國立交通大學

資訊科學系

碩 士 論 文

有效減少通訊回合數且向前安全的會議金鑰建立協定

Round-Efficient Conference Key Agreement Protocols with Forward Secrecy

研 究 生：李振魁

指導教授：曾文貴 教授

中 華 民 國 九 十 三 年 六 月

有效減少通訊回合數且向前安全的會議金鑰建立協定

Round-Efficient Conference Key Agreement Protocols with Forward Secrecy

研 究 生：李振魁　　Student ：Chen-Kuei Lee

指導教授：曾文貴　　Advisor：Dr. Wen-Guey Tzeng

國 立 交 通 大 學
資 訊 科 學 系
碩 士 論 文

A Thesis
Submitted to Department of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer and Information Science

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

Round-Efficient Conference Key Agreement Protocols with Forward Secrecy

Student: Chen-Kuei Lee   Advisor: Dr. Wen-Guey Tzeng

Department of Computer and Information Science

National Chiao Tung University

Abstract

A conference key agreement protocol allows a group of participants to establish a common secret key distributively, such that all their communications afterward are encrypted by the key. By this way, the participants can communicate securely over an open network. We propose two provably forward secure conference key agreement protocols under the broadcast channel model. Also, we prove its security under the Bellare-Rogaway model. The adversary that attacks our protocols can be either passive or active. A passive adversary tries to learn the conference key by listening to the communication of participants, while an active adversary tries to impersonate as a legal participant or disrupt conference key establishment among the honest participants. Further, in our protocol, we would like to focus on both round efficiency and forward secrecy.

Key words: Conference Key, Round-efficient, Forward-secure

有效減少通訊回合數且向前安全的會議金鑰建立協定

學生：李 振 魁　　　　　　　　　　　　指導教授： 曾 文 貴　博士

國立交通大學資訊科學學系（研究所）碩士班

摘　要

當一群使用者想要在公開的網路上安全的召開會議、傳送訊息時，他們需要一把共享的金鑰來對所傳送的訊息加密，以免遭到竊聽。而會議金鑰建立協定，就是用來建立此一共享金鑰的方法。

在金鑰建立的過程中，我們須確保其正確性及隱密性。在有部分惡意參與者從中傳送不正確訊息的情況下，其它的參與者仍要可以正確的建立金鑰。同時我們也保證，不合法的使用者無法從金鑰建立的過程中所交換的訊息，得知會議的金鑰。此外，我們希望會議金鑰的建立具有向前安全的性質，也就是若使用者的私密金鑰遭到竊取，並不會影響到之前所建立的會議金鑰的安全性。除了正確、安全之外，金鑰建立時的效率也是很重要的考量，所以我們希望能儘量減少其通訊的回合數。

因此在本篇論文中，我們提出了兩個能有效減少通訊回合數且具向前安全性質的會議金鑰建立協定，並且完整的證明其安全性。

關鍵詞：會議金鑰、有效減少回合數、向前安全

誌　　謝

在此感謝我的指導老師曾文貴教授，在我碩士班兩年的學習過程中，不只讓我在學業上受益良多，更在生活上以及言行上給我許多教導。此外，我要感謝口試委員，交大資工系蔡錫鈞教授和清大資工系孫宏民教授，在論文上給予我許多良好的建議和指導，讓我的論文更加完善。除此之外我要感謝實驗室同學，尚宸、兆儀、坤杉和佩琳的幫忙，實驗室學長成康、惠龍和季穎學姊的指導，以及實驗室學弟妹們在精神方面的鼓勵。

最後，我要感謝我的家人，不論在精神或物質上都給予我極大的支持，讓我在無後顧之憂的情況下可以順利完成學業。在此，謹以此文獻給我所有我想要感謝的人。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A conference key protocol allows a group of participants to establish a common secret key such that all their communications afterward are encrypted by the key. Consequently, the participants can communicate securely over an open network. The conference key protocol can be broadly divided into key distribution and key agreement protocols [27]. In key distribution protocols, a key is selected by a chairman and then securely transmitted to the other participants. In key agreement protocols, all participants contribute information to compute a common shared key.

In this thesis, we will propose two provably forward secure conference key agreement protocols under the broadcast channel model, which assures all sent messages can be received intact. The adversary that attacks our protocols can be either passive or active. A passive adversary (eavesdropper) tries to learn the conference key by listening to the communication of participants, while an active adversary (impersonator and malicious participant) tries to impersonate as a legal participant or disrupt conference key establishment among the honest participants.

Besides, communication efficiency of a conference key protocol is also an important issue. It usually concerned with the number of messages that been sent and received during a protocol, and the number of rounds in the protocol. In our protocols, we would like to focus on both round efficiency and forward secrecy.

1.1 Motivation

To communicate with other people over the network has become a trend due to the conveniences and economic benefits provided by Internet. Many people start to exchange messages or hold conferences over the network so that the participants have a long distance relationship can communicate with each other easily. There are already many applications provide such service, like the Internet Relay Chat (IRC), NetMeeting, or MSN Messenger. But these applications usually use a centralized server to control or forward the messages during the meeting, therefore every participant must connect to the server to join the conference. Once the server fails, all conferences will be interrupted.

To solve this disadvantage, we may want a distributed approach, such that the conference can be held without much help from the server. Participants can use the existential Internet infrastructure to broadcast the messages over the open network, instead of forwarding the messages by a single server. But in this case, everyone joins the multicast group may receive the broadcasted messages. If we want to provide the security and privacy of the conference, we must use some techniques to encrypt the messages sent over the network.

The Conference Key Agreement Protocol can provide such functionality by distributively generating a secret key from the participants of the conference.

In the Conference Key Agreement Protocol, the secret key used to encrypt the messages is contributed by every participant, instead of being designated by a central server or chairman. Thus, no participant can influence the final secret key. However, we must avoid the case that if some participants want to break the establishing process of the secret key by sending malicious message, so we include the concept of Publicly Verifiable Secret (PVS), which is a zero knowledge proof system, and can be used to provide checking for message consistency in our protocol.

1.2 Previous Works

CONFERENCE KEY AGREEMENT. There have been a lot of researches on conference key agreement protocols. Most of these protocols are based on generalization of Diffie and Hellman's famous key exchange protocol [16]. For instance, Ingemarsson, Tang and Wong [21] give a set of protocols, and Steiner, Tsudik and Waidner [29] also propose three protocols. None of their basic protocols provide authentication of the participants. Thus, these protocols are not secure against active attacks. Though Ateniese *et al.* [1, 2] propose two methods to make one of the protocols of Steiner *et al.* provide authenticated group key agreement, Pereira and Quisquater [28] have described a number of potential attacks.

Burmester and Desmedt [13] proposed a round-efficient protocol which provides forward secrecy and costs only two rounds to establish the conference key. However, their protocol can not resist the attack of malicious participants. Later, Just and Vaudenay [24] modified the protocol in [13] to provide authentication, and recently Choi *et al.* [15] transform the protocol in [13] into ID-based version which works in elliptic curve groups. The protocol of Joux [23] is the only currently known group key agreement protocol that can be completed in a single round and still provide forward secrecy, but their protocol can only work with three parties. In terms of fault tolerance, most proposed protocols except [26, 30] do not have this capability, so a malicious participant can easily spoil the conference by making other participants to compute different conference key.

Provable security for protocols. Another important contribution in cryptographic protocol research is the first mathematical security proof of a simple entity authentication protocol proposed by Bellare and Rogaway [4]. Though their work discuss only the two-party case, many authors extend the same idea to include public-key base key transport [6], key agreement protocol [7], password-based protocol [3, 9], and conference key protocols [12, 10, 11].

1.3 Organization

The remainder of this thesis is organized as follows. The next chapter reviews some preliminaries and basic techniques. Chapter 3 describes the communication model and the security model, and defines security properties of a conference key agreement protocol. Chapter 4 gives a formal proof of a protocol proposed by Tzeng and Tzeng based on the security definitions of Bellare and Rogaway, and also presents two new forward secure conference key agreement protocols with their proofs. Finally, chapter 5 gives the comparison of new protocols with existed one, and then concludes.

Chapter 2

Preliminaries

In this chapter, we would like to introduce some assumptions that support the security of our protocols. We also describe the general notations of an encryption scheme, signature scheme, and forward secure version of these schemes. Finally, we review the concept of zero knowledge proof system. Then use this tool to construct the Publicly Verifiable Secret (PVS) protocol, which we will use in our protocol.

2.1 Assumptions

We will remind two algorithmic assumptions in this section — Discrete Logarithm Assumption (DLA) and Decisional Diffie-Hellman Assumption (DDHA). We use the following setting for these two assumptions:

– p : a large prime number that is $2q + 1$, where q is also a large prime.

– g : a generator for the subgroup G_q of all quadratic residues in Z_p^*.

– $x \in_R S$ denote that x is chosen from the set S uniformly and independently.

Discrete Logarithm Assumption (DLA)

The discrete logarithm (DL) problem is to compute $x \equiv \log_g y \pmod p$ from given (y, g, p), where $p = 2q + 1$, g is a generator of G_q and $y \in_R G_q$. In general, we assume that DL problem is computationally infeasible. Thus, we have the following formal description of DLA.

Assumption 1 (Discrete Logarithm Assumption)
There is no probabilistic polynomial time algorithm that can solve any significant portion of instances of $x \equiv \log_g y \pmod p$, where $p = 2q + 1$, p and q are both prime, g is a generator for the subgroup G_q of all quadratic residues in Z_p^* and $y \in_R G_q$. That is, assume R_n be the set of n-bit prime $p = 2q + 1$, for any probabilistic polynomial time algorithm A, for any large enough prime n, for any $k > 0$,

$$\Pr_{p \in R_n,\, g, y \in G_q} \left[\, A(y, g, p) = \log_g y \bmod p \,\right] \leq 1/n^k.$$

Decisional Diffie-Hellman Assumption (DDHA)

The Decisional Diffie-Hellman (DDH) problem is to distinguish the following two probability ensembles $R = \{R_n\}$ and $D = \{D_n\}$,

– $R_n = (g, p, g^x \bmod p, g^y \bmod p, g^z \bmod p)$, where

  - p is a randomly chosen n-bit prime with $p = 2q + 1$ and q is also a prime.
  - g is a randomly chosen generator of order-q subgroup G_q of Z_p^*.
  - x, y, z are chosen uniformly and independently from Z_q^*.

