

國立交通大學

網路工程研究所

碩 士 論 文

一個可證明且伺服器端驅動的密碼認證  
金鑰交換制

A Provable Server-Triggering Password-based  
Authenticated Key Exchange Protocol

研 究 生：蔡家宏


指導教授：曾文貴 教授

中 華 民 國 九 十 八 年 六 月

# 國立交通大學

## 網路工程研究所

### 碩士論文



一個可證明且伺服器端驅動的密碼認證  
金鑰交換制

A Provable Server-Triggering Password-based  
Authenticated Key Exchange Protocol

研究生：蔡家宏

指導教授：曾文貴 教授

中華民國九十八年六月

一個可證明且伺服器端驅動的密碼認證金鑰交換機制  
A Provable Server-Triggering Password-based  
Authenticated Key Exchange Protocol

研究生：蔡家宏

Student : Chya-Hung Tsai

指導教授：曾文貴

Advisor : Weng-Guey Tzeng

國立交通大學  
網路工程研究所  
碩士論文



Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

# 一個可證明且伺服器端驅動的認證金鑰交換機制

學生：蔡家宏

指導教授：曾文貴 博士

國立交通大學網路工程研究所碩士班



基於密碼的認證金鑰交換機制僅需要使用者記憶一組安全度低的密碼即可完成運作，其方便性及彈性被廣泛應用在客戶/伺服器的架構上。其中在非對稱性的協定中，伺服器端儲存使用者密碼的相對應轉換值，而非直接儲存明文的密碼，如此一來，即使伺服器遭到入侵也不會立即洩漏出使用者的密碼。近年來，許多基於密碼的金鑰交換協定被提出來，這些協定大部分都是由客戶端先發出訊息傳送至伺服器端，在這篇論文中，我們提出由伺服器端先發送訊息給客戶端的協定。在這樣的架構下，當同時有許多客戶與伺服器連線時，伺服器端可以控制流量，避免計算

資源被大量消耗。更進一步我們也正規的證明了我們所提出的協定在 random oracle model 下是安全的，同時我們利用了 CDH 以及 S-CDH 兩個困難的問題在我們的證明中。

關鍵字：金鑰交換、認證金鑰交換、基於密碼的認證金鑰交換



# A Provable Server-Triggering Password-based Authenticated Key Exchange Protocol

Student: Chya-Hung Tsai

Advisor: Dr. Wen-Guey Tzeng

Institute of Network Engineering College of Computer Science

National Chiao Tung University

## Abstract

Since it is convenient for users to memorize a low-entropy password, the password-based authentication key exchange (PAKE) protocols have been an active research topic on the client/server-based communication. Especially, the asymmetric protocols which the server stores the password images are resistant to the leak of passwords when the server becomes compromised. Many elegant protocols are proposed in the past. However, most of them will first send the short-term information to the server from client.

In this paper, we propose a provable server-triggering password-based authenticated key exchange protocol (ST-PAKE). We focus on the framework that the server generates the short-term information first and then sends it to the client. This idea has some advantage for communication. For example, when there are a large number of clients connecting to the server, the server can select which client to communicate according to the order of preference.

Also, we confront a kind of off-line dictionary attack. We call it active dictionary attack. This attack can be successfully mounted if the protocol

is not well-design. We modify our ST-PAKE protocol to the ST-PAKE-A, which is designed to resist to the active dictionary attack.

Moreover, our scheme is provably forward secrecy and resilient to the server compromise. We provide a formal security proof of our scheme under the CDH assumption and the S-CDH assumption in the random oracle model.

