. Oscar now picks public/private key pairs $(y_{\mathrm{O}}, x_{\mathrm{O}})$ so that the signature $sig_{\mathrm{O}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}})$ is the same as $sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}})$. Then
   O $\rightarrow$ B: $C_{\mathrm{O}}, sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}), \mathtt{MAC}_{sk}(sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}))$

The DSKS-UKS attacks on STS-ENCi are similar to the attacks on STS-MACi and thus will be omitted.

In DSKS-UKS attacks, the adversary knows the private key x_{O} corresponding to its public key y_{O} so that public-key validation does not prevent this type of attacks.

To prevent such an attack, [BWM99] suggests the following countermeasures:

1. Instead of sending her certificate in flow 3, Alice sends her certificate in flow 1. Since usually a certificates often consist of ID of a participant, the $ID(\mathrm{A})$ message in flow 1 can be omitted, thus reduce the overhead in flow messages. This way prevents the DSKS-UKS against the responder. However the attack against the initiator is still possible.

2. The protocol can require that, at the start of the protocol, the two intended parties in the protocol must exchange their certificates. Under this situation the assumption of DSKS-UKS no longer holds and thus the attack (which heavily depends on the assumption that certificates can be issued during a protocol run) fails. However, as indicated in [BWM99], a priori exchange of certificates increase the overhead of the protocol and sometimes may be undesirable.

3. Include the identities of the sender and intended receiver as well as the flow number in the messages being signed. An improvement of this solution is further to remove the MAC on the signatures. As pointed out in [BWM99] and many other articles, once the MAC under the session key sk has been exposed in the protocol flow, an eavesdropper can tell the MAC value of a specific message under sk; the adversary can use this to distinguish sk from a randomly uniformly selected key and thus violates the indistinguishability property of keys. Therefore, [BWM99] suggests the protocol to be

   (a) $\mathrm{A} \rightarrow \mathrm{B}$: $ID(\mathrm{A}), \alpha^{r_{\mathrm{A}}}$

   (b) $\mathrm{A} \leftarrow \mathrm{B}$: $C_{\mathrm{B}}, \alpha^{r_{\mathrm{B}}}, sig_{\mathrm{B}}(2, ID(\mathrm{B}), ID(\mathrm{A}), \alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}})$

   (c) $\mathrm{A} \rightarrow \mathrm{B}$: $C_{\mathrm{A}}, sig_{\mathrm{A}}(3, ID(\mathrm{A}), ID(\mathrm{B}), \alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}})$

   The above protocol will be referred to as STS-MAC1. [BWM99] claims that this modification in message format can prevent the DSKS-UKS. However, it was later pointed out (in [BK00]) that inclusion of the identities of the sender and the intended participant, and the flow number does not prevent the DSKS-UKS attacks (more to follow).

4. Instead of MAC the signature being sent, MACing the content of the signature. This is actually a standard ISO 11770-3, which are variants of STS-ENC and STS-MAC. The variation of STS-MAC will be referred to as ISO-STS-MAC and is described as follows:

   (a) A $\rightarrow$ B: $ID(\mathrm{A}), \alpha^{r_\mathrm{A}}$

   (b) A $\leftarrow$ B: $C_\mathrm{B}, \alpha^{r_\mathrm{B}}, sig_\mathrm{B}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}, ID(\mathrm{A})), \mathtt{MAC}_{sk}(\alpha^{r_\mathrm{B}}, \alpha^{r_\mathrm{A}}, ID(\mathrm{A}))$

   (c) A $\rightarrow$ B: $C_\mathrm{A}, sig_\mathrm{A}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}, ID(\mathrm{B})), \mathtt{MAC}_{sk}(\alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}, ID(\mathrm{B}))$

In the above suggested solutions, solution 1 does not fully solve the problem since the attack against the initiator is still possible. Solution 2 proposes a change in the assumption of the DSKS-UKS attack: if certificates are to be changed at the beginning of the protocol, this pretty much means that only registered users can participate in the protocol, and there would be no point in allowing registering public keys during a protocol run.

Solution 3 is a variation of STS-BCK; the difference is that STS-BCK only has signatures on the two exponentials and the ID of the intended recipient, while solution 3 has signatures on the two exponentials, the ID of the intended recipient, flow number, and the ID of the sender. In [BK00], Baek and Kim argued that solution 3 from above would not work if the signature schemes used are ElGamal or RSA.

Assume the ElGamal signature scheme is used in STS. Then the public key parameters of Alice is the triple $(p, \alpha, y_\mathrm{A})$, where $y_\mathrm{A} = \alpha^{x_\mathrm{A}}$, y_A is A's public key, and x_A is Alice's private key. p is a large prime with α being a generator for $\mathbb{Z}_p^*$. In flow 3 of STS-MAC1, Alice sends Bob the message $m_3 = (3, \alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}, ID(\mathrm{A}), ID(\mathrm{B}))$, and the signature on m_3 is $sig_\mathrm{A} = (r, s) = (\alpha^k, k^{-1}(H(m_3) - x_\mathrm{A} r))$, where H is a hash function used on messages before signing. Let $h = H(m_3)$. Bob verifies Alice's message by checking whether $\alpha^{-h} y_\mathrm{A}^r r^s \equiv 1 \mod p$. If so then the verification scheme outputs "valid", "invalid" otherwise. In the scenario of attack against the responder, upon seeing these messages, Oscar computes $h' = H(3, \alpha^{r_\mathrm{A}}, \alpha^{r_\mathrm{B}}, ID(\mathrm{O}), ID(\mathrm{B}))$ (note that this message replaces Alice's ID with Oscar's). Let q be a large prime which divides $p-1$. If $\gcd(s, p-1) \neq 1$ or if $\gcd(h', r) = 2$ or q, then Oscar terminates with failure. Otherwise, Oscar selects $x' \in_R \mathbb{Z}_q$, computes $t = h' - x'r$ and the new public parameter $(p, \overline{\alpha}, \overline{y_\mathrm{O}})$ will be $(p, (r^s)^{t^{-1}}, \overline{\alpha}^{x'})$. Note that Oscar knows the private key x' that corresponds to his public key $\overline{\alpha} x'$. Oscar registers his new public parameters in real time and gets his certificate C_O issued containing the new public values.

Bob now receives C_{O} and (r, s) in flow 3. Within the certificate Bob retrieves the public value $(p, \overline{\alpha}, \overline{y_{\mathrm{O}}})$, and he verifies the signature (r, s) with the public parameter $(p, \overline{\alpha}, \overline{y_{\mathrm{O}}})$. Bob first computes $h' = H(3, \alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}, ID(\mathrm{O}), ID(\mathrm{B}))$, checks whether $\overline{\alpha}^{-h'} \overline{y_{\mathrm{A}}}^{r} r^{s} \equiv 1 \mod p$. The calculation follows:

$$\begin{aligned}
\overline{\alpha}^{-h'} \overline{y_{\mathrm{A}}}^{r} r^{s} \mod p &= (r^{st^{-1}})^{-h'} (r^{st^{-1}})^{x'r} r^{s} \\
&= \left(r^{s/(h'-x'r)}\right)^{-h'} \left(r^{s/(h'-x'r)}\right)^{x'r} r^{s} \\
&= r^{(-sh'+sx'r+sh'-sa'r)/(h'-x'r)} \\
&= r^{0} \\
&= 1
\end{aligned}$$

Thus, Bob accepts. A similar illustration can be given for the RSA case.

So in the STS-BCK protocol, the adversary can just perform what's described above when he sees the message sent by Alice in flow 3. However, the adversary faces a slightly more challenging situation if dealing with STS-MAC1. In that protocol, both the intended recipient and the the sender is included in the signature. If Oscar simply forwards Bob's message to Alice in flow 2, Alice would immediately notice that Bob's intended participant is Oscar rather than her. Therefore, on top of having to modify the message in flow 3, Oscar now also has to modify the message in flow 2 and come up with a valid public parameter for that modified message. That is, Oscar must come up with a public key y_{O2} such that the signature $sig_{y_{O2}}(m') = sig_{\mathrm{B}}(2, ID(\mathrm{B}), ID(\mathrm{A}), \alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}})$ instead of the original signature $sig_{\mathrm{B}}(2, ID(\mathrm{B}), ID(\mathrm{A}), \alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}})$. If Oscar is able to do it, then Oscar has performed an existential forgery to the signature scheme. If the protocol uses any signature scheme that is GMR-secure ([GMR84]), then this attack is not possible. Even in situations where the signature scheme is not GMR-secure, the intuition suggests that it is very unlikely for the two public keys Oscar has to produce in flow 2 and flow 3 to be the same; Oscar must re-register his public key during the protocol run. The original assumption only states that Oscar is allowed to register his public key during a protocol run, but does not specify whether Oscar can change/re-register public keys during a protocol run. In most real-life systems, users are allowed to change their password/public keys at their own will, and it is a good practice to change keys often. This implies such an attack is possible and realistic but Oscar would certainly create more delays in the protocol. Whether Oscar may successfully conduct the attack before the protocol times out is another issue Oscar has to consider.

In STS-MAC1, however, Bob needs to know the ID of his intended participant before flow 2 so that he can produce a proper signature used in the protocol. Bob gets the ID in flow 1. Again, instead of sending ID(A), Alice's certificate can be sent and the same certificate does not have to be sent in flow 3. This way reduces the overhead of the flow and achieves the same purpose as STS-MAC1. However, sending certificate in flow 1 will then imply solution 1 suggested above, so the claim here is that solution 3 is just equivalent to solution 1. Similarly, STS-BCK requires signing the other party's ID and this makes DSKS-UKS not possible.

It seems at first sight that [BK00] is wrongfully worried about the security of STS. However, it does bring up two important issues of related issues: (1) the proper security definition of a signature scheme, and (2) the proper usage of a signature scheme. In the original GMR-security of signature schemes, we are pretty much assuming there is only one user in the world who has public keys and can sign messages; everyone else is just an verifier. Clearly it is not the case of today's world. The GMR-security is often referred to as the security of signature schemes in the *single-user setting*. Apparently, a more proper definition of secure signature schemes should be in a *multi-user setting* instead. This leads to [MS04]'s definition of a signature scheme.