– $D_n = (g, p, g^x \bmod p, g^y \bmod p, g^{xy} \bmod p)$, where

  - p is a randomly chosen n-bit prime with $p = 2q + 1$ and q is also a prime.
  - g is a randomly chosen generator of order-q subgroup G_q of Z_p^*.
  - x and y are chosen uniformly and independently from Z_q^*.

In cryptology, we assume that probability ensembles R and D are not polynomially distinguishable. Therefore, we have the following assumption.

Assumption 2 (Decisional Diffie-Hellman Assumption)
Let $p = 2q + 1$, p and q are both primes, g is a generator for the subgroup G_q of all quadratic residues in Z_p^* and $x, y, z \in_R G_q - \{1\}$. Then the following two random-variable tuples $D_n = (g, p, g^x, g^y, g^{xy})$ and $R_n = (g, p, g^x, g^y, g^z)$ are computationally indistinguishable. That is, for any probabilistic polynomial time algorithm A, for any large enough prime n, for any $k > 0$,

$$|\Pr[A(R_n) = 1] - \Pr[A(D_n) = 1]| \leq 1/n^k.$$

2.2 Basic Schemes

In the new protocol that we will present later, we reduce its security to the underlaying public key encryption and signature schemes. So we describe the general notations of an encryption scheme, signature scheme, and forward secure version of these schemes here.

Secure Encryption Scheme

Let k be the security parameter. A public key encryption scheme $\mathcal{C} = (\texttt{E.Gen, E.Enc, E.Dec})$ consists of three algorithms.

– The key generation algorithm $\texttt{E.Gen}$ is a polynomial time probabilistic algorithm, which on input 1^k, output a pair (e, d) of matching public and private keys, respectively.

– The encryption algorithm $\texttt{E.Enc}(\cdot)$ is a polynomial time probabilistic algorithm, which takes a public key e and a message m chosen from a message space M associated to e, and returns a ciphertext c. We denote this as $c \leftarrow \texttt{E.Enc}(e, m)$.

– The decryption algorithm $\texttt{E.Dec}(\cdot)$ is a polynomial time deterministic algorithm, which takes a private key d and a ciphertext c, and returns the corresponding plaintext m. We denote this as $m \leftarrow \texttt{E.Dec}(d, c)$ and assume $\texttt{E.Dec}(d, \texttt{E.Enc}(e, m)) = (m)$ for every $(e, d) \leftarrow \texttt{E.Gen}(1^k)$.

We use the security definition called semantic security that proposed by Goldwasser and Miicali [18]. For any probabilistic polynomial time adversary

$\mathcal{A}$, he plays the `IND-CCA` game with the challenger. We define the advantage of the adversary playing the `IND-CCA` game as $\mathtt{Adv}^{\mathcal{A}}(k) = 2\Pr[b' = b] - 1$. We say that the encryption scheme $\mathcal{C}$ is secure if the adversary's advantage is negligible.

Secure Signature Scheme

Let k denote the security parameter. A digital signature scheme $\mathcal{S} = (\mathtt{S.Gen}, \mathtt{S.Sig}, \mathtt{S.Ver})$ consists of three algorithms.

- The key generation algorithm `S.Gen` is a polynomial time probabilistic algorithm, which on input 1^k, output a pair (e, d), where e is the (public) verification key and d is the corresponding (private) signing key.
- The signing algorithm $\mathtt{S.Sig}(\cdot)$ is a polynomial time probabilistic algorithm, which takes a signing key d and a message m chosen from a message space M, and outputs a signature σ. We denote this as $\sigma \leftarrow \mathtt{S.Sig}(d, m)$.
- The verification algorithm $\mathtt{S.Ver}(\cdot)$ is a polynomial time deterministic algorithm, which takes a verification key e, a message m and its corresponding signature σ, and outputs 1 if the signature is valid, otherwise outputs 0. We assume that $\mathtt{S.Ver}(e, m, \mathtt{S.Sig}(d, m)) = 1$ for every $(e, d) \leftarrow \mathtt{S.Gen}(1^k)$.

We say a signature scheme is secure if it is computationally infeasible for any adversary to forge a signature on any message (existential forgery) even under adaptive chosen-message attacks [20].

Forward Secure PKI

Forward Secure Encryption Scheme Let k denote the security parameter, N be the total number of time periods. A public-key key-evolving encryption scheme $\mathcal{FE} = (\texttt{FE.Gen}, \texttt{FE.Upd}, \texttt{FE.Enc}, \texttt{FE.Dec})$ consists of four algorithms.

– The key generation algorithm `FE.Gen` is a polynomial time probabilistic algorithm, which on input 1^k and N, output a public key PK and an initial secret key SK_0.

– The key update algorithm $\texttt{FE.Upd}(\cdot)$ is a polynomial time probabilistic algorithm, which takes a public key PK and an index $i < N$ of the current time period, and the associated secret key SK_i, and returns the secret key SK_{i+1} for the following time period. This is denoted as $SK_{i+1} \leftarrow \texttt{FE.Upd}(PK, i, SK_i)$

– The encryption algorithm $\texttt{FE.Enc}(\cdot)$ is a polynomial time probabilistic algorithm, which takes a public key PK, an index $i \leq N$ of a time period, and a message m, and returns a ciphertext c. We denote this as $c \leftarrow \texttt{FE.Enc}(PK, i, m)$.

– The decryption algorithm $\texttt{FE.Dec}(\cdot)$ is a polynomial time deterministic algorithm, which takes a public key PK, an index $i \leq N$ of the current time period, the associated secret key SK_i, and a ciphertext c, returns the corresponding plaintext m. This is denoted as $m \leftarrow \texttt{FE.Dec}(PK, i, SK_i, c)$, and we assume that $\texttt{FE.Dec}(PK, i, SK_i, \texttt{FE.Enc}(PK, i, m)) = (m)$ for any index $i \in [0, N)$, and for every $(PK, SK_0) \leftarrow \texttt{FE.Gen}(1^k, N)$.

We use the security notion of forward-secure against chosen-ciphertext attacks (`fs-CCA`) proposed in [14]. The advantage of the adversary playing the `fs-CCA` game is defined as $\texttt{Adv}^{\mathcal{A}}(k) = 2\Pr[b' = b] - 1$. We say that the public-key key-evolving encryption scheme $\mathcal{FE}$ is secure if the adversary's advantage is negligible.

Forward Secure Signature Scheme Let k denote the security parameter, N be the total number of time periods. A public-key key-evolving digital signature scheme $\mathcal{FS} = (\texttt{FS.Gen}, \texttt{FS.Upd}, \texttt{FS.Sig}, \texttt{FS.Ver})$ consists of four algorithms.

– The key generation algorithm `FS.Gen` is a polynomial time probabilistic algorithm, which on input 1^k and N, and outputs a public key PK and an initial secret key SK_0.

– The key update algorithm $\texttt{FE.Upd}(\cdot)$ is a polynomial time probabilistic algorithm, which takes an index $i < N$ of the current time period, and the associated secret key SK_i, and returns the secret key SK_{i+1} for the following time period. This is denoted as $SK_{i+1} \leftarrow \texttt{FS.Upd}(i, SK_i)$

– The signing algorithm $\texttt{FS.Sig}(\cdot)$ is a polynomial time probabilistic algorithm, which takes a signing key SK_i, an index $i \leq N$ of a time period, and a message m, and returns a signature σ for time period i. We denote this as $(i, \sigma) \leftarrow \texttt{FS.Sig}(SK_i, i, m)$.

– The verification algorithm $\texttt{FS.Ver}(\cdot)$ is a polynomial time deterministic algorithm, which takes a public key PK, a candidate signature (i, σ), and a message m, then outputs 1 if the signature is valid, otherwise outputs 0. We assume that $\texttt{FS.Ver}(PK, m, \texttt{FS.Sig}(SK_i, m)) = 1$ for every message m and time period $i \in [0, N)$.

2.3 Zero Knowledge Proof System

In a Conference Key Agreement Protocol, since we can not assume that all participants are honest, we must provide some methods to avoid the malicious participants sending invalid messages to interfere the key agreement procedure. The concept of zero knowledge proof system proposed by Goldwasser *et al.* [19] can achieve this goal. Zero knowledge proofs are proofs that gives a conviction and reveals nothing about the validity of the assertion being proven. It must satisfy the properties: completeness, soundness, and zero knowledge. Further, we can use this tool to construct a Publicly Verifiable Secret (PVS) protocol, which one user can send a secret to the other participants while everyone can verify that all participants receive the same secret. In this section, we review the PVS protocol presented by Tzeng and Tzeng [31], and give a more general form of PVS protocol that use any secure encryption scheme.

Publicly Verifiable Secret Protocol (PVS)

Assume that (x_i, y_i) is the private and public key pair of participant U_i. If participant U_i wants to send to secret $g_i^k \bmod q$ to all the other participants in a public verifiable way, he broadcasts $u_{i,j} = y_j^{k_i} \bmod p$, for $i \leq j \leq n$, where $k_i \in_R Z_q$. Another participant U_j can obtain the shared secret $g^{k_i} \bmod p$ from

U_i by computing $(U_{i,j})^{x_j^{-1}} \bmod p$. The PVS proof system shows that

- $\log_{y_1} u_{i,1} \equiv \log_{y_2} u_{i,2} \equiv \cdots \equiv \log_{y_n} u_{i,n} \pmod{p}$, and
- U_i knows the exponent $k_i = \log_{y_j} u_{i,j} \pmod{p}$, for $1 \leq j \leq n$.

with negligible error probability $1/2^t$ according to security parameter t. The PVS proof system is:

1. $P \rightarrow V : b_j = y_j^r \bmod p, 1 \leq j \leq n$, where $r \in_R Z_q$;
2. $V \rightarrow P : c \in_R [0..2^t - 1]$;
3. $P \rightarrow V : w = r - ck_i \bmod q$;
4. V checks whether $b_j = y_j^w \cdot u_{i,j}^c \bmod p, 1 \leq j \leq n$.