**Keywords:** key exchange, authenticated key exchange protocol, password-based authentication key exchange



## 誌

## 謝

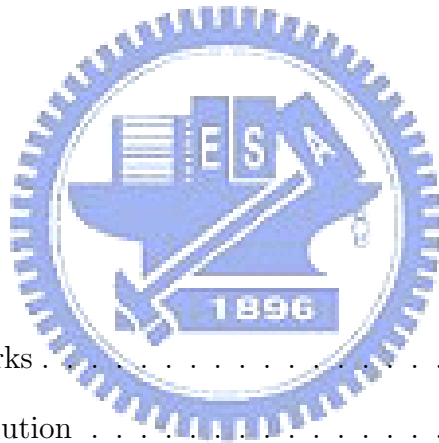
首先感謝我的指導老師曾文貴教授，傳授我許多關於密碼理論的知識，讓我對這個領域有更深一層的認識，並了解所謂的安全並非口說無憑，而是需要提供一套完整的正規證明。另外，我要感謝口試委員，中研院呂及人教授、交大謝續平教授與交大蔡錫鈞教授，在論文中給我許多建議與指導，讓我的論文更加完善。除此之外，我也要感謝實驗室的孝盈學姊、思維、毅睿、韶瑩和學弟們對我的幫忙。特別感謝畢業學長成康以及助理天心的協助，還有新竹的成大幫陪我度過第三個兩年。

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



# Contents

Abstract in Chinese	i
Abstract	iii
Acknowledgement	v
Table of Contents	vi
<b>1 Introduction</b>	<b>1</b>
1.1 Related Works . . . . .	3
1.2 Our Contribution . . . . .	4
1.3 Organization . . . . .	6
<b>2 Definition</b>	<b>7</b>
2.1 Security Model . . . . .	7
2.2 Security Assumptions . . . . .	12
<b>3 The Server-triggering Password-based Authenticated Key Exchange Protocol</b>	<b>14</b>

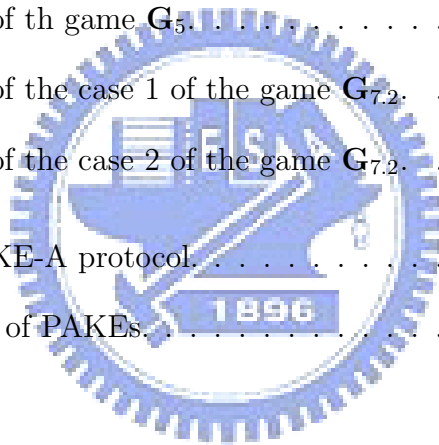


<b>4</b>	<b>Security Proof</b>	<b>17</b>
<b>5</b>	<b>Discussion</b>	<b>29</b>
5.1	Other security issues . . . . .	29
5.2	Comparison . . . . .	32
<b>6</b>	<b>Conclusion</b>	<b>34</b>
	<b>Bibliography</b>	<b>35</b>



# List of Figures

3.1	The ST-PAKE protocol. . . . .	16
4.2	Simulation of the oracle queries. . . . .	20
4.3	Simulation of th game $G_5$ . . . . .	23
4.4	Simulation of the case 1 of the game $G_{7.2}$ . . . . .	26
4.5	Simulation of the case 2 of the game $G_{7.2}$ . . . . .	27
5.1	The ST-PAKE-A protocol. . . . .	31
5.2	Comparison of PAKEs. . . . .	33



# Chapter 1

## Introduction

How to establish a secure channel between two communicating parties is an important problem on the networks. Generally speaking, a public key exchange protocol (KE) allows two parties who share no secret information to compute a shared session key over a public channel [DH76]. Due to lack of identity authentication, it's vulnerable to the man in the middle attack. The authenticated key exchange (AKE) protocols not only generate the session key between two parties, but also ensure authenticity of the involved parties. There are many efficient AKE protocols. Some of them require the users to hold devices, mainly for holding secret keys of a public key system. It is sometimes inconvenient. Therefore it is preferred to establish secure channels on a short memorable secret.

Password-based authenticated key exchange (PAKE) protocols are attractive, since each user needs to remember just one password. However, it is hard to design such protocols since low-entropy passwords are vulnerable to malicious dictionary attacks. A secure PAKE protocol should meet the

following requirements for the dictionary attacks.

- *Resilient to on-line dictionary attacks*: Even if the active adversary deviates from the protocol arbitrarily, he cannot eliminate a significant number of possible passwords.
- *Resilient to off-line dictionary attacks*: Any eavesdropper who records the transcripts of one or more sessions cannot eliminate a significant number of possible passwords by examining the recorded transcripts.

The on-line dictionary attacks can be easily avoided by the server. The server can reject the connection from the client after a few failure tries of login. So we focus on resilience to the off-line dictionary attacks.