Note that enforcing public keys to be distinct in all three versions considered here does not help in preventing the attack: the attacker can select a message in a format of his choice and creates the corresponding private/public key pairs such that the signature used by Alice is also his signature on the new modified message. (see [MS04])

5.4 Summary of Chapter

In this chapter, a case study of STS is conducted to examine the desired features a secure AKE should possess. The results from many previous papers are combined together to give an integrated presentation. After demonstrating several attacks described in the literature, a secure version of STS which possesses all the desired features, STS-BCK, is given (under the multi-party security definition of signature schemes).

In Chapter 6, we will also demonstrate a design methodology gives a secure AKE protocol with all the desired features by showing an AKE protocol constructed via the design methodology results in STS-BCK.

Chapter 6

Design Methodologies of Authenticated Key Exchange Protocols

6.1 Introduction and Overview

From the previous chapters, we now have a model of communication, the definition of secure authenticated key exchange protocols, and a list of desired of features of such a protocol. The only question left in creating a real secure authenticated key exchange protocol is how to design one.

Similar to the case in treating notions of security, the "attack and fix" approach seems unreasonable. Of course, people can always come up with a protocol and give a model for proof of security, but the method used to create the protocol in the first place is still "out of thin air". Even though Bellare and Rogaway's model is an important milestone in treating authenticated key exchange protocols, proving a protocol is secure under the Bellare-Rogaway model (perhaps with slight modification to fit various settings) is still a difficult job. It would be much nicer if a general approach can be adapted so that there is a universal step-by-step guideline to creating a secure AKE protocol.

In this chapter, two design approaches are presented and investigated. One is referred to as Bellare-Canetti-Krawczyk's Modular approach (or the BCK modular approach, or the modular approach for short), where they start with a key exchange protocol in an "ideal" setting, then apply their authenticators to yield an authenticated key exchange (or transport) protocol in a real, unauthenticated setting. The

second one is referred to as the progressive approach, which takes an existing secure mutual authentication protocol that uses two challenge-response phases, and replaces the random challenges with Diffie-Hellman exponentials to create a secure authenticated key exchange protocol. (Note that the progressive approach is very similar to the design methodology suggested by Bellare and Rogaway in [BR93B], but the settings for each method are very different.) We describe how each of the approach work by adapting each approaches to establish the same secure version of the STS protocol. The proof of security will be given, and followed by discussions, comparisons, and comments on these two approaches.

6.2 The Bellare-Canetti-Krawczyk's Modular Approach

This section is the description of the design methodology in [BCK98]. Please note that the formal model and definition of security of [BCK98] is described in Section 4.6; This section focuses on the steps in creating secure AKE protocols.

6.2.1 Overview and Concept

The world of [BCK98] has three parts: (1) An ideal world where key exchange is always safe; the adversary can do no harm to a key exchange protocol. (2) An authenticated environment where all messages sent in this environment are authenticated. Adversaries cannot send any messages and claim it is from another entity. (3) The "real world"; the adversary is powerful and can do various things (as will be explained later) to attack honest participants. Environment (3) resembles the normal computing environment we are in, and the other two are the worlds we wish to have.

The idea of the modular design approach is to first look at a simpler version of authenticated key exchange, namely just key exchange, then add additional authentication feature into the protocol. The design approach in this world takes the following steps, and an illustration is given in Figure 6.1:

1. Design a key exchange protocol KE and prove, in the authenticated network, that the adversary cannot be more harmful than an adversary in the ideal key exchange world.

2. Apply an *authenticator* to KE to yield a modified protocol KE'. Prove that KE' in the unauthenticated network *emulates* KE in the authenticated network. By Theorem 3 of [BCK98] KE' is a secure authenticated key exchange protocol in the "real world".

6.2.2 Constructing the Secure STS-BCK

Given the above design methodology, we now demonstrate an example of applying the modular approach to design a secure key exchange protocol. The examples of secure key exchange protocols in the authenticated links model, given in [BCK98], are the Diffie-Hellman key exchange and an encryption based key transport protocol ([BCK98] does not seem to distinguish key exchange and key transport very much). Also, three examples of $\mathsf{MT}-$authenticators are given: one is based on a secure signature scheme (under the GMR definition of security), one is based on a secure MAC, and another is based on a secure encryption scheme. In our example, the Diffie-Hellman key exchange (DH) and the signature based $\mathsf{MT}-$authenticator, $\mathsf{MT}-\mathsf{sig}$, and its corresponding authenticator λ_{sig} are used to construct the secure authenticated key exchange protocol in the unauthenticated links network.

For a review of the Diffie-Hellman problem (DHP), the reader can refer back to Subsection 3.3. The Diffie-Hellman key exchange protocol is derived almost directly from DHP. Suppose two entities A and B wish to establish a key *sk*. They first agree on a group $\mathbb{Z}_p$, where p is a prime, a generator $\alpha \in \mathbb{Z}_p$ with order q, where q is another prime. A first picks (uniformly and randomly) an exponent $r_{\mathrm{A}} \in_R \mathbb{Z}_q^*$ and sends $\beta_{\mathrm{A}} = \alpha^{r_{\mathrm{A}}}$ to B. B then picks uniformly and randomly another exponent $r_{\mathrm{B}} \in_R \mathbb{Z}_q^*$ and sends $\beta_{\mathrm{B}} = \alpha^{r_{\mathrm{B}}}$ to A. A computes the established key as $(\beta_{\mathrm{B}})^{r_{\mathrm{A}}} = (\alpha^{r_{\mathrm{B}}})^{r_{\mathrm{A}}} = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$. B computes the established key as $(\beta_{\mathrm{A}})^{r_{\mathrm{B}}} = (\alpha^{r_{\mathrm{A}}})^{r_{\mathrm{B}}} = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$. An illustration is given in Figure 6.2. From [BCK98] the protocol DH is proven to be a secure key exchange protocol in the authenticated environment, provided that the DDH is intractable.

The signature-based MT-authenticator is as follows. If A wishes to send message m to B in an authenticated fashion, she first sends m to B. Upon receiving m, B replies with m and a challenge r'_{B} to A. After A receives the reply from B, she sends the same message m, along with her signature on the tuple consisting of m, the random challenge r'_{B}, and B's identity $ID(\mathrm{B})$. B verifies that the signature he receives is valid, then outputs "B has received message m from A". (This output can take various forms. For example, the output may be a transition to the "accept" state internal to B's machine.) Figure 6.3 illustrates the signature-based MT-authenticator.