Theorem 1 ([31]) *Assume the DLA. The PVS proof system above is complete, sound and zero knowledge.*

Non-interactive PVS

We want a proof system to provide fault tolerance in our protocol, but for efficiency we want it to be non-interactive. We give some basic ideas of the non-interactive proof system in the following. In a non-interactive proof system, the prover P produces a string to meet all the properties of an interactive proof system without interacting with the verifier V. Hence, we need a collision resistant hash function $\mathcal{H}$ to replace the verifier's role in the original interactive proof system (i.e. generating the challenge c). We achieve this goal by applying well known technique proposed by Feige *et al.* [17]. We describe the non-interactive PVS (NIPVS) used later as follows:

– The prover U_i randomly selects $r \in Z_q$, and computes

$c = \mathcal{H}(g \| y_1 \| \cdots \| y_n \| u_{i,1} \| \cdots \| u_{i,n} \| y_1^r \| \cdots \| y_n^r)$,

where $\|$ is the concatenation operation of strings.

– The prover U_i sets $w = r - ck_i$, and sends (c, w) as his proof.

– The verifier checks (c, w) sent by U_i for NIPVS satisfies

$c = \mathcal{H}(g \| y_1 \| \cdots \| y_n \| u_{i,1} \| \cdots \| u_{i,n} \| y_1^w u_{i,1}^c \| \cdots \| y_n^w u_{i,n}^c)$,

then he can assure that $\log_{y_1} u_{i,1} \equiv \log_{y_2} u_{i,2} \equiv \cdots \equiv \log_{y_n} u_{i,n} \pmod p$, which means all participants receive the same secret g^{k_i}.

Zero Knowledge Proof for any NP Problem

Based on standard intractability assumptions, it is already known how to construct a non-interactive zero knowledge proof for any NP-set [25]. Thus, we can assume that we can find a non-interactive zero knowledge proof for the following problem.

More generally, if we assume that the secure encryption scheme exists, and again, if the participant U_i wants to send the secret value $g_i^k \bmod q$ to all the other participants in a public verifiable way. Then he broadcasts $u_{i,j} = \texttt{Enc}(y_j, g^{k_i})$, for $i \leq j \leq n$, where $k_i \in_R Z_q$. We can use the PVS proof system to shows that

– $\texttt{Dec}(y_1, u_{i,1}) \equiv \texttt{Dec}(y_2, u_{i,2}) \equiv \cdots \equiv \texttt{Dec}(y_n, u_{i,n})$, and

– U_i knows the secret $g^{k_i} = \texttt{Dec}(y_j, u_{i,j})$, for $1 \leq j \leq n$.

And, we use NIP(Consistence of g^{k_i}) to denote this proof system in our protocol.

Chapter 3

Our Models

We introduce the communication model and the security model used in our protocol as well as the precise security requirement of a conference key agreement protocol as follows.

3.1 Communication Model

The communication model we use later for distributed security was first proposed by Bellara and Rogaway [4, 5], who give a formal specification on entity authentication and authenticated key distribution protocols. We will use its refined version [3], which is more suitable for the multi-party environment.

Protocol Participants. In this communication model, we have a finite and nonempty set $ID = \{U_1, \ldots, U_N\}$ of all users in system, and the total number of users N is polynomial in the security parameter k. Each user has an unique identifier U from the set ID, where user $U \in ID$ is named by a fixed length string, and a group of users who want to establish a conference key is called the set of participants.

COMMUNICATION ENVIRONMENT. In our communication model, all users are connected by a broadcast network, which is an unauthenticated broadcast channel, and there is no private channel existed between users. All messages sent cannot be altered, blocked or delayed, that is, the adversary faithfully relays flows between participants. Nevertheless, the attacker can inject malicious messages. For simplicity, we assume that the network is fully synchronous, which means all users send their messages to the other recipients (or receive messages from the other senders) simultaneously in a single round.

LONG-LIVED KEYS. Each user in system has a long-lived secret key, and a corresponding public key, obtained at the beginning of the protocol using a key distribution algorithm `Gen`. The system also has a public directory that can be accessed by everyone, which contains the system's public parameters and each user's public key.

3.2 Security Model

For security, we assume that all communication among interacting parties are controlled by the adversary. The main idea is to model instances of users via oracles available to the adversary, modeling various kinds of attacks by appropriate queries to these oracles, having some notion of partnering, and requiring semantic security of the session key via `Test` queries.

ADVERSARY. The adversary $\mathcal{A}$ is a probabilistic polynomial-time Turing machine that controls all the communications during the protocol runs, and does this by interacting with a set of oracles. Oracle Π_U^s represents the actions of participant U in the protocol run indexed by instance s, each participant may run many instances at the same time, and interactions with the adversary are called oracle queries. We now explain each query that is available to the adversary, and summarize it in Table 3.1.

(1) $\texttt{Send}(U, s, m)$ — This query allows the adversary to send message m to oracle Π_U^s. The oracle runs the protocol normally, and sends back the response. If the received message m is not of the expected format, the oracle may simply halt. Otherwise, the adversary can know whether the oracle accepts the session key or not, as well as the session ID and the partner ID. The adversary can use this query to initiate a new protocol instance by sending a special message m=$\texttt{Init}$ to a participant. This query models the possibility of an adversary $\mathcal{A}$ causing an instance to come into existence in the real-world, for that instance to receive communications faked by $\mathcal{A}$, and to respond what a honest participant does in protocol.

$\texttt{Send}(U, s, m)$	Send message m to oracle Π_U^s
$\texttt{Reveal}(U, s)$	Reveal session key accepted by Π_U^s
$\texttt{Corrupt}(U)$	Reveal the long-lived secret key hold by U
$\texttt{Test}(U, s)$	Ask a challenge to distinguish session key accepted by Π_U^s

Table 3.1: Queries available to the adversary

(2) `Reveal`(U, s) — This query models the adversary's ability to get session keys. In real-world, the session key might be lost for many kinds of reasons, like hacking or cryptoanalysis, thus loss of a session key should not be damaging to other sessions. If an oracle Π_U^s has accepted, holding some session key sk, then this query returns sk to the adversary. We call an oracle is *opened* if it has been the object of a `Reveal` query.

(3) `Corrupt`(U) — This query tries to model the insider attacks by adversary, as the dishonest participant tries to disrupt the process of key agreement in real-world. This query returns the oracle's long-lived key to the adversary, thus the adversary can then control the behavior of participant U. We call a participant is *corrupted* if it has been the object of a `Corrupt` query.

(4) `Test`(U, s) — Once the oracle Π_U^s has accepted, holding a session key sk, then the adversary can ask for a challenge to distinguish sk from a random key. The oracle will flips a coin b, if $b = 1$ then sk is returned, otherwise a random string drawn from the same distribution as session key is returned. This query is asked just once by the adversary.

(5) $h(m)$ — Finally, this is a collision-resistant ideal hash function, which is used in random oracle model. Not only the adversary, but the protocol and the long-lived key generator may depend on this hash function. To avoid the replay attack, we always compute it involve with session ID *SID* (or session token ST).

ORACLE PARTNERING. There are various ways to define partner oracles in Bellare-Rogaway model. In this thesis, we use the adaptations from [3]. Fix a protocol P, an adversary $\mathcal{A}$, and during the protocol execution, we say that oracles Π_U^i and $\Pi_{U'}^{i'}$ are partnered if both oracles accepted, holding the same session key, session ID and partner ID. In our protocol, we assume that the partner ID is the concatenation of each user's identifier U in set of participants $\mathcal{U}$. Thus, we define the partnering of a set of oracles formally as follow.

Definition 3 *A set of oracles are partnered if the following conditions hold:*

- *They agree on the same set of participants* $\mathcal{U} \subseteq ID$.
- *They have accepted with the same sk, SID and PID.*

Oracle Status. As we mention above, we call an oracle is *opened* if it has been the object of a `Reveal` query, and we call an oracle is *corrupted* if it has been the object of a `Corrupt` query. Besides, during the protocol runs, an oracle may *accept* at any time, which means the oracle has hold a particular session key (sk), session ID (*SID*) and partner ID (*PID*). The session key sk is used to protect the following communication during conference. The *SID* is an identifier which can be used to uniquely name the sequence of conference session established by a participant, while the *PID* names the set of participants which the instance believes it has just communicate with. The *SID* and *PID* are not secret, so the adversary can know these information. Oracle also has a status called *terminate*, which means oracle has what it wants, and won't send any further messages. An instance may wish to accept now, and terminate later. As in real-world, a participate believes he is now holding a correct session key, but before using that key, he may want to wait for a confirmation message to terminate.

Freshness. We have two notions of freshness — with and without forward secrecy, both depend on the status of oracle. Their formal definition are as follows.

Definition 4 (Basic Freshness) *We say that an oracle Π^s_U is fresh at end of its execution if:*

- *Π^s_U has accepted with set of participants Π^*.*
- *Π^s_U and all other oracles in Π^* are unopened.*
- *All participants within Π^* are uncorrupted.*

Definition 5 (Freshness with forward secrecy) *We say that an oracle Π^s_U is fs-fresh at end of its execution if:*

- *Π^s_U has accepted with set of participants Π^*.*
- *Π^s_U and all other oracles in Π^* are unopened.*
- *All participants within Π^* are uncorrupted before `Test` query.*

Security. We define the security of the protocol by the following game played by adversary $\mathcal{A}$ and a set of oracles Π^s_U for some $\mathcal{U} = \{U_1, \ldots, U_n\}$. At first, the key generation function `Gen` will assign the long-lived private key to each user and publish the system security parameters, as well as all user's public key. Then adversary $\mathcal{A}(1^k)$ starts interacting with oracles and making any queries of `Send`, `Reveal`, or `Corrupt`. At some stage during execution, $\mathcal{A}$ does a `Test` query on a *fresh* (or *fs-fresh*) oracle Π^s_U to get a return challenge sk'. Then the adversary may continue to make other queries. Finally, he terminates and outputs a bit b'. If the adversary guesses that sk' is the corresponding session key which Π_U^s is involved, then outputs $b' = 1$, else outputs $b' = 0$, and we say that the adversary wins the game if $b' = b$. Assume $\texttt{Success}$ be the event that $\mathcal{A}$ wins the game, his advantage is $\texttt{Adv}^{\mathcal{A}}(k) = 2\Pr[\texttt{Success}] - 1$.