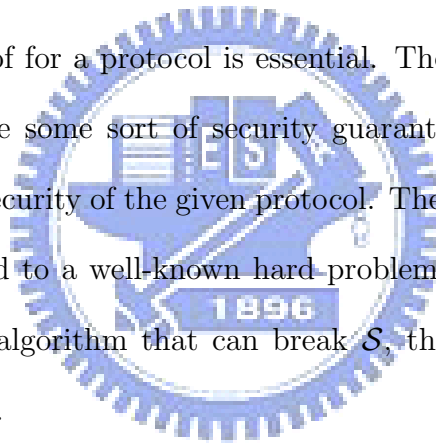
PAKE protocols are also useful under the client-server architecture. In this architecture, the server needs to store the identities of the clients and the corresponding verification values for passwords which are called verifiers. The server can check the validity of incoming messages from the clients via the verifiers. In addition to resilience to the dictionary attacks, the protocols should meet the following security requirements.

- *Forward secrecy*: The compromise of the long-term keys of a set of principals does not reveal the session keys established in previous protocol runs involving those principals.
- *Resilient to the server compromise*: The compromise of the secret information of the server does not compromise the session key established

in the current session.

The forward secrecy models the case that even the long-term keys are revealed some day, the session keys generated in the old sessions are still safe. This is necessary since we cannot ensure the safety of the long-term keys in a very long period of time. In a risky environment, it is possible that the server is compromised. The adversary may obtain all the information stored in the server. Even if all the information in the server is revealed, the adversary can't get more information about the current session key.

The security proof for a protocol is essential. The protocols for security purposes should have some sort of security guarantee. We usually give a reduction proof for security of the given protocol. The security of the subject protocol  $\mathcal{S}$  is reduced to a well-known hard problem  $\mathcal{P}$  in the sense that if there is an efficient algorithm that can break  $\mathcal{S}$ , then there is an efficient algorithm to solve  $\mathcal{P}$ .



## 1.1 Related Works

Password-based authenticated key exchange (PAKE) has been extensively studied in the last few years. The first PAKE scheme is introduced in 1989 [LGSN89]. The seminal work in this area is the encrypted key exchange protocol (EKE) proposed by Bellare and Merritt [BM92]. EKE proposes a novel method that using a long-term symmetry key to protect a short-term public key. This method can effectively resist to the off-line dictio-

nary attacks. The original EKE protocol requires that the server stores the plaintext password in order to decrypt the message sent from the client. Subsequently Bellare and Merritt propose an augmented EKE protocol (A-EKE) [BM93], which requires the server to store the verifier of the password. Even the adversary compromises the server; he can't extract the password from the verifier easily. It's also called the asymmetric protocol. Due to its simplicity and convenience, many proposed protocols are based on A-EKE [Wu98, Kwo01, Mac02, Jab97]. However, the above schemes or improved schemes lack a formal security proof. The first security model for 2-party authenticated key exchange protocols is introduced by Bellare and Rogaway [BR93]. Since then, a number of provable secure protocols and improvements have been proposed [BCP03, Kwo04, WZ06, PNKW07, SCWL07].

The AMP (Authentication via Memorable Password) protocol of Kwon [Kwo01] is a variant of A-EKE. It is also a standard for IEEE P1363. Nevertheless, the AMP protocol does not have a formal security proof. Kwon later proposes TP-AMP [Kwo04] which reduced one-round from the original four-round AMP and gives a formal security proof.

## 1.2 Our Contribution

In this paper, we propose a practical server-triggering password-based authenticated key exchange protocol (ST-PAKE). Different from the most of previous protocols, our protocol is the scheme that the server generates the

short-term information first and is proved secure. In most network protocols, the client and the server run the handshaking protocol before executing the subsequent task. Our protocol is suitable for the following situation. When there are a large number of clients connecting with the server, the server can select which clients to communicate with and efficiently control the flows. It prevents the server from the exhaustion of the computational resources. In contrast, the traditional key exchange protocols are triggered by the client. It may be that a large number of clients triggering the protocol simultaneously. This cause heavy system load.

In addition to the passive dictionary attack, we confront a kind of off-line dictionary attack. We call it active dictionary attack. This attack can be successfully mounted in our ST-PAKE protocols. We modify our ST-PAKE protocol to the ST-PAKE-A, which is designed to resist to the active dictionary attack.