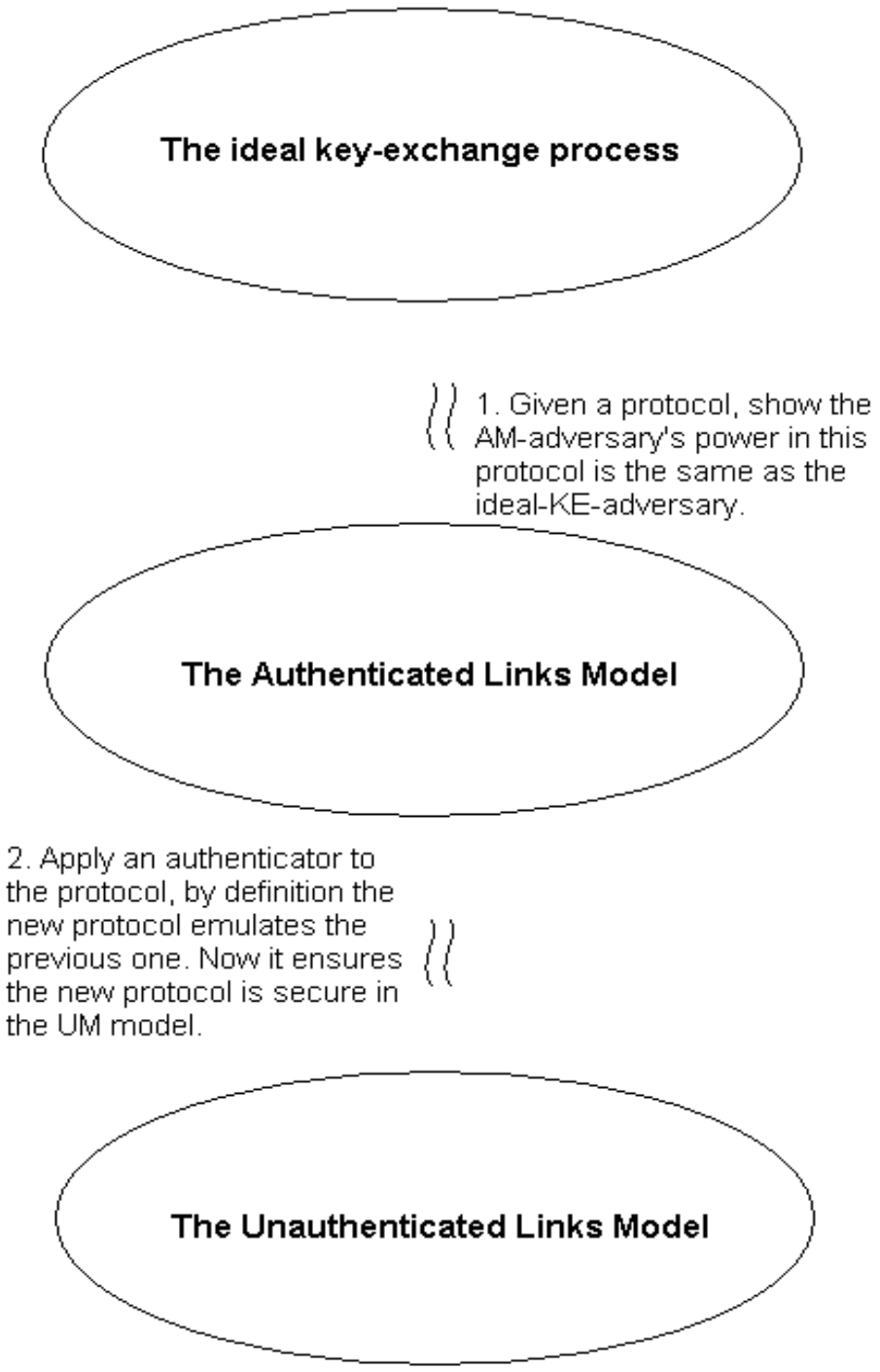

Figure 6.1: BCK's Process of Creating Secure Authenticated Key Exchange Protocol

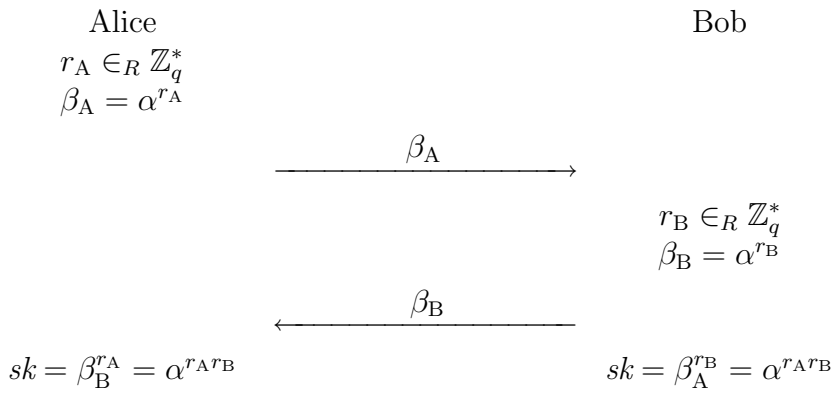

Figure 6.2: Diffie-Hellman Key Exchange

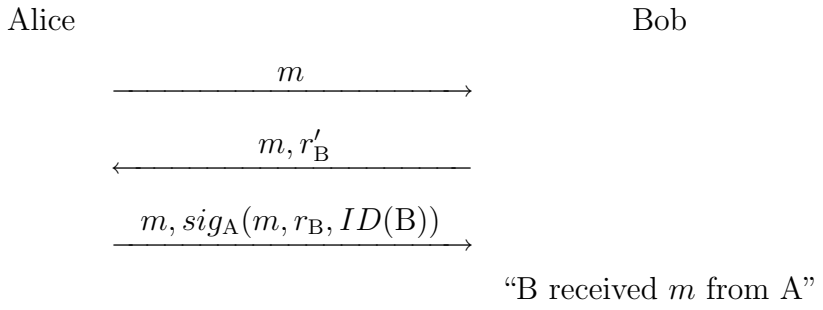

Figure 6.3: Signature based MT-Authenticator

As explained in [BCK98], an MT-authenticator can be used to construct an authenticator and to apply to secure protocols in the authenticated links model to achieve security in the unauthenticated links model. It authenticates every message transferred in the protocol. Therefore, the Diffie-Hellman key exchange protocol DH can be further developed using the signature based authenticator:

1. (Flow 1 of DH) Alice tries to send a random exponent α_{A}^{r} to Bob. Alice picks a random exponent $r_{\mathrm{A}} \in_R \{\mathbb{Z}_q^*\}$ and sends $\alpha^{r_{\mathrm{A}}}$ to Bob.
2. (Flow 1 of DH) Bob sends back $\alpha^{r_{\mathrm{A}}}$ and his random challenge r_{B}' to Alice.
3. (Flow 1 of DH) Alice sends her exponential, $\alpha^{r_{\mathrm{A}}}$, and her signature on $\alpha^{r_{\mathrm{A}}}$, r'_{B}, and $ID(\mathrm{B})$ to Bob. Bob accepts the exponential as being authenticated.
4. (Flow 2 of DH) Bob uniformly randomly picks an exponent r_{B} and sends $\alpha^{r_{\mathrm{B}}}$ to Alice.
5. (Flow 2 of DH) Alice sends Bob $\alpha^{r_{\mathrm{B}}}$ and her random challenge r'_{A}.
6. (Flow 2 of DH) Bob sends back $\alpha^{r_{\mathrm{B}}}$ and his signature on $\alpha^{r_{\mathrm{B}}}$, r'_{A}, and $ID(\mathrm{A})$. Alice accepts $\alpha^{r_{\mathrm{B}}}$ as an authenticated message. Each Alice and Bob computes the session key to be $sk = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$.

An illustration is shown in Figure 6.4.

6.2.3 Proof of Security

To show the security of STS-BCK version 1, the first step is to apply Proposition 9 of [BCK98], which says that the protocol DH is a secure key exchange protocol in the authenticated links network. Theorem 3 and Proposition 4 of [BCK98] say that the described method of incorporating the signature based MT-authenticator gives an authenticator. Finally, Corollary 8 of [BCK98] ensures that applying the signature-based MT-authenticator to DH results in a secure key exchange protocol in the unauthenticated network. Thus we have a proof. Note that the proofs in [BCK98] are partially sketches; there does not seem to be a complete proof of [BCK98]'s claim in the literature. In [Sho99], however, Shoup says the proofs in the BCK model does not resemble enough real-life situations and one protocol safe in the BCK model can be still vulnerable in real life. See also the Appendix in the full version of [CK01].

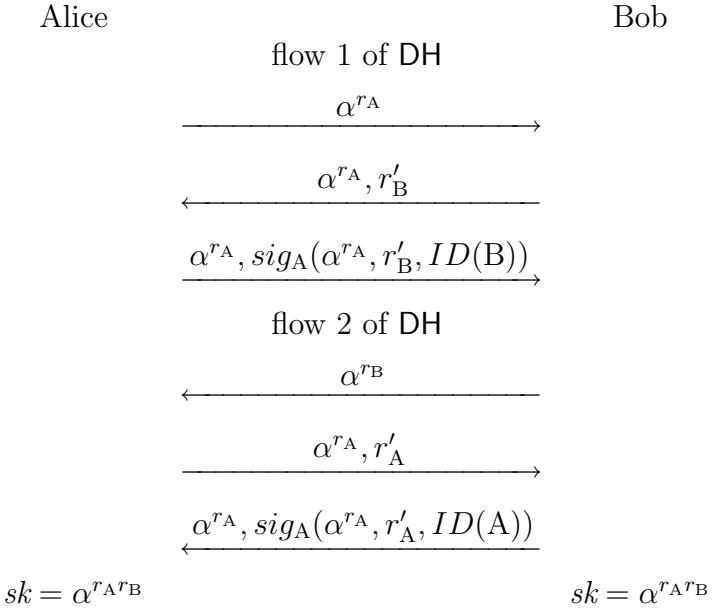

Figure 6.4: STS-BCK Version 1

6.2.4 Further Refinement

Obviously, the STS-BCK version 1 is not the STS-BCK we showed in the previous chapter. However, notice that the exponentials sent from A to B and vice versa are in a way also random values. Therefore, there are some redundancies in STS-BCK version 1, and the overheads in the protocol flows can be further optimized.

Note that B sends a random challenge r'_B in flow 2 of STS-BCK version 1, and B also sends his random exponential α^{r_B} in flow 4. If we make these two random values equal and they are sent in the same flow, the order of authenticating flow 2 of DH will not be changed. Correspondingly, A's signature in flow 3 will be on the content of her random exponential, B's random exponential, and the ID of B. The resulting protocol, referred to as STS-BCK version 2, is illustrated in Figure 6.5.

Notice that STS-BCK version 2 has 4 flows as compared to 6 flows in STS-BCK version 1. In version 1, the overheads consists of transmission of 6 random exponentials, 2 random values (which may or may not be from the random exponential domain), and two signatures (Note that the length of a signature is usually bounded regardless of the content being signed. Therefore, the extra content in a signature does not imply more overhead in a protocol flow.) However, in version 2, the flows only consist of transmission of 6 random exponentials, 0 random values

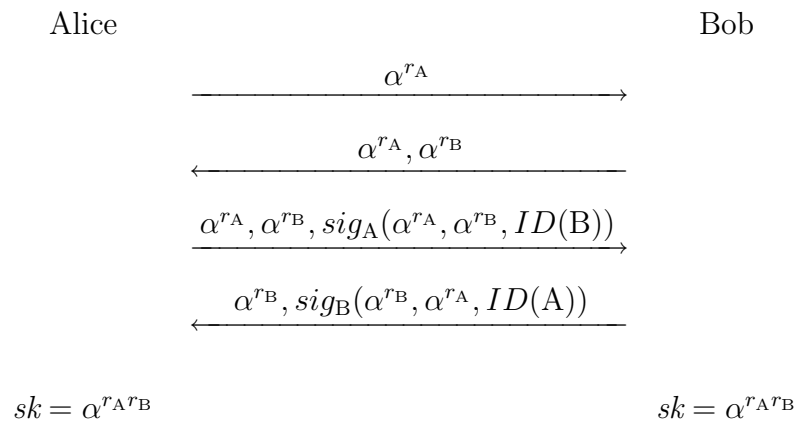

Figure 6.5: STS-BCK Version 2

(as opposed to 2 in version 1), and 2 signatures. The reduced number of protocol flows and the reduced overhead in the flows will mean improvement in the run time in actual implementation. Also, the order of message transfer of the authenticator is preserved, therefore it seems reasonable (although no proof is given) that the security of STS-BCK version 2 is not compromised.

[BCK98] gives an "outline" of truncating the excessive overheads in applying authenticators. The paper claims that the original 6-flow protocol can be reduced into a 3-flow one by "piggy-backing" the messages. However, it is not clear to the author how the protocol can be simplified without altering the order of an authenticator applied in the protocol, and how one can be sure that the simplified protocol is still secure (since it is in a different form from the original one). Let the secure 3-flow version of STS-BCK be named STS-BCK version 3 for now. Blake-Wilson and Menezes in [BWM98] present the protocol produced by their interpretation of [BCK98] is STS-BCK version 3. The simplification process and the preservation of security after simplification remains unclear, but the protocol from [BWM98] is illustrated in Figure 6.6.