3.3 Security Requirements

From [31, 8], we can summarize that a conference key agreement protocol should meet the following requirements:

- **Authentication:** an outsider of set of participants cannot impersonate as a legal participant.
- **Validity:** in the presence of a benign adversary, all honest partner oracles accept the same session key.
- **Fairness:** the session key should be determined unbiasedly by all honest participants together.
- **Fault tolerance:** no coalition of malicious participants can spoil the conference by making honest participants compute different session key.
- **Indistinguishability:** for every probabilistic polynomial time adversary $\mathcal{A}$, the advantage $\texttt{Adv}^{\mathcal{A}}(k)$ to distinguishing test keys is negligible.
- **Forward secrecy:** exposure of the long-lived secret key does not enable an adversary to break the session key established at any prior time.

Then we derived a formal definition of secure conference key agreement protocol, and the version that with forward secrecy.

Definition 6 *We say that a protocol P is a secure conference key agreement protocol if the following properties are satisfied:* Authentication*,* Validity*,* Fairness*,* Fault tolerance*, and* Indistinguishability*.*

Definition 7 *A protocol P is a forward secure conference key agreement protocol if the following properties are satisfied:* Authentication*,* Validity*,* Fairness*,* Fault tolerance*,* Indistinguishability*, and* Forward secrecy*.*

Chapter 4

The New Protocols

We start to describe three conference key agreement protocols. First one is an adaptation from Tzeng and Tzeng's protocol [31], and then two new protocols with forward secrecy will be presented. Some notions and symbols used throughout our protocols are provided in Table 4.1.

Symbol	**Description**
P	The protocol
$\mathcal{A}$	The adversary
$\mathcal{U}$	The set of participants involved in protocol
Π_U^s	The s-th instance of participant U
SK_i (x_i)	Long-lived (secret) key of user U_i
PK_i (y_i)	Public key of user U_i
SID	Session ID
PID	Partner ID
ST	Session token
sk	Session key (Conference key)

Table 4.1: Symbols and Notions

4.1 Protocol Conf-1

PROTOCOL. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be the initial participant set, and each participant U_i, $1 \leq i \leq n$, knows $\mathcal{U}$. Without loss of generality, we assume that U_1 is the initiator who calls for a conference for the set $\mathcal{U}$ and sets the session token ST. Before executing this protocol, each participant are given a public key and private key pair by running algorithm `Gen`, and the key pair $(PK_i, SK_i) = (y_i, x_i)$ satisfies $y_i = g^{x_i} \bmod p$. Let h be a collision-resistant hash function, which is used in the modified ElGamal signature scheme, and it always computed involving with session token ST, which is unique for each conference session to prevent the replay attack.

In our protocol, each participant U_i first select a random value k_i and compute his partial secret $g^{k_i} \bmod p$, then transfers this secret to the other participants by sending $u_{i,j} = y_j^{k_i} \bmod p$, $1 \leq j \leq n$, thus ensure that only the participant $U_j \in \mathcal{U}$ can extract the partial secret $g^{k_i} \bmod p$ using his secret key x_i. U_i also sends NIPVS$(g, y_1, \ldots, y_n, u_{i,1}, \ldots, u_{i,n})$ to convincing the other participants that all the other participants receive the same partial secret, along with the signature (r_i, s_i) of his partial secret for authentication.

After receiving messages from the other participants, U_i starts to check whether each participant U_j, $j \neq i$, sends the valid messages and authenticates U_j's identity. If the check fails, U_i excludes U_j from the set of honest participants. Finally, U_i computes the conference key according to the set of honest participants. We now formalize our protocol in Figure 4.1.

– System parameters are g, p, q, and hash function $h(\cdot)$

– Each participant U_i holds his secret key x_i, and can access all other participants public keys y_j, for $1 \leq j \leq n$.

The participant U_i does the following two steps:

Step 1. **Message Sending**

(a) Randomly select k_i, $R_i \in Z_q$.

(b) Broadcast $u_{i,j} = y_j^{k_i} \bmod p$, for all $j \neq i$.

(c) Broadcast NIPVS$(g, y_1, \ldots, y_n, u_{i,1}, \ldots, u_{i,n})$.

(d) Broadcast signature of partial secret, $\texttt{Sig}(g^{k_i}) = (r_i, s_i)$ where $r_i = g^{R_i} \bmod p$ and
$s_i = R_i^{-1}(h(SID, r_i, g^{k_i}) - r_i x_i) \bmod q$.

Step 2. **Conference Key Computing**

(a) Compute $c_j = (u_{j,i})^{x_i^{-1}} \bmod p$, for all $j \neq i$.

(b) Check (r_j, s_j) is the correct signature[1] of c_j, for all $j \neq i$.

(c) Verify NIPVS$(g, y_1, \ldots, y_n, u_{j,1}, \ldots, u_{j,n})$.

(d) If participant U_j's message passes the check in previous two steps, then add U_j to honest participant set $\mathcal{U}_i$.

(e) Compute the conference key sk of session SID, where

$$sk = \prod_{j \in \mathcal{U}_i} c_j \bmod p = g^{k_{j,1} + \cdots + k_{j,m}}, \forall j \in \mathcal{U}_i$$

---

[1]Set $z_j = h(SID, r_i, g^{k_i})$, and check whether $g^{z_j} = y_j^{r_j} r_j^{s_j}$.

Figure 4.1: Protocol Conf-1

Security Analysis. For security, we prove that protocol Conf-1 meets all the security requirements defined in previous chapter, except for the forward secrecy. First of all, we show that this protocol is validity, fairness, and fault tolerance against malicious participants in Lemma 1. Then we follow Bellare and Rogaway's model closely to prove its authentication and indistinguishability in Lemma 2 and Lemma 3 respectively. Last, we conclude our proofs in Theorem 2.

Lemma 1 (Fault tolerance, Validity and Fairness [31]) *All honest participants who follow the protocol compute a common conference key with an overwhelming probability no matter how many participants are malicious. Furthermore, the common conference key is determined by the honest participants unbiasedly.*

Proof. For fault tolerance, we show that all honest participants will compute the same honest participant set in a high probability. Because all users only connected with broadcast network in our model, every participant receives the same messages. If a malicious participant U_i wants to send $(y_1, \ldots, y_n, u_{i,1}, \ldots, u_{i,n})$ such that not all $\log_{y_j} u_{i,j}$, $1 \leq j \leq n$, are equal, the probability that he can construct $\mathrm{NIPVS}(g, y_1, \ldots, y_n, u_{i,1}, \ldots, u_{i,n})$ is at most T/q, which is negligible, where T is U_i's runtime. Using this tool, all honest participants can exclude the malicious participants with high probability, and an honest participant who follow the protocol would be accepted by other honest participants as "honest" with high probability, too. Thus, any honest participant will not be excluded by any other honest participants, and any malicious participant who tries to cheat other participants to accept a different partial secret will be excluded by all honest participants. Eventually, all honest participants who follow our protocol will compute the same honest participant set with high probability.

For validity, we show that all honest participants compute the same conference key. Since we provide fault tolerance in our protocol, each honest participant U_i would compute the same participant set $\mathcal{U}_i$, for $1 \leq i \leq m$, then U_i uses his private key x_i to compute the partial key $c_j = (u_{j,i})^{x_i^{-1}} = g^{k_j} \bmod p$, for all $j \neq i$. Therefore, all users in honest participant set derive the same session key with an overwhelming probability.

For fairness, our session key is the multiplication of all partial key c_i, for $1 \leq i \leq m$, no one can biased this value since each honest participant choose k_i uniformly and independently over Z_q. Thus, no honest participants can bias the session key in our protocols. $\square$

Among many extensions of Bellare and Rogaway's model, we follow Bresson *et al.* [12] to divide the proof into two cases, the adversary $\mathcal{A}$ breaks our protocol either by forging a signature with respect to some participant's signing key, or without forging a signature. For authentication, we show that if $\mathcal{A}$ gains her advantage by forging a signature, we use $\mathcal{A}$ to construct a signature forging algorithm $\mathcal{F}$ against signature scheme $\mathcal{S}$, by guessing which participant that $\mathcal{A}$ will choose to producing a forgery during the protocol runs. For indistinguishability, if $\mathcal{A}$ could break the protocol without altering the content of the flows (i.e. forging a signature of some messages), then we can construct an algorithm $\mathcal{D}$ to solve an instance of the DDH problem.

Lemma 2 (Authentication) *Assume the random oracle model. If an outsider $\mathcal{A}$ can impersonate as a legal participant U_i by forging his signature with a non-negligible advantages ϵ within time t, being allowed to query the signing oracle q_s times. Then we can use $\mathcal{A}$ to construct a signature forging algorithm $\mathcal{F}$ against signature scheme $\mathcal{S}$, which succeeds with a non-negligible advantages ϵ/n within time $t' \leq t + q_s T(k)$.*

Proof. We use $\mathcal{A}$ to construct a forging algorithm $\mathcal{F}$ for the signature scheme $\mathcal{S}$. Given some participant's public key e in signature scheme $\mathcal{S}$ and accessed to a signing oracle for the corresponding secret key d. The successful $\mathcal{F}$ must output a valid signature[1] (m, σ) for some message m, which was not asked to the signing oracle previously. The forger $\mathcal{F}$ does as following:

1. **Setup**
   (a) Randomly choose a participant $U' \in \mathcal{U}$.
   (b) For participant $U_i = U'$, assign the given e as his public key y_i.
   (c) For other participants $U_i \neq U'$, runs key generating function `Gen` of protocol to assign user U_i's key pair (y_i, x_i), where $y_i = g^{x_i}$.
2. **$\mathcal{F}$ runs $\mathcal{A}$ as subroutine**

   $\mathcal{F}$ answers $\mathcal{A}$'s queries as follows, and maintains a list H for hash queries.