Our protocol is based on the A-EKE protocol. We use the verifier as the long-term secret key to protect the message from the client. The server can use the verifier to decrypt the message and compute the session key. Our scheme is provably secure with forward secrecy and resilience to the server compromise attack under the CDH assumption and the S-CDH assumption in the random oracle model.



## 1.3 Organization

The remainder of this thesis is organized as follows. The chapter 2 describes the security model and two security assumptions for our security proof. Chapter 3 introduces our new ST-PAKE protocol for detail. Chapter 4 gives a formal security proof for our ST-PAKE protocol. Chapter 5 discusses the active dictionary and gives the comparison of our ST-PAK protocol with other famous PAKE protocols. Finally, chapter 6 gives a conclusion for this paper.



# Chapter 2

## Definition

### 2.1 Security Model

In this section, we recall the security model of Bellare et al. [BPR00] and Bresson et al. [BCP03] for password-based key exchange protocols.

**Participants:** We define a set *PARTY* of users, and partition them into two finite sets *CLIENT* and *SERVER*. Each participant in the password-based key exchange protocol is either a client  $C \in CLIENT$  or a server  $S \in SERVER$ .

**Long-Term keys:** Each client  $C \in CLIENT$  has a secret password  $\pi_c$  and each server  $S \in SERVER$  has a table which stores the verifiers  $V(\pi_c)$  for passwords. The verifier can be used to verify the client's password.

**Protocol Execution:** The capabilities of an adversary  $\mathcal{A}$  are modeled in the form of oracle queries. For all  $U \in \{C, S\}$ , let  $U^i$  denote the instance  $i$  of a party  $U$  and  $U^i$  is considered as an oracle. When the adversary wants

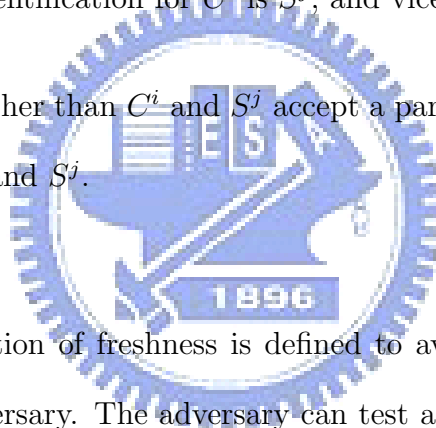
to invoke a party in the protocol, he must create an instance of a party via oracle queries. The query types available to the adversary are as follow:

- **Send**( $M, U^i$ ): This query models the active attack in which the adversary may modify or inject the message on the public channel. The output of this query is a message that the oracle  $U^i$  would generate upon receipt of message  $M$ .
- **Execute**( $C^i, S^j$ ): This query models the passive attack in which the adversary eavesdrops on honest execution between a client instance  $C^i$  and a server instance  $S^j$ . The output of this query consists of the messages that were exchanged during the honest execution of the protocol.
- **Reveal**( $U^i$ ): This query models the compromise of session keys by the adversary. The output of this query is the session key held by the instance  $U^i$ .
- **Corrupt**( $U$ ): This query models the compromise of the password (client) or the transformed-password (server) by the adversary, in order to deal with forward secrecy. The output of this query is the password or its verifier.
- **Test**( $U^i$ ): This query models the semantic security of session keys. It is answered by flipping a coin  $b$ . If  $b = 1$ , then return the real session

key. Otherwise, a random value is returned.

**Partnering:** Two instance  $C^i$  and  $S^j$  are said to be partners if all of the following condition hold:

- (1) Both of  $C^i$  and  $S^j$  accept.
- (2)  $C^i$  and  $S^j$  share the same session.
- (3) The partner identification for  $C^i$  is  $S^j$ , and vice versa.
- (4) No instances other than  $C^i$  and  $S^j$  accept a partner identification that is equal to  $C^i$  and  $S^j$ .



**Freshness:** The notion of freshness is defined to avoid the trivial attacks mounted by the adversary. The adversary can test a fresh session only. We now define two freshness notions *fresh* and *semi-fresh*. Note that the *fresh* is defined only for the case that the server is not compromised and the *semi-fresh* is defined in order to consider the resistance to server compromise. Furthermore, we add the requirement of the forward secrecy into the above two cases. So we describe all the four combinations respectively.