Note that the number of flows in version 3 has been reduced from 4 to 3. The overheads in version 3 consist of transmitting only 2 random exponentials and 2 signatures, which is a great improvement from the 6 exponentials and 2 signatures in version 2. However, version 3 seems to be moving flow 4 of version 2 into flow 2, and the original flow 2 in version 2 is completely erased. Also, Alice's exponential in flow 3 of version 2 is also erased in version 3, with flow number unchanged. This way, the number of overheads and its order have been changed and different from the original signature-based MT-authenticators. Although it seems acceptable to make

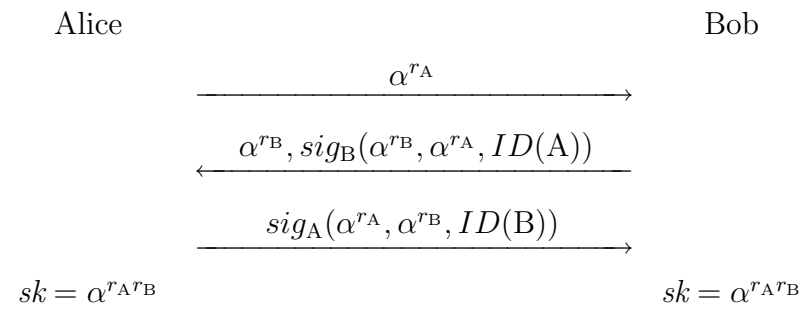

Figure 6.6: STS-BCK Version 3

such modifications while preserving the security of the protocol, no proof is given in the literature that this is secure, and the original paper does not propose any concrete way (nor examples) of applying authenticators to key exchange protocols (in the authenticated setting) and reducing the overheads while maintaining the same level of security.

6.2.5 Discussions and Comments

Although some discussions are already mentioned while describing the construction of STS-BCK via the modular approach, it is worthwhile to look at the design approach again and analyze the concept, range of applications, and potential problems and future developments.

To start with, it is mentioned again that in the original paper ([BCK98]), the concept and brief algorithm of constructing a secure key exchange protocol is given, however, no concrete example is provided in the paper, and it may lead to misinterpretation of the content into vulnerable implementations. Also, Blake-Wilson and Menezes's interpretation of constructing STS-BCK version 3 is not straightforward. In the possibility of their interpretation making changes to the original authenticator, it may yield a secure protocol, but with no proof (since the original authenticator has been altered, a new model or new proof may be needed). The reduction also raises the concern that whether applying authenticators to secure key exchange protocols in the authenticated setting will generate excessive overhead or more protocol flows than necessary. However, [BCK98] does not seem to put much emphasis on this issue. Also, how to optimize the real implementation with respect to number of flows and amount of overheads, is still unclear from the paper.

Regardless of possible implementation issues, the concept of layering settings via giving different limitations to adversaries may suggest a wider range of appli-

cation of this concept. For example, the encryption based key "exchange" protocol (proven secure in the authenticated links model) is in fact a key transport protocol. This implies that authenticators can also be applied to at least another usage other than key exchange. Since authenticators give one way authentication only, the application of authenticators can be used in constructing key establishment protocols (key exchange, key transport, etc.) in various settings such as achieving unilateral or mutual authentication and with or without key confirmation (if key confirmation is achieved by simply having one additional flow in the protocol, e.g. using a MAC).

The focus of [BCK98] is on key establishment between two entities in a distributed computing network. The intended setting of this paper is then very different from Bellare and Rogaway's model in [BR93B] (which is intended for a generalization to group key distribution). One potential future work of such a model may be generalizing the design methodology so that it also applies to the group key distribution case.

6.3 The Progressive Approach: From Mutual Authentication Protocols to Authenticated Key Exchange

Another approach to the design of secure AKE protocols is based on modifying only the essential parts in mutual authentication protocols to further achieve secure key exchange. In other words, it start with a secure mutual authentication protocol, then progressively change some of the components to achieve key exchange and thus will be referred to as the progressive approach. The idea of progressive approaches first appeared in [BR93B] by Bellare and Rogaway for group key distribution in the secret-key cryptography setting. However, Bellare and Rogaway did not go into details of designing a protocol; rather they only mentioned a secure AKE protocol can be created using such an approach. In [BR93B], not only does every user shares a secret key with another before the protocol starts, there was no need for integrity and non-repudiation tools. The approach described in this section, however, is a similar idea but in the asymmetric setting targeted to achieve authenticated key exchange. Originally developed by Professor Doug Stinson, this idea was never published but used as teaching material. The concept of this approach is similar to what was suggested by Bellare and Rogaway but in an asymmetric setting. It is believed that such a design methodology for the asymmetric setting may give an alternative method of designing a secure key exchange protocol.

The rest of this section is as follows: The assumption and the communication model will be described, and some contrasts and similarities from the other models will be mentioned. The concept of design approach is then discussed. The steps of constructing a secure key exchange protocol are given, followed by an illustration of constructing the same STS-BCK developed in [BCK98]. A complete proof of security is given, and the method of proving security is also explained before the description of the proof. Finally, some comments and discussions will be provided to complete this subsection.

6.3.1 Setting and Assumptions

The setting described for this approach will be based on more intuitive notions rather than a formalization of real life situations. This is done because (1) the proof of the protocol security later does not require a formal model, and (2) it is easier to understand the material.

Our computing environment is the normal distributed computing environment, where each user and the adversary are polynomially bounded in computing power as well as space. The asymmetric setting follows the traditional public key infrastructure (PKI): a trusted authority (TA) is present to issue legitimate certificates, certify each user (including the adversary) and register their private and public key pairs. The TA has a universal verification algorithm ver_{TA} to verify the validity of certificates. We require that each user must have their private/public key pairs registered as well as issued a certificate from TA before the user can start any protocol run with any other user. Note that in Blake-Wilson and Menezes' model, although not explicitly specified, distribution (or registration) of public/private key pairs and certificates also must be done prior to the start of the protocol.

In terms of security of signature schemes, we require the secure definition of [MS04] (the secure signature scheme in the multi-user setting), and we emphasize that each user must have a distinct public key, for otherwise one entity can impersonate another very easily. To illustrate this point, suppose that A and B have the same public key (the verification key) $psk_{\mathrm{A}} = psk_{\mathrm{B}}$. Note that their private key may or may not be the same. Suppose A signs a message m using her private key (the signing key) ssk_{A} to get $s = sig_{ssk_{\mathrm{A}}}(m)$. In the verification process, the verify algorithm is deterministic. That is, given the same key k and a signed message on m' using k, $sig_k(m')$, the verification process $ver_k(m', sig_k(m'))$ will always return true. Therefore, if $psk_{\mathrm{A}} = psk_{\mathrm{B}}$, then $ver_{psk_{\mathrm{A}}}(s, m) = \mathsf{true}$, but also $ver_{psk_{\mathrm{B}}}(s, m) = \mathsf{true}$. Thus if B uses A's signature as his own on message m, everyone would accept the

verification process; impersonation occurs. This requirement of signature security does not appear in [BR93B, BCK98, BWM97].

One practical concern about the enforcement of distinct public keys is that it does not seem possible in practice. In most theoretical models, there is only one TA, and it is very easy for a TA to verify that each user registers a different public key than the others (also the TA can make sure that, when a user registers its public key, the user must know the corresponding private key). However, in a real-world setting such as North America, there is certainly more than one TAissuing certificates and registering private/public keys. If two TAs use the same signature scheme with the same security parameter and same generator of the group, it is very likely that an entity A can register her public key to TA1 and another entity C can register the same public key to TA2. Thus in situations where the communication is conducted with the TAs off-line, A and C can impersonate each other.

The adversary's ability is the same as in both the Bellare-Rogaway and Blake-Wilson-Menezes model. However, we separate the ability of the adversary (and thus the adversary) into two categories: passive attacks and active attacks. Passive attacks consists of eavesdropping only; active attacks consists of message injection, modification, dropping or delaying messages, impersonation, and known-session key attacks. An adversary that only performs passive attacks is referred to as the passive adversary, and an adversary that does both passive and active attacks are referred to as an active adversary.

A secure mutual authentication protocol here is defined as one in which *no honest participant should accept after the adversary has been active.* This definition complies with the definition of mutual authentication in [BR93B].

Recall that [BR93B] defines mutual authentication as (1) if two (honest) parties have matching conversations, then both parties should accept, and (2) (negligible probability for $\mathsf{No-Matching}$) the probability of one honest party accepting while there is no other honest party engaging a matching conversation with it is negligible. From the above, condition (1) certainly complies with our definition. Regarding condition (2), note that $\mathsf{No-Matching}$ could happen only when an adversary is active, and condition (2) requires that the honest participant should not accept when active attacks happen. Thus condition (2) also complies with our definition.

The security of key exchange says that when both honest parties accept, the adversary should not be able to learn any information about the session key.

6.3.2 Design Concepts

The design concept follows directly from the separate treatment of mutual authentication and key exchange: start with a secure mutual authentication protocol that consists of random challenge and response technique, then replace each random challenge with Diffie-Hellman exponentials to form a Diffie-Hellman key exchange. Assumption: the random challenges and the Diffie-Hellman exponentials must be chosen randomly from domains with the same distribution, usually a uniform distribution. The procedure of proving security also follows from the design principle: first prove that the mutual authentication protocol is secure. Then, after replacing the random challenges with DH exponentials, impersonation is not possible, so the only active attack possible would be the known-session key attack. The security of the key exchange protocol can be proven once it is shown secure against the known session key attack. The steps in enumerated form are given as follows:

1. Design a mutual authentication protocol using random challenge and response technique. Prove its security. Note that in order to show mutual authentication, two random challenges are needed.
2. Replace the two random challenges with Diffie-Hellman exponentials to form a variation of Diffie-Hellman key exchange protocol. Let the resulting protocol be called MADHKE (stands for "Mutual Authenticated Diffie-Hellman Key Exchange").
3. Prove the security of MADHKE given that impersonation is not possible. The only way the adversary can retrieve information about the session key is by either eavesdropping or participating in the protocol as a legitimate user (using his own ID).

An illustration of this approach is given in Figure 6.7.

6.3.3 Construction of Secure Key Exchange Protocols with the Example of STS

We now demonstrate an example of constructing a secure authenticated key exchange protocol via the steps above. The proof is given in the next subsection and therefore only the construction is illustrated in this subsection.