   – $\texttt{Send}(U_i, s, m)$: $\mathcal{F}$ outputs what he should output, follows the protocol. When he needs to generate the signature of partial secret $\texttt{Sig}(g^{k_i})$ for selected user $U_i = U'$, he queries the signing oracle. Otherwise, he can sign by himself because he owns all keys.

   – $\texttt{Reveal}(U_i, s)$: returns sk that $\Pi^s_{U_i}$ was involved.

[1]i.e. $\texttt{S.Ver}(e, m, \sigma) = 1$

– $\texttt{Corrupt}(U_i)$: If $U_i = U'$ then $\mathcal{F}$ fails, else returns participant U_i's long-lived secret key x_i.

– $h(m)$: If $(m, h(m))$ is not in $\mathcal{F}$'s list H, returns a random string r and adds (m, r) to the list, else returns message m's corresponding hash value.

3. **Output**

During the execution of $\mathcal{A}$, if $\mathcal{A}$ makes a query $\texttt{Send}(\cdot, (m, \sigma))$, where σ is a valid signature on m, respect to y_i for $U_i = U'$, and m was not queried to signing oracle previously, then $\mathcal{F}$ outputs (m, σ) as his forgery. Otherwise, when $\mathcal{A}$ terminates, the forger $\mathcal{F}$ fails.

At the beginning, we assume that $\mathcal{A}$ can forge a signature with a non-negligible advantages ϵ within time t. And the probability of this forgery respected to our chosen participant U' is at least $1/n$. Thus the forger $\mathcal{F}$ will succeed with probability $\texttt{Succ}_{\mathcal{S}}^{cma}(\mathcal{F}) \geq \epsilon/n$, and the running time is $t' \leq t + q_s T(k)$, where $T(k)$ is the running of querying a signing oracle. □

Lemma 3 (Indistinguishability) *Assume the random oracle model. If an adversary $\mathcal{A}$ could break the protocol without altering the content of the flows, with advantages at least ϵ within time t. Then we can construct an algorithm $\mathcal{D}$ to solve an instance of the DDH problem with advantages ϵ/n within time $t' \leq t + q_h T(k)$.*

Proof. Given an instance of the DDH problem (g, p, u_1, u_2, u_3), we show that the algorithm $\mathcal{D}$ can distinguish the input $(g, p, u_1, u_2, u_3) \in D_n$ from $(g, p, u_1, u_2, u_3) \in R_n$ with non-negligible advantages, while runs $\mathcal{A}$ as subroutine. It does as following:

1. **Setup**

   (a) Randomly choose two participants $U', U'' \in \mathcal{U}$.

   (b) For participant $U_i = U''$, assign the given u_1 as his public key y_i.

   (c) For other participants $U_i \neq U''$, assigns user U_i's key pair as follows: random $r_i \in Z_q$, and set $(y_i, x_i) = (g^{r_i}, r_i)$, thus his key pair is in correct form $y_i = g^{x_i}$.

2. **$\mathcal{D}$ runs $\mathcal{A}$ as subroutine**

   $\mathcal{D}$ answers $\mathcal{A}$'s queries as follows, and maintains a list H for hash queries.

   – `Send`(U_i, s, m) : $\mathcal{D}$ does following steps

     i. If $U_i = U'$, then set $c_i = u_2$, else random $k_i \in Z_q$ and set $c_i = g^{k_i}$.

     ii. If $U_i = U'$, then set $u_{i,j} = u_3$ for $U_j = U''$ and $u_{i,j} = u_2^{r_j}$ for $U_j \neq U', U''$, else set $u_{i,j} = y_j^{k_i}$, for all $j \neq i$.

     iii. Forges the NIPVS by hash oracle. Randomly selects (c, w) and sets $H(g\|y_1\|\cdots\|y_n\|u_{i,1}\|\cdots\|u_{i,n}\|y_1^w u_{i,1}^c\|\cdots\|y_n^w u_{i,n}^c) = c$ in list H.

     iv. If $U_i = U''$, $\mathcal{D}$ needs to forge the signature of c_i, he randomly selects $a \in Z_q$ and $b \in Z_q^*$, then returns $(r_i, s_i) = (g^a y_i^b, -r_i b^{-1})$, and set $H(SID, r_i, u_2) = -r_i a b^{-1}$ in list H. Otherwise he can sign by himself because he generates all other participants' key.

   – `Reveal`(U_i, s) : Returns sk that $\Pi_{U_i}^s$ was involved. Though $\mathcal{D}$ does not know U_i's secret key when $U_i = U''$, he can compute the session key sk from other participants under his control.

   – `Corrupt`(U_i) : If $U_i = U''$ then $\mathcal{D}$ fails, else returns participant U_i's long-lived secret key x_i.

- `Test`(U_i, s) : Flips a coin $b \in \{0,1\}$. If $b = 1$ then returns the $sk = u_2 \cdot \prod g^{k_j}$, for $j \in \mathcal{U}_i$, else returns a random string drawn from the same distribution as session key.
- $h(m)$: If $(m, h(m))$ is not in $\mathcal{F}$'s list H, returns a random string r and adds (m, r) to the list, else returns message m's corresponding hash value.

3. **Output**

   Eventually, adversary $\mathcal{A}$ will output a guess b', and wins the game if $b = b'$. If adversary $\mathcal{A}$ wins, then the distinguisher $\mathcal{D}$ outputs b, otherwise output a random bit $b'' \in \{0,1\}$.

At the beginning, we assume that break the protocol without altering the content of the flows, with advantages at least ϵ within time t. And the probability that $\mathcal{D}$ gives the random sk is $1/2$. The probability that adversary $\mathcal{A}$ uses messages sent to U'' distinguishing real sk is at least $1/n$. Thus the distinguisher $\mathcal{D}$ will succeed with probability $\texttt{Succ}^{DDH}(\mathcal{D}) \geq 1/2 + \epsilon/n$, and the running time is $t' \leq t + q_h T(k)$, where $T(k)$ is the running of querying a hash oracle. □

Theorem 2 *Assume the random oracle model and broadcast channel. The protocol* Conf-1 *meets all security requirements:* Authentication*,* Validity*,* Fairness*,* Fault tolerance*, and* Indistinguishability*.*

Proof. Firstly we show the protocol is validity, fairness, and fault tolerance in Lemma 1. Then its authentication proved in Lemma 2, and finally the indistinguishability is proved in Lemma 3. Thus we can conclude that the protocol meets all security requirements as mentioned. □

4.2 Protocol Conf-2

Though the protocol Conf-1 is round-efficiency, it does not provide forward secrecy. In protocol Conf-2, we add an extra round to exchange a temporary random public key. In this way, our protocol can provide forward secrecy.

Protocol. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be the initial participant set, and each participant U_i, $1 \leq i \leq n$, knows $\mathcal{U}$. Without loss of generality, we assume that U_1 is the initiator who calls for a conference for the set $\mathcal{U}$ and sets the session token ST. Before executing this protocol, each participant are given a public key and private key pair by running algorithm `Gen`, and the key pair $(PK_i, SK_i) = (y_i, x_i)$ satisfies $y_i = g^{x_i} \bmod p$. Let h be a collision-resistant hash function, which is used in the modified ElGamal signature scheme, and it always computed involving with session token ST, which is unique for each conference session to prevent the replay attack.

In this protocol, each participant U_i first selects a random value v_i and compute his temporal public key $Y_i = y_i^{v_i} \bmod p$, then transfers this key to the other participants along with its signature. After all participants can authenticate the new public key, then the set of participants in conference using this temporal public key to run Conf-1. We formalize this protocol in Figure 4.2.

Security Analysis. For security, we prove that protocol Conf-2 meets all the security requirements defined in previous chapter. First of all, we show that this protocol is validity, fairness, and fault tolerance against malicious

– System parameters are the same as Conf-1.

The participant U_i does the following four steps:

Step 1. **Temporal Public Key Exchange**

(a) Randomly select $R'_i \in Z_q$ and $v_i \in Z_q^*$.

(b) Broadcast $Y_i = y_i^{v_i} \bmod p$.

(c) Broadcast signature of temporal key, $\mathtt{Sig}(Y_i) = (r'_i, s'_i)$ where $r'_i = g^{R'_i} \bmod p$ and
$s'_i = R'^{-1}_i(\, h(SID, r'_i, Y_i) - r'_i\, x_i) \bmod q$.

Step 2. **Temporal Public Key Verification**

(a) Check (r'_j, s'_j) is the correct signature of Y_j, for all $j \neq i$.

Step 3. **Message Sending**

(a) Randomly select k_i, $R_i \in Z_q$.

(b) Broadcast $u_{i,j} = Y_j^{k_i} \bmod p$, for all $j \neq i$.

(c) Broadcast NIPVS$(g, Y_1, \ldots, Y_n, u_{i,1}, \ldots, u_{i,n})$.

(d) Broadcast signature of partial secret, $\mathtt{Sig}(g^{k_i}) = (r_i, s_i)$ where $r_i = g^{R_i} \bmod p$ and
$s_i = R_i^{-1}(\, h(SID, r_i, g^{k_i}) - r_i\, x_i\,) \bmod q$.

Step 4. **Conference Key Computing**

(a) Compute $c_j = (u_{j,i})^{(x_i\, v_i)^{-1}} \bmod p$, for all $j \neq i$.

(b) Check (r_j, s_j) is the correct signature of c_j, for all $j \neq i$.

(c) Verify NIPVS$(g, Y_1, \ldots, Y_n, u_{j,1}, \ldots, u_{j,n})$.

(d) If participant U_j's message passes the check in previous two steps, then add U_j to honest participant set $\mathcal{U}_i$.