-An instance  $U^i$  is **nfs-fresh** (*fresh* with no requirement for forward secrecy) unless all of the following events do not occur:

(1) a  $\text{Reveal}(U^i)$  query occurs

(2) a  $\text{Reveal}(U^j)$  query occurs where  $U^j$  is the partner of  $U^i$

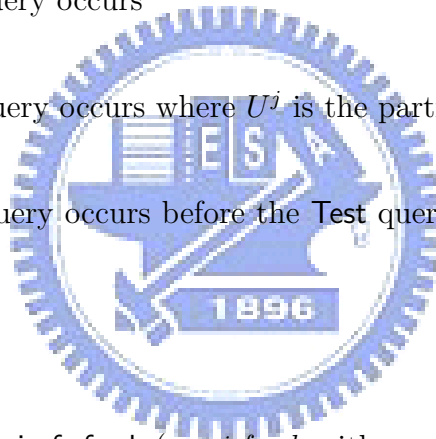
(3) a  $\text{Corrupt}(U)$  query occurs

-An instance  $U^i$  is **fs-fresh** (*fresh* with forward secrecy) unless all of the following events do not occur:

(1) a  $\text{Reveal}(U^i)$  query occurs

(2) a  $\text{Reveal}(U^j)$  query occurs where  $U^j$  is the partner of  $U^i$

(3) a  $\text{Corrupt}(U)$  query occurs before the  $\text{Test}$  query and the  $\text{Send}(U^i, M)$  query



-An instance  $U^i$  is **semi-nfs-fresh** (*semi-fresh* with no requirement for forward secrecy) unless all of the following events do not occur:

(1) a  $\text{Reveal}(U^i)$  query occurs

(2) a  $\text{Reveal}(U^j)$  query occurs where  $U^j$  is the partner of  $U^i$

(3) a  $\text{Corrupt}(C)$  query occurs

(4) a  $U \in \text{CLIENT}$  and  $\text{Corrupt}(S)$  query occurs

-An instance  $U^i$  is **semi-fs-fresh** (*semi-fresh* with forward secrecy) unless all of the following events do not occur:

- (1) a  $\text{Reveal}(U^i)$  query occurs
- (2) a  $\text{Reveal}(U^j)$  query occurs where  $U^j$  is the partner of  $U^i$
- (3) a  $\text{Corrupt}(C)$  query occurs before the  $\text{Test}$  query and the  $\text{Send}(U^i, M)$  query
- (4) a  $U \in \text{CLIENT}$  and  $\text{Corrupt}(S)$  query occurs before the  $\text{Test}$  query and the  $\text{Send}(U^i, M)$  query

**Semantic Security:** Consider an execution of the key exchange protocol  $\mathcal{P}$  by the adversary  $\mathcal{A}$ , who can given access to the  $\text{Reveal}$ ,  $\text{Execute}$ ,  $\text{Send}$  oracles and ask a single  $\text{Test}$  query to a  $\text{nfs-fresh}$  instance. Then, outputs a guess bit  $b'$ . If  $b' = b$ , we say that  $\mathcal{A}$  wins the game. The advantage of the adversary  $\mathcal{A}$  in the security game of the AKE protocol is defined as by  $\text{Adv}_{\mathcal{P}}^{\text{nfs-ake}}(\mathcal{A}) \stackrel{\text{def}}{=} |2 \Pr[b = b'] - 1|$ . Similarly, we also define  $\text{Adv}_{\mathcal{P}}^{\text{fs-ake}}(\mathcal{A})$ ,  $\text{Adv}_{\mathcal{P}}^{\text{s.nfs-ake}}(\mathcal{A})$  and  $\text{Adv}_{\mathcal{P}}^{\text{s.fs-ake}}(\mathcal{A})$  for the different secure proposes.

## 2.2 Security Assumptions

The security proof of our protocol is based on the CDH and S-CDH assumptions.

**Computational Diffie-Hellman Assumption:** The CDH assumption states that given  $g^u$  and  $g^v$ , where  $u$  and  $v$  are drawn from  $\mathbb{Z}_p$  randomly, it is hard to compute  $g^{uv}$ .