We start by describing a secure mutual authentication protocol, which will be denoted as MAP. If entities Alice (denoted by A) and B (denoted by Bob) wish to authenticate themselves to each other, then

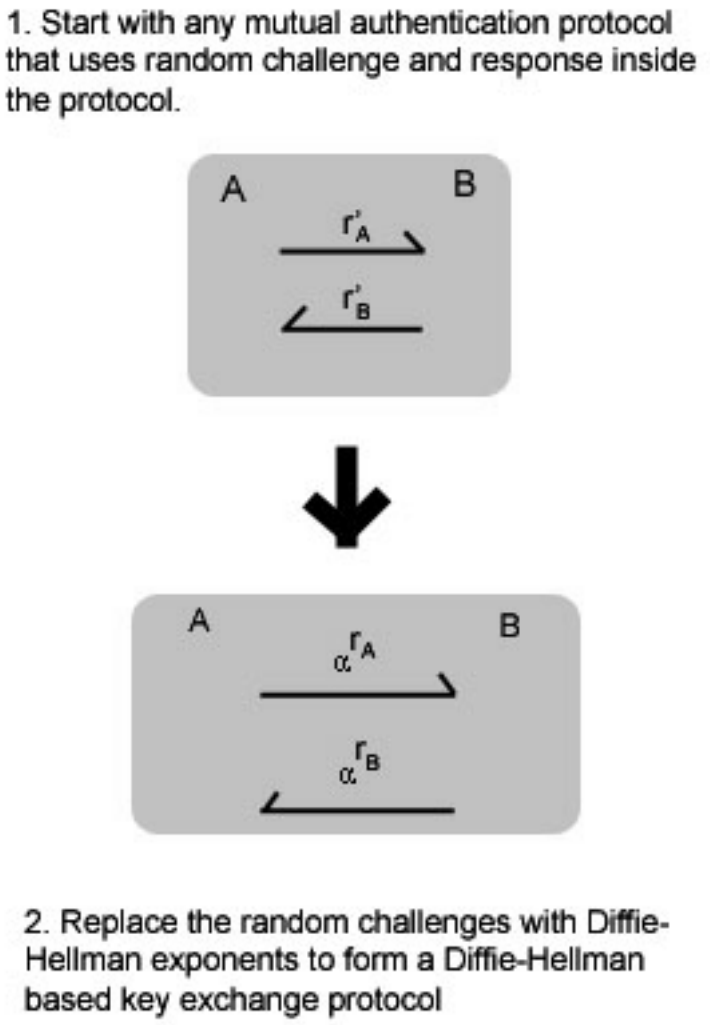

Figure 6.7: How to prove security of key exchange protocol

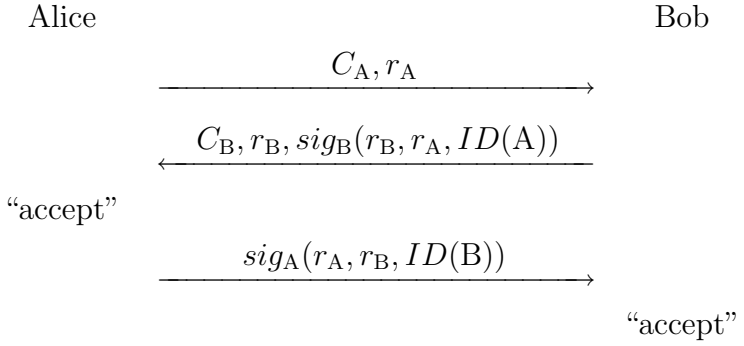

Figure 6.8: The Secure Mutual Authentication Protocol (MAP)

1. Alice (the initiator) chooses a random challenge r_{A} uniformly and randomly from a particular domain $\mathcal{D}$ and sends C_{A} and r_{A} to Bob.

2. Bob (the responder) chooses a random challenge r_{B} uniformly and randomly from the same domain $\mathcal{D}$. Bob computes

$$y_{\mathrm{B}} \quad = \quad sig_{\mathrm{B}}(r_{\mathrm{B}}, r_{\mathrm{A}}, ID(\mathrm{A}))$$

   and sends $C_{\mathrm{B}}, r_{\mathrm{B}}, y_{\mathrm{B}}$ to Alice.

3. Alice verifies the validity of C_{B} using ver_{TA}. Once verified, Alice extracts Bob's public key (the verification key) and verifies the validity of y_{B} using ver_{B}. If y_{B} is not valid, then Alice outputs "reject" and terminates the session. Otherwise, Alice outputs "accept", she computes

$$y_{\mathrm{A}} \quad = \quad sig_{\mathrm{A}}(r_{\mathrm{A}}, r_{\mathrm{B}}, ID(\mathrm{B}))$$

   and then sends y_{A} to Bob.

4. Bob verifies the validity of C_{A} using ver_{TA}. Once verified, Bob extracts Alice's verification key and verifies the validity of y_{A} using ver_{A}. If y_{A} is not valid, then Bob outputs "reject" and terminates the session. Otherwise, Bob outputs "accept" and the session is successful.

An illustration is shown in Figure 6.8

The proof of MAP's security is given in the next subsection.

Note that there are two transmissions of random challenges: one from Alice to Bob and another one the other way around. Now replace those random challenges with Diffie-Hellman exponentials. That is, instead of just sending the random values r_{A} and r_{B}, send $\alpha^{r_{\mathrm{A}}}$ and $\alpha^{r_{\mathrm{B}}}$, where α is a generator of the group $\langle\alpha\rangle \subseteq \mathbb{Z}_p$, and the order of α is some prime number q. It is required that choosing random exponentials should follow the same distribution with choosing random numbers and the DDH is intractable.

The resulting protocol becomes:

1. Alice chooses uniformly and randomly a random number $r_{\mathrm{A}} \in_R \mathbb{Z}_q^*$ and sends C_{A}, $\alpha^{r_{\mathrm{A}}}$ to Bob.

2. Bob chooses uniformly and randomly a random number $r_{\mathrm{B}} \in_R \mathbb{Z}_q^*$ and computes $y_{\mathrm{B}} = sig_{\mathrm{B}}(\alpha^{r_{\mathrm{B}}}, \alpha^{r_{\mathrm{A}}}, ID(\mathrm{A}))$ and sends C_{B}, r_{B}, and y_{B} to Alice.

3. Alice verifies the validity of C_{B} using ver_{TA} and extracts Bob's verification key. Then Alice verifies the validity of y_{B} using ver_{B}. If y_{B} is not valid, then Alice outputs "reject" and terminates this session. Otherwise, Alice computes $y_{\mathrm{A}} = sig_{\mathrm{A}}(\alpha^{r_{\mathrm{A}}}, \alpha^{r_{\mathrm{B}}}, ID(\mathrm{B}))$ and $sk = (\alpha^{r_{\mathrm{B}}})^{r_{\mathrm{A}}} = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$, she outputs "accept", and she sends y_{A} to Bob.

4. Bob verifies the validity of C_{A} using ver_{TA} and extracts Alice's verification key. Then Bob verifies the validity of y_{A} using ver_{A}. If y_{A} is not valid, then Bob outputs "reject" and terminates the session. Otherwise, $sk = (\alpha^{r_{\mathrm{B}}})^{r_{\mathrm{A}}} = \alpha^{r_{\mathrm{A}} r_{\mathrm{B}}}$ and Bob outputs "accept"; the protocol terminates successfully.

An illustration is shown in Figure 6.9.

Notice that the protocol STSMAP is identical to STS-BCK version 3 (see Figure 6.6) except for the exchange of certificates. The STS-BCK version 3 assumes that certificates are exchanged prior to start of the protocol (see the model), and in STSMAP the certificates are exchanged in protocol flows. In practice, the process of exchanging certificates still needs to be authenticated, which leads back to the same problem in authenticated key exchange. It is more practical and more efficient to exchange certificates during protocol flows. When Alice (the initiator) sends the random exponential in the first flow, she must indicate who she is at some point in order for Bob to know who to send the message back to. This "return address" feature can be done in the IP layer, where the return path is just the IP address or email address of Alice. Alice can also include her certificate in the first

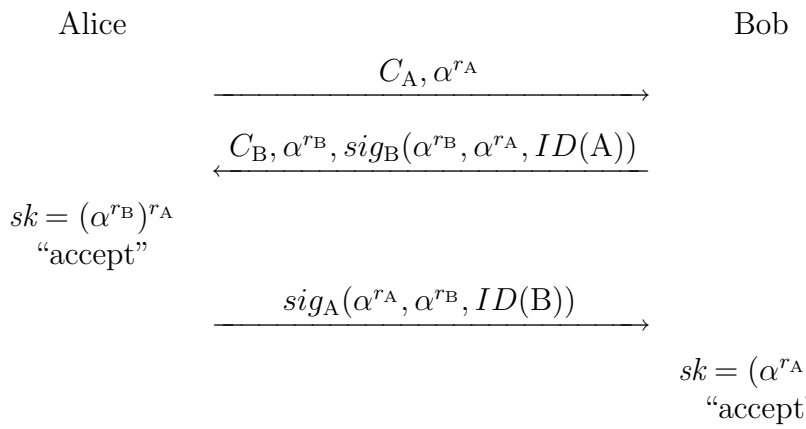

Figure 6.9: The STS derived from MAP (STSMAP)

flow, and since certificates contain Alice's ID, then Bob would know who to send the succeeding messages to. Therefore, we claim that the STS-BCK in practice should be implemented in the same way as STSMAP, but the security of the two are equivalent.

6.3.4 Proof of Security

The proof of security is separated into two parts: first prove that STSMAP is a secure mutual authentication protocol, then prove it is also a secure authenticated key exchange protocol under the assumption that the adversary cannot impersonate any legitimate user (this is a consequence of being a mutual authentication protocol).

Note that we believe the proof will suffice the security definition given by [BWJM97]. Also, we believe the proof also cover Shoup's comments of [BCK98]'s security issues with the model. Briefly speaking, an adversary should be allowed to be active even after it has queried the test oracle call. We claim that our proof, which does not follow the structure of formal modelling, has taken the above scenario into consideration so that the proof of security is valid in real life.

Assumptions

- C_{A} and C_{B} are public certificates issued by a trusted third party (TA), and the validation of C_{A} and C_{B} can be verified by a public verification algorithm

ver_{TA}. Also, the verification algorithm ver_{A} of A, (ver_B of B) can be obtained from C_{A} (and C_{B}, respectively). The corresponding signing algorithm (signature scheme) sig_{A} and sig_{B} are known only to A, B, respectively.