(e) Compute the conference key sk of session SID, where

$$sk = \prod_{j \in \mathcal{U}_i} c_j \bmod p = g^{k_{j,1} + \cdots + k_{j,m}}, \forall j \in \mathcal{U}_i$$

Figure 4.2: Protocol Conf-2

participants in Lemma 4. Then we follow Bellare and Rogaway's model to prove its authentication and indistinguishability in Lemma 5 and Lemma 6 respectively. Finally, conclude our proofs in Theorem 3. The forward secrecy is achieved by using the *fs-fresh* oracle definition, and we will explain them later.

Lemma 4 (Fault tolerance, Validity and Fairness) *All honest participants who follow the protocol compute a common conference key with an overwhelming probability no matter how many participants are malicious. Furthermore, the common conference key is determined by the honest participants unbiasedly.*

Proof. In this protocol, all user can authenticate the temporal public keys, since we use a the modified ElGamal signature scheme to sign the temporal key. When the temporal keys are authenticated, the rest steps are identical to the protocol CONF-1, thus its fault tolerance, validity and fairness can be proved similarly. □

Here, we also follow Bresson *et al.* [12] to divide the proof into two cases, the adversary $\mathcal{A}$ breaks our protocol either by forging a signature with respect to some participant's signing key, or without forging a signature. For authentication, we show that if $\mathcal{A}$ gains her advantage by forging a signature, we use $\mathcal{A}$ to construct a signature forging algorithm $\mathcal{F}$ against signature scheme $\mathcal{S}$, by guessing which participant that $\mathcal{A}$ will choose to producing a forgery during the protocol runs. For indistinguishability, if $\mathcal{A}$ could break the protocol without altering the content of the flows (i.e. forging a signature of some messages), then we can construct an algorithm $\mathcal{D}$ to solve an instance of the DDH problem.

For forward secrecy, we use the different definition on fresh oracle, which called *fs-fresh* as we mentioned in section 3.2. In this definition, the adversary $\mathcal{A}$ can make a `Corrupt` query on U after asking a `Test` query on U, and since the query $\texttt{Corrupt}(U)$ only returns participant U's long-lived secret key, it will not disclose the information of session key established previously in forward secure setting.

Lemma 5 (Authentication) *Assume the random oracle model. If an outsider $\mathcal{A}$ can impersonate as a legal participant U_i by forging his signature with a non-negligible advantages ϵ within time t, being allowed to query the signing oracle q_s times. Then we can use $\mathcal{A}$ to construct a signature forging algorithm $\mathcal{F}$ against signature scheme $\mathcal{S}$, which succeeds with a non-negligible advantages ϵ/n within time $t' \leq t + q_s T(k)$.*

Proof. We use $\mathcal{A}$ to construct a forging algorithm $\mathcal{F}$ for the signature scheme $\mathcal{S}$. Given some participant's public key e in signature scheme $\mathcal{S}$ and accessed to a signing oracle for the corresponding secret key d. The successful $\mathcal{F}$ must output a valid signature[2] (m, σ) for some message m, which was not asked to the signing oracle previously. The forger $\mathcal{F}$ does as following:

1. **Setup**
   (a) Randomly choose a participant $U' \in \mathcal{U}$.
   (b) For participant $U_i = U'$, assign the given e as his public key y_i.

[2]i.e. $\texttt{S.Ver}(e, m, \sigma) = 1$

(c) For other participants $U_i \neq U'$, runs key generating function Gen of protocol to assign user U_i's key pair (y_i, x_i), where $y_i = g^{x_i}$.

2. **$\mathcal{F}$ runs $\mathcal{A}$ as subroutine**

   $\mathcal{F}$ answers $\mathcal{A}$'s queries as follows, and maintains a list H for hash queries.

   – Send(U_i, s, m) : $\mathcal{F}$ outputs what he should output, follows the protocol. When he needs to generate the signature of partial secret Sig(g^{k_i}) for selected user $U_i = U'$, he queries the signing oracle. Otherwise, he can sign by himself because he owns all keys.

   – Reveal(U_i, s) : returns sk that $\Pi^s_{U_i}$ was involved.

   – Corrupt(U_i) : If $U_i = U'$ then $\mathcal{F}$ fails, else returns participant U_i's long-lived secret key x_i.

   – $h(m)$: If $(m, h(m))$ is not in $\mathcal{F}$'s list H, returns a random string r and adds (m, r) to the list, else returns message m's corresponding hash value.

3. **Output**

   During the execution of $\mathcal{A}$, if $\mathcal{A}$ makes a query Send$(\cdot, (m, \sigma))$, where σ is a valid signature on m, respect to y_i for $U_i = U'$, and m was not queried to signing oracle previously, then $\mathcal{F}$ outputs (m, σ) as his forgery. Otherwise, when $\mathcal{A}$ terminates, the forger $\mathcal{F}$ fails.

At the beginning, we assume that $\mathcal{A}$ can forge a signature with a non-negligible advantages ϵ within time t. And the probability of this forgery respected to our chosen participant U' is at least $1/n$. Thus the forger $\mathcal{F}$ will succeed with probability $\mathtt{Succ}^{cma}_{\mathcal{S}}(\mathcal{F}) \geq \epsilon/n$, and the running time is $t' \leq t + q_s T(k)$, where $T(k)$ is the running of querying a signing oracle. $\square$

Lemma 6 (Indistinguishability) *Assume the random oracle model. If an adversary $\mathcal{A}$ could break the protocol without altering the content of the flows, with advantages at least ϵ within time t. Then we can construct an algorithm $\mathcal{D}$ to solve an instance of the DDH problem with advantages ϵ/n within time $t' \leq t + q_h T(k)$.*

Proof. Given an instance of the DDH problem (g, p, u_1, u_2, u_3), we show that the algorithm $\mathcal{D}$ can distinguish the input $(g, p, u_1, u_2, u_3) \in D_n$ from $(g, p, u_1, u_2, u_3) \in R_n$ with non-negligible advantages, while runs $\mathcal{A}$ as subroutine. It does as following:

1. **Setup**
   (a) Randomly choose two participants $U', U'' \in \mathcal{U}$.
   (b) For participant $U_i = U''$, assign the given u_1 as his public key y_i.
   (c) For other participants $U_i \neq U''$, assigns user U_i's key pair as follows: random $r_i \in Z_q$, and set $(y_i, x_i) = (g^{r_i}, r_i)$, thus his key pair is in correct form $y_i = g^{x_i}$.
2. **$\mathcal{D}$ runs $\mathcal{A}$ as subroutine**

   $\mathcal{D}$ answers $\mathcal{A}$'s queries as follows, and maintains a list H for hash queries.
   – `Send`(U_i, s, m) : If the query is executed step 1 or step 2 , $\mathcal{D}$ would follow protocol. Otherwise, does following steps
      i. If $U_i = U'$, then set $c_i = u_2$, else random $k_i \in Z_q$ and set $c_i = g^{k_i}$.
      ii. If $U_i = U'$, then set $u_{i,j} = u_3$ for $U_j = U''$ and $u_{i,j} = u_2^{r_j v_j}$ for $U_j \neq U', U''$, else set $u_{i,j} = Y_j^{k_i}$, for all $j \neq i$.
      iii. Forges the NIPVS by hash oracle. Randomly selects (c, w) and sets $H(g \| Y_1 \| \cdots \| Y_n \| u_{i,1} \| \cdots \| u_{i,n} \| Y_1^w u_{i,1}^c \| \cdots \| Y_n^w u_{i,n}^c) = c$ in list H.

iv. If $U_i = U''$, $\mathcal{D}$ needs to forge the signature of c_i, he randomly selects $a \in Z_q$ and $b \in Z_q^*$, then returns $(r_i, s_i) = (g^a y_i^b, -r_i b^{-1})$, and set $H(SID, r_i, u_2) = -r_i a b^{-1}$ in list H. Otherwise he can sign by himself because he generates all other participants' key.

– `Reveal`(U_i, s) : Returns sk that $\Pi_{U_i}^s$ was involved. Though $\mathcal{D}$ does not know U_i's secret key when $U_i = U''$, he can compute the session key sk from other participants under his control.

– `Corrupt`(U_i) : If $U_i = U''$ then $\mathcal{D}$ fails, else returns participant U_i's long-lived secret key x_i.

– `Test`(U_i, s) : Flips a coin $b \in \{0, 1\}$. If $b = 1$ then returns the $sk = u_2 \cdot \prod g^{k_j}$, for $j \in \mathcal{U}_i$, else returns a random string drawn from the same distribution as session key.

– $h(m)$: If $(m, h(m))$ is not in $\mathcal{F}$'s list H, returns a random string r and adds (m, r) to the list, else returns message m's corresponding hash value.

3. **Output**

Eventually, adversary $\mathcal{A}$ will output a guess b', and wins the game if $b = b'$. If adversary $\mathcal{A}$ wins, then the distinguisher $\mathcal{D}$ outputs b, otherwise output a random bit $b'' \in \{0, 1\}$.

Initially, we assume that break the protocol without altering the content of the flows, with advantages at least ϵ within time t. Then the probability that $\mathcal{D}$ gives the random sk is $1/2$. The probability that adversary $\mathcal{A}$ uses messages sent to U'' distinguishing real sk is at least $1/n$. Thus the distinguisher $\mathcal{D}$ will succeed with probability $\texttt{Succ}^{DDH}(\mathcal{D}) \geq 1/2 + \epsilon/n$, and the running time is $t' \leq t + q_h T(k)$, where $T(k)$ is the running of querying a hash oracle. □

Theorem 3 *Assume the random oracle model and broadcast channel. The protocol* Conf-1 *meets all security requirements:* Authentication, Validity, Fairness, Fault tolerance, Indistinguishability, *and* forward secrecy.

Proof. Firstly we show the protocol is validity, fairness, and fault tolerance in Lemma 4. Then its authentication proved in Lemma 5, and finally the indistinguishability is proved in Lemma 6. And the forward secrecy is achieved by using the *fs-fresh* definition on oracle, which allows `Corrupt` queries after asking a `Test` query on some user U. Thus we can conclude that the protocol meets all security requirements as mentioned. □

4.3 Protocol Conf-3

In previous section, we use an extra round to exchange the temporal public key to provide forward secrecy. In this section we will combine the forward-secure PKI with previous protocol Conf-1 to achieve forward secrecy. We describe our motivation as follows. Since we use the public-key cryptosystem in our protocol, in the working flows, all users must connect to a trusted Certification Authority (CA) to confirm the validity of other participants' public keys before performing key agreement. Moreover, since many forward secure cryptosystem have been proposed (e.g. [14, 22]), we can assume that our system provides such functionality. Therefore, we can use CA as a *token service server* (*TSS*), when we queries the Certificate Revocation List (CRL) from CA, it issues the common session token ST simultaneously. Thus all conference participants can update their keys to the same time period.

PROTOCOL. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be the initial participant set, and each participant U_i, $1 \leq i \leq n$, knows $\mathcal{U}$. Without loss of generality, we assume that U_1 is the initiator who calls for a conference for the set $\mathcal{U}$ and register $\mathcal{U}$ along with *SID* to the *TSS*. Before executing this protocol, each participant are given a public key and private key pair which can be update periodically, which are provided by forward secure PKI.

At the beginning, each participant U_i updates his secret key to the time period *ST*, which he gets from the *TSS*. Then he runs similar step as protocol CONF-1, except for encrypting and signing the partial secret using the corresponding key of time period *ST*. After he computes the session sk, he must update the secret key again (i.e. update his key to time period $ST + 1$). This step is important to our forward secrecy property, because once the participant's key is obtained by attacker, the updated key prevents the attacker to trace back the encryption key we used in conference key agreement. We describe the detail of this protocol in Figure 4.3.

SECURITY ANALYSIS. For security, we prove that protocol CONF-3 meets all the security requirements defined in previous chapter. Again, we show that this protocol is validity, fairness, and fault tolerance against malicious participants in Lemma 7. Then prove its authentication and indistinguishability in Lemma 8 and Lemma 9 respectively. Finally, conclude our proofs in Theorem 4.

- System parameters are g, p, q, and hash function $h(\cdot)$
- Each participant U_i holds his secret key $x_{i(t)}$, which can be updated to corresponding time period t, and can access all other participants public keys y_j, for $1 \leq j \leq n$.
- Before each conference starts, all participants must connect to the *Token Service Server* (*TSS*), to get the session token *ST*.

The participant U_i does the following two steps:

Step 1. **Message Sending**

(a) Update key $x_{i(t)}$ to the time period *ST*, i.e. $x_{i(ST)}$.

(b) Randomly select $k_i \in Z_q$.

(c) Broadcast $u_{i,j} = \texttt{FE.Enc}(y_j, ST, g^{k_i})$, for all $j \neq i$.

(d) Broadcast NIP(Consistence of g^{k_i}).

(e) Broadcast $s_i = \texttt{FS.Sig}(x_{i(ST)}, ST, (u_{i,1}, \ldots, u_{i,n}, ST))$.

Step 2. **Conference Key Computing**

(a) Compute $c_j = \texttt{FS.Dec}(y_i, ST, x_{i(ST)}, u_{j,i})$, for all $j \neq i$.

(b) Check $\texttt{FS.Ver}(y_j, (u_{j,1}, \ldots, u_{j,n}, ST), s_j) = 1$, for all $j \neq i$.

(c) Verify NIP(Consistence of g^{k_j}).

(d) If participant U_j's message passes the check in previous two steps, then add U_j to honest participant set $\mathcal{U}_i$.

(e) Compute the conference key *sk* of session *SID*, where

$$sk = h(\prod_{j \in \mathcal{U}_i} c_j \bmod p, SID, ST), \forall j \in \mathcal{U}_i$$

(f) Update key $x_{i(ST)}$ to the next time period, i.e. $x_{i(ST+1)}$.

Figure 4.3: Protocol Conf-3

Lemma 7 (Fault tolerance, Validity and Fairness) *All honest participants who follow the protocol compute a common conference key with an overwhelming probability no matter how many participants are malicious. Furthermore, the common conference key is determined by the honest participants unbiasedly.*

Proof. In this protocol, all arguments about its validity and fairness are similar to lemma 1. For fault tolerance, we use an generic form of non-interactive proof (NIP) system to prove the consistence of the partial secret that participant sent. We can design a NIP based on what kind of forward secure encryption scheme we use, because based on standard intractability assumptions, it is known how to construct a non-interactive zero-knowledge proof for any NP-set [25]. Once we has the NIP, the fault tolerance is guaranteed as we explain in lemma 1. □

As previous proof for protocol Conf-2, we assume that the adversary $\mathcal{A}$ breaks our protocol either by forging a signature with respect to some participant's signing key, or without forging a signature. For authentication, we show that if $\mathcal{A}$ gains her advantage by forging a signature, we use $\mathcal{A}$ to construct a signature forging algorithm $\mathcal{F}$ against a forward secure signature scheme $\mathcal{FS}$, by guessing which participant that $\mathcal{A}$ will choose to producing a forgery during the protocol runs. For indistinguishability, if $\mathcal{A}$ could break the protocol without altering the content of the flows (i.e. forging a signature of some messages), then we can construct an algorithm $\mathcal{X}$ against the underlying forward secure encryption scheme $\mathcal{FE}$ in the multi-user setting.

Lemma 8 (Authentication) *Assume the random oracle model. If an outsider $\mathcal{A}$ can impersonate as a legal participant U_i by forging his signature with a non-negligible advantages ϵ within time t, being allowed to query the signing oracle q_s times. Then we can use $\mathcal{A}$ to construct a signature forging algorithm $\mathcal{F}$ against forward secure signature scheme $\mathcal{FS}$, which succeeds with a non-negligible advantages ϵ/n within time $t' \leq t + q_s T(k)$.*

Proof. We use $\mathcal{A}$ to construct a forging algorithm $\mathcal{F}$ for the forward secure signature scheme $\mathcal{FS}$. Given some participant's public key e in signature scheme $\mathcal{S}$ and accessed to a signing oracle for the corresponding secret key d. The successful $\mathcal{F}$ must output a valid signature[3] (m, σ) for some message m, which was not asked to the signing oracle previously. The forger $\mathcal{F}$ does as following:

1. **Setup**
   (a) Randomly choose a participant $U' \in \mathcal{U}$.
   (b) For participant $U_i = U'$, assign the given e as his public key y_i.
   (c) For other participants $U_i \neq U'$, runs key generating function `Gen` of protocol to assign user U_i's key pair $(y_i, x_{i(t)})$, where $t = 0$ at initially.
2. **$\mathcal{F}$ runs $\mathcal{A}$ as subroutine**

   $\mathcal{F}$ answers $\mathcal{A}$'s queries as follows, and maintains a list H for hash queries.

   – $\texttt{Send}(U_i, s, m)$: $\mathcal{F}$ outputs what he should output, follows the protocol. When he needs to generate the signature of partial secret $\texttt{FS.Sig}(\cdot)$ for selected user $U_i = U'$, he queries the signing oracle. Otherwise, he can sign by himself because he owns all keys.

[3]i.e. $\texttt{FS.Ver}(e, m, \sigma) = 1$

- $\texttt{Reveal}(U_i, s)$: returns sk that $\Pi^s_{U_i}$ was involved.
- $\texttt{Corrupt}(U_i)$: If $U_i = U'$ then $\mathcal{F}$ fails, else returns participant U_i's long-lived secret key $x_{i(t)}$, for some time period t.
- $h(m)$: If $(m, h(m))$ is not in $\mathcal{F}$'s list H, returns a random string r and adds (m, r) to the list, else returns message m's corresponding hash value.

3. **Output**

During the execution of $\mathcal{A}$, if $\mathcal{A}$ makes a query $\texttt{Send}(\cdot, (m, \sigma))$, where σ is a valid signature on m, respect to y_i for $U_i = U'$, and m was not queried to signing oracle previously, then $\mathcal{F}$ outputs (m, σ) as his forgery. Otherwise, when $\mathcal{A}$ terminates, the forger $\mathcal{F}$ fails.

At the beginning, we assume that $\mathcal{A}$ can forge a signature with a non-negligible advantages ϵ within time t. And the probability of this forgery respected to our chosen participant U' is at least $1/n$. Thus the forger $\mathcal{F}$ will succeed with probability $\texttt{Succ}^{fs-cma}_{\mathcal{S}}(\mathcal{F}) \geq \epsilon/n$, and the running time is $t' \leq t + q_s T(k)$, where $T(k)$ is the running of querying a signing oracle. □

Lemma 9 (Indistinguishability) *Assume the random oracle model. If an adversary $\mathcal{A}$ could break the protocol without altering the content of the flows, with advantages at least ϵ within time t. Then we can construct an algorithm $\mathcal{X}$ against the underlying forward secure encryption scheme $\mathcal{FE}$ in the multi-user setting with ϵ/nq_h within time $t' \leq t + q_h T(k)$.*

Proof. Given an instance of *fs-CCA* Game in multi-user setting as follows:

- public keys $e_1, \ldots, e_{n-1}$ generated by algorithm $\texttt{Gen}$.
- two random string m_1 and m_2.

– ciphertexts $C_1 = \texttt{FE.Enc}(e_1, T, m_b), \ldots, C_{n-1} = \texttt{FE.Enc}(e_{n-1}, T, m_b)$ for some fixed time period T, and $b \in_R \{0, 1\}$.

The goal of $\mathcal{X}$ is to guess whether $b = 0$ or $b = 1$ with non-negligible advantages, while runs $\mathcal{A}$ as subroutine. It does as following:

1. **Setup**

   (a) Randomly choose a participant $U_1, \ldots, U_n \in \mathcal{U}$.

   (b) For participant $U_1, \ldots, U_{n-1}$, assign the given e_i as his public key y_i.

   (c) For other participants in $\mathcal{U}$, assigns their encryption key pair by $\texttt{FE.Gen}$, but for simplicity, we exclude them in protocol runs except U_n.

   (d) For all participants in $\mathcal{U}$, assigns user's signing key pair using $\texttt{FS.Gen}$

2. **$\mathcal{X}$ runs $\mathcal{A}$ as subroutine**

   $\mathcal{X}$ answers $\mathcal{A}$'s queries as follows, and maintains a list H for hash queries.