- A and B have perfect random number generators for exponents of α. Suppose each random number is k bits, then the probability of one number occurring is $1/2^k$ (uniform distribution).
- The signature scheme is secure. That is, the probability of forging a valid signature scheme $sig(x)$ after seeing q valid signatures $sig(x_1), sig(x_2), ..., sig(x_q)$ is at most ε, where ε denotes a (supposedly) small quantity ≥ 0.
- ver_{TA}, C_{A}, C_{B}, ver_{A}, and ver_{B} are all publicly accessible and known to all parties at the beginning of the protocol.
- Only Alice and Bob have access to sig_{A} and sig_{B}, respectively.

Proof that STSMAP is a secure mutual authentication protocol

Proof. Suppose on the contrary that STS is not a secure mutual authentication protocol, then Oscar can either

1. impersonate Alice to Bob
2. impersonate Bob to Alice
3. impersonate both Alice and Bob to Bob and Alice, respectively.

The probability of Oscar's success of impersonation is

$$\max\{\Pr[\text{case1}], \Pr[\text{case2}], \Pr[\text{case3}]\}.$$

Case1: Suppose Oscar impersonates A to B. That is, B "accepts" at the end of a protocol run. This implies that in the third flow B receives $y'_{\mathrm{A}} = sig_A(a', \alpha^{r_{\mathrm{B}}}, ID(\mathrm{B}))$, where a' is the random exponential B receives in the first flow, and r_{B} is the random number generated by B, $\alpha^{r_{\mathrm{B}}}$ is the value sent by B in the second flow. Now Oscar may obtain a valid y'_A in the following ways:

1. y'_A is constructed by Oscar. By our assumption, Oscar has seen q valid sig_A signatures, thus he can only succeed with probability ε.

2. y'_A has been previously constructed by A. The only way this could happen is that Oscar replays an exponential a' previously used by A, say, in session i, and B sends the b' he sends in session i. Let the previously viewed signatures be $s_1 = sig_A(ID(B)||\alpha^{a_{i1}}||\alpha^{b_{i1}}), s_2 = sig_A(ID(B)||\alpha^{a_{i2}}||\alpha^{b_{i2}}), ..., s_q = sig_A(ID(B)||\alpha^{a_{iq}}||\alpha^{b_{iq}})$, and let $S = \{s_1, s_2, ..., s_q\}$. Let $S_{a'}$ be the set of distinct $\alpha^{a_{ij}}, j = 1..q$ in S, and define two elements $a' = \alpha^a$ and $b' = \alpha^b$ to be *associated* if $\exists s \in S$ such that $s = sig_A(ID(B)||a'||b')$. Let $S_{a'}(\alpha^b)$ be the list of α^b's that are associated with a'. Note that in $S_{a'}(\alpha^b)$ there might be identical elements (since it's a list instead of a set). let $S_{a'} = \{a'_1, a'_2, ..., a'_t\}$ and let $|S_{a'_1}(\alpha^b)| = q_1, |S_{a'_2}(\alpha^b)| = q_2, ..., |S_{a'_t}(\alpha^b)| = q_t$. Note that $q_1 + q_2 + ... + q_t = q$. Thus,

$$
\begin{aligned}
& \Pr[\text{Oscar uses a previously observed signature by A}] \\
= \; & \Pr[\text{Oscar sends an } \alpha' \in S_{a'} \text{ in the first flow} \\
& \wedge \text{ B sends back an associated } \alpha^b \text{ in the second flow}] \\
= \; & \sum_{i=1}^{t} \Pr[\text{Oscar sends an } \alpha'_i \text{ in the first flow} \\
& \wedge \text{ B sends back an } \alpha^b \text{ associated with } \alpha'_i \text{ in the second flow}] \\
= \; & \sum_{i=1}^{t} \Pr[\text{B sends back an } \alpha^b \text{ associated with } \alpha'_i | \text{ Oscar sends } \alpha'_i] \times \\
& \Pr[\text{Oscar sends } \alpha'_i] \\
\leq \; & \sum_{i=1}^{t} \Pr[\text{B sends an } \alpha^b \text{ associated with } \alpha'_i | \text{ Oscar sends } \alpha'_i] \\
& (\text{since } \Pr[\text{Oscar sends } \alpha'_i] \leq 1) \\
= \; & \sum_{i=1}^{t} \frac{|S_{a'}(b)|}{2^k} \\
= \; & \frac{q_1}{2^k} + \frac{q_2}{2^k} + \cdots + \frac{q_t}{2^k} \\
= \; & \frac{q}{2^k}
\end{aligned}
$$

Therefore, the probability of Oscar impersonating A using q previously observed signature is at most $q/2^k$.

3. If y'_A is previously constructed by B, notice that B only constructs signatures of the form $sig_B(ID(A)||\alpha^b||\alpha^a)$, so this scenario does not happen.

Therefore, Pr[Oscar impersonates A] $\leq q/2^k + \varepsilon$.

Case2: Suppose Oscar impersonates B to A, then A receives $y'_B = sig_B(ID(A)||b'||a')$, where a' is sent by A to Oscar in the first flow, and b' is the value sent along with y'_B in the second flow. Oscar can do this via

1. constructing y'_B himself. By our assumption the probability of achieving this is $\leq \varepsilon$.

2. using a previous signature by A. This is not possible by similar argument as before.

3. using a previous signature by B. As long as A sends an $a' \in S_{a'}$, then there exists a signature $sig_B(ID(A)||b'||a') \in S$ for some b'. Therefore,

$$\begin{aligned} & \Pr[\text{Oscar attacks using a previous signature by B}] \\ = \; & \Pr[\text{A sends an } a' \in S_{a'}] \\ = \; & \frac{|S_{a'}|}{2^k} \leq \frac{q}{2^k} \end{aligned}$$

Therefore, the total probability of Oscar impersonating B is $\leq q/2^k + \varepsilon$.

Case3: The third case which Oscar impersonates both A and B is definitely harder than the previous two cases, thus the probability of Oscar impersonating A and B is $\leq q/2^k + \varepsilon$.

Therefore, the probability of Oscar's success of impersonation is at most $\leq q/2^k + \varepsilon$. The security of STSMAP's mutual identification feature has been established as a function of security of its underlying primitives (the signature scheme and the random number generator). Since we assume that the signature scheme is secure and the random number generator is perfectly random, we conclude that STSMAP is a secure mutual authentication protocol. □

Proof that STSMAP is a secure authenticated key exchange protocol

Proof. The proof is conducted in 3 steps. First, we show that a passive attacker can break the protocol no more easily than solving the decision Diffie-Hellman problem (DDH). The second step is to show that STS-BCK is a secure protocol under mutual authentication. This ensures that an active attacker can only impersonate either

Alice or Bob with a small probability ε'. The third step is to show that active attackers without impersonation can succeed no more easily than solving DDH.

Recall that the DDH problem is: given a $(\beta_1 = \alpha^{a_1}, \beta_2 = \alpha^{a_2}, \beta_3)$ tuple, is $\beta_3 = \alpha^{a_1 a_2}$?

Step 1: Suppose Oscar's goal is to learn some information about the session key *sk*. Since this information can be arbitrary, we can model the adversary's gain of this piece of information by allowing the adversary to tell the real session key from any other random key with non-negligible probability. In terms of the previous formal models, the adversary can guess the random bit test generates with probability non-negligibly greater than 1/2. Suppose Oscar has an polynomial-time oracle $\mathcal{O}$ such that it can return some information about the session key $sk = \alpha^{ab}$ given inputs $\beta_1 = \alpha^a$, $\beta_2 = \alpha^b$, and many valid transcript information tuples $(\alpha^{a_1}, \alpha^{b_1}, sk_1)$, $(\alpha^{a_2}, \alpha^{b_2}, sk_2)$, $\ldots$, $(\alpha^{a_q}, \alpha^{b_q}, sk_q)$ with probability $1/2 + \varepsilon$, where $\varepsilon > 0$. Here, a_i, b_i, are generated by a perfect random number generator, and thus the α^{a_i}s and α^{b_i}s follow a particular distribution $\mathcal{D}$. We claim that Oscar can use this oracle to solve DDH, thus the oracle should not exist (with run time complexity polynomial, anyway).

Given the problem instance $(\beta_1 = \alpha^a, \beta_2 = \alpha^b, \beta_3)$, Oscar can generate random numbers $a_1,\ a_2, \ldots,\ a_q,\ b_1,\ b_2, \ldots,\ b_q$, and compute triples $(\alpha^{a_1}, \alpha^{b_1}, \alpha^{a_1 b_1})$, $(\alpha^{a_2}, \alpha^{b_2}, \alpha^{a_2 b_2})$, $\ldots$, $(\alpha^{a_q}, \alpha^{b_q}, \alpha^{a_q b_q})$. These tuples follow the distribution $\mathcal{D}$ of the observed tuples in some legitimate STS sessions. Now Oscar calls

$$\mathcal{O}(\beta_1 = \alpha^a, \beta_2 = \alpha^b, (\alpha^{a_1}, \alpha^{b_1}, \alpha^{a_1 b_1}), (\alpha^{a_2}, \alpha^{b_2}, \alpha^{a_2 b_2}), \ldots, (\alpha^{a_q}, \alpha^{b_q}, \alpha^{a_q b_q}))$$

and retrieves to get the information about the session key *sk*in polynomial time. Now Oscar is able to tell if *sk*$= \beta_3$ with a probability non-negligibly greater than 1/2, and thus can answer the DDH problem with probability non-negligibly greater than 1/2. Thus Oscar has solved DDH in polynomial time, i.e. we have a reduction.

Step 2: We have proven from previous section that STS is a secure mutual authentication protocol with small probability of success (say ε'). Thus the probability of Oscar getting the session key *sk* via impersonation is at most ε' (assuming active attacks of Oscar).