   – $\texttt{Send}(U_i, s, m)$: $\mathcal{D}$ does following steps

   i. If $ST = T$ and $U_i = U_n$, then set $u_{i,j} = C_j$ for $1 \le j \le n-1$, and set else $u_{i,j} = \texttt{FE.Enc}(e_j, T, m_0)$. Otherwise, just follows the protocol.

   ii. Forges the NIP$(\cdot)$ by hash oracle if necessary.

   – $\texttt{Reveal}(U_i, s)$: Since we model the session key as the output of random oracle, we only need to return a random string or the corresponding value if the key revealed previously.

   – $\texttt{Corrupt}(U_i)$: If $U_i \neq U_n$ then $\mathcal{X}$ fails, else returns participant U_n's long-lived secret key $x_{i(t)}$ for some time period.

   – $\texttt{Test}(U_i, s)$: returns a random string sk, which is selected from the distribution of session key.

- $h(m)$: If $(m, h(m))$ is not in $\mathcal{F}$'s list H, returns a random string r and adds (m, r) to the list, else returns message m's corresponding hash value.

3. **Output**

   Eventually, adversary $\mathcal{A}$ terminates and output a guess b'. $\mathcal{X}$ determinates what should guess using the random oracles's hash list H. He checks all queries made by $\mathcal{A}$ of the form $h = (z, SID, ST)$. If there exist an entry such that $z = m_0 \cdot \prod c_j \bmod p$, for all $1 \leq j \leq n$, then $\mathcal{X}$ returns his guess $b'' = 0$, else returns $b'' = 1$.

At the beginning, we assume that $\mathcal{A}$ can break the protocol without altering the content of the flows, with advantages at least ϵ within time t. Thus the algorithm $\mathcal{X}$ will succeed with probability $\mathtt{Succ}^{fs-CCA}_{m-\mathcal{FE}}(\mathcal{X}) \geq \epsilon / nq_h$, where q_h is the number of hash queries made by adversary, and the running time is $t' \leq t + q_h T(k)$, where $T(k)$ is the running of querying a hash oracle. $\square$

Theorem 4 *Assume the random oracle model and broadcast channel. The protocol* Conf-1 *meets all security requirements:* Authentication*,* Validity*,* Fairness*,* Fault tolerance*,* Indistinguishability*, and* forward secrecy*.*

Proof. We have argued that the protocol is validity, fairness, and fault tolerance in Lemma 7. Then its authentication is described in Lemma 8, and the indistinguishability is proved in Lemma 9. Finally, we note that the forward secrecy is implicit in breaking scheme $\mathcal{FE}$ and $\mathcal{FS}$ in the last two lemmas. Thus we can say that the protocol meets all above security requirements. $\square$

Chapter 5

Conclusion

We have presented two new round-efficient conference key agreement protocols and provided the proofs of their security. Both of these protocols meet all our security requirements. In the following section, some proposed conference key agreement protocols are compared with ours in terms of efficiency and security. Finally, we give our conclusions and discuss some future works.

5.1 Comparison

Since our protocols are focus on forward secrecy and round efficiency, we compare them with some existed conference key agreement protocols in this section. For security, we consider the properties: authentication, fairness, forward secrecy, and fault tolerance. It's summarized in Table 5.1.

Protocol	auth.	fairness	forward secrecy	fault tolerance
Conf-1	Yes	Yes	No	Yes
Conf-2	Yes	Yes	Yes	Yes
Conf-3	Yes	Yes	Yes	Yes
BD94 [13]	No	Yes	Yes	No
TT00 [31]	Yes	Yes	No	Yes
BM03 [8]	Yes[1]	Yes	No	No

Table 5.1: Security Comparison of Conference Key Agreement Protocols

For efficiency, we analyze the performance of the protocols in terms of communication rounds and messages complexity. We compare them in Table 5.2.

Protocol	rounds	broadcasts	messages size[2]	total messages
Conf-1	1	1	$O(n)$	$O(n^2)$
Conf-2	2	2	$O(n)$	$O(n^2)$
Conf-3	1	1	$O(n)$	$O(n^2)$
BD94 [13]	2	2	$O(1)$	$O(n)$
TT00 [31]	1	1	$O(n)$	$O(n^2)$
BM03 [8]	1	1	$O(1)$	$O(n)$

Table 5.2: Efficiency Comparison of Conference Key Agreement Protocols

[1]It will suffer from replay attack.
[2]It counts number of messages that one participant sent.

5.2 Conclusion

We have proposed two new conference key agreement protocols, which used different approach to achieve the forward secrecy. Both of these protocols have been proved secure under Bellare and Rogaway's model and meet the security requirements: authentication, validity, fairness, fault tolerance, indistinguishability and forward secrecy. Though protocol CONF-2 adds an extra round to provide forward secrecy, it can still be done in constant rounds, and as efficient as the well known protocol proposed by Burmester and Desmedt [13]. Our original aim is finally realized in protocol CONF-3, which combines the forward secure encryption scheme, forward secure signature scheme, and uses a trusted *Token Service Server*. It needs only a single round to complete the key agreement, and also provides forward secrecy.

Since we do not assume that all participants are honest, we use the Publicly Verifiable Secret protocol to ensure that all participants send out consistent partial secrets to other participants. This seems to be the major disadvantage of our protocols, because of the message complexity become $O(n^2)$ for n participants.

Therefore, it would be interesting to find a single round protocol that meets all our security requirements and costs only $O(n)$ messages complexity. Besides, it remains an open problem that whether it is possible to construct a multi-party contributory conference key agreement protocol which completes in single round and also provides forward secrecy, while without the help of the underlying forward secure schemes.

Bibliography

[1] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS '98)*, pages 17–26. ACM Press, 1998.

[2] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, 2000.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proceedings of Advances in Cryptology - EUROCRYPT '00*, volume 1807 of *LNCS*, pages 139–155. Springer-Verlag, 2000.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of Advances in Cryptology - CRYPTO '93*, volume 773 of *LNCS*, pages 232–249. Springer-Verlag, 1994.

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

[6] S. Blake-Wilson and A. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Proceedings of 5th Security Protocols Workshop*, volume 1361 of *LNCS*, pages 137–158. Springer-Verlag, 1998.

[7] S. Blake-Wilson and A. Menezes. Authenticated diffie-hellman key agreement protocols. In *Proceedings of Selected Areas in Cryptography (SAC '98)*, volume 1556 of *LNCS*, pages 339–361. Springer-Verlag, 1999.

[8] C. Boyd and J. M. G. Nieto. Round-optimal contributory conference key agreement. In *Proceedings of the Public-Key Cryptography (PKC '03)*, volume 2567 of *LNCS*, pages 161–174. Springer-Verlag, 2002.

[9] V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Proceedings of Advances in Cryptology - EUROCRYPT '00*, volume 1807 of *LNCS*, pages 156–171. Springer-Verlag, 2000.

[10] V. Boyko, P. D. MacKenzie, and S. Patel. Provably authenticated group diffie-hellman key exchange - the dynamic case. In *Proceedings of Advances in Cryptology - Asiacrypt '01*, volume 2248 of *LNCS*, pages 290–309. Springer-Verlag, 2001.

[11] V. Boyko, P. D. MacKenzie, and S. Patel. Dynamic group diffie-hellman key exchange under standard assumptions. In *Proceedings of Advances in Cryptology - EUROCRYPT '02*, volume 2332 of *LNCS*, pages 321–336. Springer-Verlag, 2002.

[12] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group diffie-hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security (CCS '01)*, pages 255–264. ACM Press, 2001.

[13] M. V. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Proceedings of Advances in Cryptology - EUROCRYPT '94*, volume 950 of *LNCS*, pages 275–286. Springer-Verlag, 1995.

[14] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *Proceedings of Advances in Cryptology - EUROCRYPT '03*, volume 2656 of *LNCS*, pages 255–271. Springer-Verlag, 2003.

[15] K. Y. Choi, J. Y. Hwang, and D. H. Lee. Efficient id-based group key agreement with bilinear maps. In *Proceedings of the Public-Key Cryptography (PKC '04)*, volume 2947 of *LNCS*, pages 130–144. Springer-Verlag, 2004.

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

[17] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

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

[19] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[20] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[21] I. Ingemarsson, D. T. Tang, and C. K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, IT-28(5):714–720, 1982.

[22] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *Proceedings of Advances in Cryptology - CRYPTO '01*, volume 2139 of *LNCS*, pages 332–354. Springer-Verlag, 2001.

[23] A. Joux. A one round protocol for tripartite diffie-hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory (ANTS '00)*, volume 1838 of *LNCS*, pages 385–394. Springer-Verlag, 2000.

[24] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Proceedings of Advances in Cryptology - ASIACRYPT '96*, volume 1163 of *LNCS*, pages 36–49. Springer-Verlag, 1996.

[25] J. Kilian and E. Petrank. An efficient noninteractive zero-knowledge proof system for np with general assumptions. *Journal of Cryptology*, 11(1):1–27, 1988.

[26] B. Klein, M. Otten, and T. Beth. Conference key distribution protocols in distributed systems. In *Proceedings of 4th IMA Conference on Cryptography and Coding*, pages 225–242, 1995.

[27] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Inc., 1996.

[28] O. Pereira and J.-J. Quisquater. A security analysis of the Cliques protocols suites. In *Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 73–81. IEEE Computer Society Press, 2001.

[29] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security (CCS '96)*, pages 31–37. ACM Press, 1996.

[30] W.-G. Tzeng. A practical and secure-fault-tolerant conference-key agreement protocol. In *Proceedings of the Public-Key Cryptography (PKC '00)*, volume 1751 of *LNCS*, pages 1–13. Springer-Verlag, 2000.

[31] W.-G. Tzeng and Z.-J. Tzeng. Round-efficient conference key agreement protocols with provable security. In *Proceedings of Advances in Cryptology - ASIACRYPT '00*, volume 1976 of *LNCS*, pages 614–628. Springer-Verlag, 2000.