Step 3: Assume that Oscar actively attacks the protocol without impersonation. Then the only way for Oscar to actively attack the protocol is to legitimately participate in the protocol. There are two scenarios to Oscar's participation: (1) Oscar plays the initiator of the protocol and (2) Oscar plays the responder. In scenario (1), Oscar picks a $\beta_1 \in \langle\alpha\rangle$ according to a certain rule, say $\mathcal{RG}_{\mathrm{O}}$ (the word "regel" means rule in German) and sends β_1 to his partner Bob. Bob returns a $\beta_2 =$

α^b where b is chosen by a perfect random number generator (by our assumption). Note that in the proof of STS's security as a key exchange protocol, the third flow is irrelevant.

In Scenario 2, Oscar receives a $\beta_1 = \alpha^a$ from an initiator Alice and returns $\beta_2 \in \langle\alpha\rangle$ according to his rule $\mathcal{RG}_{\mathrm{O}}$.

That means, if Oscar actively attacks, then one of $\beta_1,\ \beta_2$ is chosen by Oscar according to his rule $\mathcal{RG}_{\mathrm{O}}$, and the other one is chosen by a perfect RNG.

However, such transcript tuples can be easily generated by Oscar himself using the following method: Let $T = \{(\beta_1^i, \beta_2^i, sk^i)|i = 1..q\}$ be the set of all q transcript tuples Oscar records through participating legitimately in the protocol. Oscar can simulate T by creating β_1^i according to his rule $\mathcal{RG}_{\mathrm{O}}$, and then using a perfect RNG to get random numbers r_i, and then creating $\beta_2^i \leftarrow \alpha^{r_i}$. The corresponding session key sk^i can be computed as $sk^i = (\beta_1^i)^{r_i}$ (note that β_1^i is of the form α^{r_j} and the session key is supposed to be $\alpha^{r_i r_j}$). If a certain number, t_r, of transcript tuples where Oscar being the responder is needed, then Oscar can pick $Tr = \{(\beta_1^i, \beta_2^i, sk^i)|i = 1 \ldots t_r\}$ and create $Tr' = \{(\beta_1^i\prime, \beta_2^i\prime, sk^i)|\beta_1^i\prime = \beta_2^i\prime, i = 1 \ldots t_r\}$ to replace Tr.

Therefore, if there exists an oracle $\mathcal{O}$ that can take $T,\ \beta_1 = \alpha^{r_1},\ \beta_2 = \alpha^{r_2}$, and output $sk = \alpha^{r_1 r_2}$, then providing $T' = \{(\beta_1^i, \beta_2^i, sk^i)|i = 1 \ldots q\}$ the set of transcript tuples simulated by Oscar (he may be playing both the initiator and the responder) and $\beta_1,\ \beta_2$, the oracle is still able to output sk. Thus participating in the protocol does not give Oscar more advantage to break STS. Therefore, active attacks do not give Oscar more advantage of breaking STS.

From Step 1 to Step 3, STS is a secure protocol under key exchange with success probability $\max\{\varepsilon, \varepsilon'\}$ where ε is the probability of solving DDH, and ε' is the probability of success with passive attacks. □

6.3.5 Discussions and Comments

The progressive approach described in this subsection takes place in the asymmetric setting with side cryptographic tools such as signature schemes, TAs, and certificates. Although similar ideas have appeared in [BR93B], the settings and thus the steps of building a secure authenticated key exchange protocol are different.

Unlike the formalism and general description in [BCK98], the progressive approach focuses on concrete guidelines for building a secure protocol and gives the proof of security. The proof is given in a very intuitive way, yet not sacrificing its completeness and correctness. Although formalism is perhaps the preferred way of

proofs mathematically, the "intuitive" proof serves a better tool for understanding the material and is a better lecture tool for explaining difficult concepts.

Regarding the methodologies of proving security, our approach first limits the adversary's ability to passive attacks and known-session-key attacks by proving the protocol is a secure mutual authentication protocol. Then the second step is to show that the additional ability of known-session-key attack does not enable the adversary to have more effect than attacking passively. This particular concept is similar to the modular approach: where the adversary's ability varies in three different settings, and the ultimate goal is to show that the protocol allows the adversary in the unauthenticated setting to do no more than what it can do in an ideal setting (in that setting the adversary is not powerful).

In this approach, the way of proving the protocol secure against the known-session-key attack is separated from the other attacks; the way of proving security is by way of simulation, proving that whatever information the adversary can retrieve by requesting session keys can be done by passively simulating the process itself. This part does not seem to appear is any other formal models or approaches.

The concept of the progressive approach is to start by having an existing protocol (note, however, that this "existing" protocol may need to be designed from scratch in some cases) and modifying the random challenges without changing too many attributes of the protocol (e.g. the number of flows, amount of overheads, distribution of certain probabilistic processes, etc.). It is a very simple yet effective process, in both theory and practice. However, this approach focuses only on the key exchange process in the asymmetric setting, and it also requires very specific type of mutual authentication protocols and key exchange processes to start with. In a way the variety of protocols produced from this approach may seem very limited. However, after surveying in the literature on the existing protocols for mutual authentication and key exchange, the only secure way of key exchange in practice seems to be Diffie-Hellman (for its straight-forward way of providing forward secrecy), and the only secure way of mutual authentication seems to involve two-way challenge-response using random challenges. In simple words, the secure tools are pretty much limited, thus the limitation of our approach is not a big issue. It is an open question if this approach can scale to more general purposes such as key transport, unilateral authentication, and key distribution.

Note that the resulting protocol is identical to STS-BCK described in the previous section, and thus it also possesses the desired features of a secure AKE protocol.

6.4 Comparison of the Two Approaches

This subsection concludes the above description of the two approaches to designing a secure authenticated key exchange protocol. Some words and discussions may seem redundant as they have appeared in previous subsections, but it is a side-by-side comparison, so the reader may get a more clear understanding of the differences between the two design methodologies. This subsection does not focus on issues of clarity, concreteness, or correctness. It is merely comparing the "effects" provided by the concepts and procedures. Future researchers or practitioners may adapt their preferred approach based on the attributes they need by the information in this subsection.

- **Design Concepts:**
  The modular approach suggests to take a key exchange protocol, proven to be secure in the authenticated setting, then add an authenticator to form an authenticated key exchange protocol in the "real world".

  The progressive approach suggests to start with a secure mutual authentication protocol that uses random challenges, then replace each random challenge with Diffie-Hellman exponentials to become a Diffie-Hellman based authenticated key exchange protocol.

- **Compatibility of Components:**
  In modular approach, any combination of key exchange protocols (secure in the authenticated setting) and authenticators can result in a secure authenticated key exchange protocol (in theory).

  Specific to this version of progressive approach, the mutual authentication protocol must use random challenges in a two-way challenge-response mode. The key exchange protocol applied has to be Diffie-Hellman exponentials or any other similar mechanism that exchanges two random values. In the generalization case (if there is one), then the "nature" of the mutual authentication protocol is tightly bounded to the choices of key exchange mechanisms. It perhaps does not offer a very wide range of compatibilities between the two components of authenticated key exchange protocols.

- **Overheads and the Resulting Protocol:**
  Although the authenticators guarantees the messages transferred in the flows

are authenticated, the resulting protocol seems to contain a large number of protocol flows and overheads. There is no mention in the paper of a clear way to reduce the number of flows or the amount of overheads in the general case. If no general way of reducing the number of flows or the overheads exists, the modular approach may be mainly of theoretical interest over an implementation possibility.

The progressive approach only replaces the two random values (from any domain) with two random Diffie-Hellman exponentials. The number of flows needed in the resulting protocol is the same as the starting mutual authentication protocol, and there is no increased overhead associated with this procedure if we assume the original two random values are picked from a subset of the Diffie-Hellman key space.

- **Ways of Proving Security:**
  In [BCK98], the way of proving security is "modularized"; There are two components of the proof. One is to (1) prove a key exchange protocol is secure in the authenticated links model through emulation, another is to (2) prove that a "compiler" that makes a protocol emulates another protocol is an authenticator. In the above sections, examples of proofs have been demonstrated, and it is thought not difficult to follow the pattern of proofs. More conveniently, if the developed protocols uses a key exchange protocol that's proven to be secure in the authenticated setting already, and the protocol also uses a proven secure authenticator, then there is no proof needed for the resulting protocol.

  Protocols developed via following the progressive approach are rather "systematic". First one needs to prove the security of a mutual authentication protocol, thus the adversary's ability is limited to passive attacks and known-session-key attacks. One then shows the resulting protocol is secure against only passive attacks based on the limitation of adversary. The third step is to show that a passive attacker with the additional ability of known-session-key attack cannot gain more information than being just a passive attacker. For any new combination of mutual authentication protocols and the corresponding key exchange process, a new proof is needed, but the "outline" of the proof is clear and can be easily followed.

Chapter 7

Future Work and Conclusion

7.1 Summary of Thesis

This thesis is focused on the design and features (**Topic**) of authenticated key exchange protocols (**Theme**) between two entities using public-key cryptography (**Setting**).

In this thesis, the desired properties and features of an authenticated key exchange protocol is analyzed through a case study of different variations of the Station-to-Station (STS) protocol. During the case study, many forms of STS are brought up, and an attack is demonstrated to show the insecurity of lacking a certain feature, then another variation of STS is introduced, and so forth. The final version mentioned in the case study, referred to as STS-BCK, is proven secure in [BCK98] and is used in almost all industrial standards of authenticated key exchange in the setting considered in this paper. It is believed that the features we analyzed is sufficient for the **Setting**.

During the survey phase of completing the thesis, it was noticed that there are essentially two approaches to designing a secure authenticated key exchange protocol. The reason for having such little researches into this topic remains unknown at this point, but I do believe having an explicit design will surely connect the theory behind a cryptosystem and the real implementation. Two approaches are analyzed and described in this thesis. One is a previous work by Bellare, Canetti, and Krawczyk; another is suggested by my supervisor, Professor Doug Stinson, during his lectures and our private conversation. These two approaches and their supporting materials are very different, but I have translated the different materials into one common language and showed these two design methodologies both lead

to secure authenticated key exchange protocols by demonstrating the example of STS. I have analyzed these two approaches from several different aspects, including range of application, possibilities of forming practical implementation, and some comments on modelling and ways they approach the definition of security. I believe it is the first work of its kind, and should be carried on further in the future.

7.2 Contribution

- Conducted a complete survey on the development of the STS protocol.
- Presented previous work of attacks and fixes of STS, and compiled a list of desired features of secure STS protocol.
- Studied and presented the modular approach of designing secure AKE protocols in [BCK98].
- Developed an original design methodology to secure AKE protocols. An example of applying such design methodology is given, and the resulting protocol is provided with a proof of security.
- Compared the two design methodologies with respect to design structure, range of application, and ways of proving security, etc.

7.3 Future Work

During the case study, some side issues associated with authentication and key exchange are also looked at. These issues are (1) the security of signature schemes in multi-user settings, (2) definition of mutual authentication and unilateral authentication, and the best choice for authenticated key exchange, and (3) format of certificates and the best time and place to exchange them. I believe these issues are all very interesting and can be potential future work opportunities. Additionally, the study of desired properties can be carried forward to different protocols in different settings and themes.

Regarding the design methodologies, it is the author's desire to see secure authenticated key exchange protocols being implemented and proven secure with less time and more certainty. Some possible future work is (1) creating more design methodologies in different themes (e.g. key transport, key distribution, encryption,

and signatures) and different settings (e.g. distributed computing environment, with timestamps, and ID- or password-based cryptosystems).

The setting used in this thesis resembles the Internet. It is thought to be the most appropriate setting for this research. However, this particular setting may not be the same as real life for many practical reasons, and the setting can only resemble so many real world settings. Issues such as the certification process, distinct public keys for each entity, and cases of multiple certifying/trusted authority can all be looked at in further detail to help provide a better structure of internet security.

Bibliography

[AST98] G. Ateniese, M. Steiner, and G. Tsudik. "Authenticated Group Key Agreement and Friends", *5th ACM Conference on Computer and Communications Security*, pp. 17-26, 1998.

[AST00] G. Ateniese, M. Steiner, and G. Tsudik. "New Multiparty Authentication Services and Key Agreement Protocols". *IEEE Journal on Selected Areas in Communications* volume 18, pp. 628-639, 2000.

[BRK96] M. Bellare, R. Rogaway, and H. Krawczyk, "Keying Hash Functions for Message Authentication", In *Advances in Cryptology: Crypto '96*, LNCS volume 1109, pp. 1-15, 1996.

[BCK98] M. Bellare, R. Canetti, and H. Krawczyk. "A Modular Approach to the Design and Analysis of Authentication and Key exchange Protocols". In Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pages 419-428. ACM Press, 1998. Full version at http://eprint.iacr.org/1998/009.

[BJ02] E. Brier and M. Joye. "Weierstrass Elliptic Curves and Side-Channel Attacks". In *Public Key Cryptography*, LNCS volume 2274, pp. 335-345, Springer, 2002.

[BK00] J. Baek and K. Kim, "Remarks on the Unknown Key-Share Attacks". *IEICE Trans. Fund.* E83-A, 12 (Dec.), pp. 2766-2769, 2000.

[BM84] M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-random Bits", *SIAM Journal on Computing*, volume 13, pp. 850-864, 1984.

[BM] C. Boyd and A. Mathuria, "Protocols for Authentication and Key Establishment", Springer, 2003.

[BR93A] M. Bellare and P. Rogaway, "Random Oracles are Practical: a Paradigm for Designing Efficient Protocols", *Proceedings of the First ACM Conference on Computer and Communications Security*, pp. 62-73, 1993.

[BR93B] M. Bellare and P. Rogaway. "Entity Authentication and Key Distribution". *Proceedings of Advances in Cryptology — CRYPTO'93*, LNCS volume 773, pp. 232-249. Springer-Verlag, 1993.

[BW99] S. Blake-Wilson, "Key Establishment Protocols and the Diffie-Hellman Problem", 1999, http://www.cacr.math.uwaterloo.ca/conferences/1999/ecc99/blake-wilson.ps

[BWJM97] S. Blake-Wilson, D. Johnson and A. Menezes, "Key Agreement Protocols and their Security Analysis", *Proceedings of the sixth IMA International Conference on Cryptography and Coding*, LNCS volume 1355, pp. 30-45, 1997. A full version of this paper is available at http://www.cacr.math.uwaterloo.ca

[BWM97] S. Blake-Wilson and A. Menezes, "Entity Authentication and Key Transport Protocols Employing Asymmetric Techniques", *Security Protocols Workshop*, pp. 137-158, 1997.

[BWM98] S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman Key Agreement Protocols", *Proceedings of SAC '98*, LNCS volume 1556, pp. 339-361.

[BWM99] S. Blake-Wilson and A. Menezes, "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol", *Proc. of PKC '99*, LNCS volume 1560, pp. 154-170, Springer-Verlag, 1999.

[CK01] R. Canetti and H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels". *Eurocrypt 2001*, LNCS volume 2054, pp. 453-474

[DOW92] W. Diffie, P.C.van Oorschot, and M.J,Wiener, "Authentication and Authenticated Key Exchanges", *Designs, Codes, and Cryptography*, volume 2, pp. 107-125. 1992.

[GM84] S. Goldwasser and S. Micali, "Probablistic Encryption", *Journal of Computer and System Science*, volume 28, pp. 270-299, 1984.

[GMR84] S. Goldwasser, S. Micali, and R. Rivest, "A "Paradoxical" Solution to the Signature Problem", *Proceedings of the IEEE 25th Annual Symposium on Foundations of Computer Science*, pp. 441-448, 1984

[GMR88] S. Goldwasser, S. Micali, and R. Rivest, "A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks", *SIAM Journal on Computing*, volume 17, No. 2, pp. 281-308, April 1988.

[Has00] M. A. Hasan. "Power Analysis Attacks and Algorithmic Approaches to their Countermeasures for Koblitz Cryptosystems". In *Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS volume 1965, pp. 93-108, 2000.

[HC98] D. Harkins and D. Carrel. "The Internet Key Exchange (IKE)", RFC 2409, 1998.

[IEEE00] IEEE. *P1363 Standard Specifications for Public-Key Cryptography*, IEEE Standard, pp. 1363-2000, January 2000.

[JSI96] M. Jakobsson, K. Sako, and R. Impagliazzo. "Designated Verifier Proofs and their Applications". *EUROCRYPT '96*, LNCS volume 1233, 1996.

[Kal00] B. Kaliski, "Key Management and ANSI X9.44", http://csrc.nist.gov/encryption/kms/x944-slides.pdf, Febuary 10th, 2000.

[KM04] N. Koblitz and A. J. Menezes, "Another Look at 'Provable Security'", Technical Report CORR 2004-20, University of Waterloo, 2004. An online version is at http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/provable.pdf.

[KOY02] J. Katz, R. Ostrovsky, and M. Yung. "Forward Secrecy in Password-only Key Exchange Protocols". *Proceedings of Security in Communication Networks '02*, LNCS volume 2576, pp. 29-44, 2002.

[Kra96] H. Krawczyk. "SKEME: A Versatile Secure Key Exchange Mechanism for Internet", *Symposium on Network and Distributed System Security*, pp. 114-127, 1996.

[LL95] Chae Hoon Lim and Pil Joong Lee. "Several Practical Protocols for Authentication and Key Exchange", *Information Processing Letters*, volume 53, pp. 91-96, 1995.

[LMQSV03] L. Law, A. Menezes, M. Qu, J. Solinas, S. Vanstone. "An Efficient Protocol for Authenticated Key Agreement", *Designs, Codes and Cryptography* volume 28, pp. 119-134, 2003

[Lowe96] G. Lowe. "Some New Attacks upon Security Protocols". *Proceedings of the 9th IEEE Computer Security Foundations Workshop (CSFW9)*, pp. 162-169. IEEE Computer Society Press, 1996.

[Mao] Wenbo Mao, "Modern Cryptography - Theory & Practice", Pearson Education, 2004.

[MOV] A. J. Menezes, P. C. van Oorchot, and S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996.

[MQV95] A. J. Menezes, M. Qu, and S. A. Vanstone. "Some New Key Agreement Protocols Providing Implicit Authentication". *Workshop on Selected Areas in Cryptography (SAC '95)*, pp. 22-32, 1995.

[MS04] A. Menezes and N. Smart, "Security of Signature Schemes in a Multi-User Setting", *Designs, Codes, and Cryptography*, volume 33, pp. 261-274, August 2004.

[OP03] E. Oswald and Bart Preneel. "A Survey on Passive Side-Channel Attacks and their Countermeasures for the NESSIE Public-Key Cryptosystems". NESSIE public reports, https://www.cosic.esat.kuleuven.ac.be/nessie/reports/, 2003.

[Orm98] H. Orman. "The OAKLEY Key Determination Protocol". RFC 2412, 1998.

[RH93] A. Rubin and P. Honeyman. "Formal Methods for the Analysis of Authentication Protocols". Technical Report 93-97, CITI, Information Technology Division, University of Michigan, 1993. http://cs.nyu.edu/?rubin/November.

[RS84] R. L. Rivest and A. Shamir, "How to Expose an Eavesdropper", *Communications of the ACM*, volume 27 No. 4, pp. 393-395, April 1984.

[RS91] C. Rackoff and D. Simon, "Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack", *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, LNCS volume 576, pp. 433-444, 1992.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems". *Communications of the ACM* volume 21, pp. 120-126, 1978.

[Sho99] V. Shoup, "On Formal Models for Secure Key Exchange", version 4, Revision of IBM Research Report RZ 3120 (April 1999), November 15, 1999.

[StiA] D. Stinson, CS758 Lecture Notes, School of Computer Science, University of Waterloo, 2003. http://www.cacr.math.uwaterloo.ca/~dstinson/CS_758/2003/slides.html.

[StiB] D. Stinson, "Cryptography Theory and Practice - Second Edition", Chapman & Hall/CRC, 2002.

[Yac91] Y. Yacobi. "A Key Distribution 'Paradox'", *Advances in Cryptography - Crypto '90*, LNCS volume 537, pp. 268 - 273, 1991.

[Yao82] A. C. Yao, "Theory and Applications of Trapdoor Functions", *Proceedings of the Twenty Third Annual Symposium on the Fundations of Computer Science*, IEEE, pp. 80-91, 1982.