The CDH assumption is defined formally by considering an Experiment  $\mathbf{Exp}_{\mathbb{G}}^{\text{CDH}}(\mathcal{A})$ , in which we select two values  $u$  and  $v$  in  $\mathbb{Z}_p$ , compute  $U = g^u$ , and  $V = g^v$ , and then give  $U$  and  $V$  to the adversary  $\mathcal{A}$ . Let  $Z$  be the output of  $\mathcal{A}$ . Then, the Experiment  $\mathbf{Exp}_{\mathbb{G}}^{\text{CDH}}(\mathcal{A})$  outputs 1 if  $Z = g^{uv}$  and 0 otherwise. Then, we define the advantage of  $\mathcal{A}$  in violating the CDH assumption as  $\mathbf{Adv}_{\mathbb{G}}^{\text{CDH}}(\mathcal{A}) = \Pr[\mathbf{Exp}_{\mathbb{G}}^{\text{CDH}}(\mathcal{A}) = 1]$  and the advantage function  $\mathbf{Adv}_{\mathbb{G}}^{\text{CDH}}(t)$ , as the maximum value of  $\mathbf{Adv}_{\mathbb{G}}^{\text{CDH}}(\mathcal{A})$  over all  $\mathcal{A}$  with time-complexity at most  $t$ .

**Strong-CDH Assumption:** The strong-CDH assumption (S-CDH) states that given  $g^u$  and  $g^v$ , where  $u$  and  $v$  are drawn randomly from  $\mathbb{Z}_p$ , it is hard to compute the pair  $(Q, T)$  with  $Q = (\frac{T}{g^{uv}})^u$ , and  $T$  is an arbitrary value.

The S-CDH assumption is defined formally by considering an Experiment  $\mathbf{Exp}_{\mathbb{G}}^{\text{xxx}}(\mathcal{A})$ , in which we select two values  $u$  and  $v$  in  $\mathbb{Z}_p$ , compute  $U = g^u$ , and  $V = g^v$ , and then give  $U$  and  $V$  to the adversary  $\mathcal{A}$ . Let  $(Q, T)$  be the

output of  $\mathcal{A}$ . Then, the Experiment  $\mathbf{Exp}_{\mathbb{G}}^{\text{S-CDH}}(\mathcal{A})$  outputs 1 if  $Q = (\frac{T}{g^{uv}})^u$  and 0 otherwise. Then, we define the advantage of  $\mathcal{A}$  in violating the S-CDH assumption as  $\mathbf{Adv}_{\mathbb{G}}^{\text{S-CDH}}(\mathcal{A}) = \Pr[\mathbf{Exp}_{\mathbb{G}}^{\text{S-CDH}}(\mathcal{A}) = 1]$  and the advantage function  $\mathbf{Adv}_{\mathbb{G}}^{\text{S-CDH}}(t)$ , as the maximum value of  $\mathbf{Adv}_{\mathbb{G}}^{\text{S-CDH}}(\mathcal{A})$  over all  $\mathcal{A}$  with time-complexity at most  $t$ .





## Chapter 3

# The Server-triggering Password-based Authenticated Key Exchange Protocol

Our protocol uses a cyclic group  $\mathbb{G} = \langle g \rangle$  of order a  $k$ -bit prime number  $q$  and four hash functions,  $h_0 : \{0, 1\}^* \rightarrow \{0, 1\}^k$  and  $H_i : \{0, 1\}^* \rightarrow \{0, 1\}^{k'}, i \in \{0, 1, 2\}$ , where  $k$  and  $k'$  are security parameters.

As shown in Figure 3.1, the protocol runs between a client  $C$  and a server  $S$ . The client  $C$  holds the password  $\pi$  and the server  $S$  holds the corresponding verifier  $\gamma$ . We assume that the password is a low-entropy string that is chosen uniformly from the password space  $\Lambda$  with size  $|\Lambda| = N$ . The public information, such as the group  $\mathbb{G}$ , the generator  $g$  and the hash functions are known to all participants. Note that all computations are in  $\mathbb{G}$  except for stating otherwise.

The protocol consists of three flows and is initiated by the server.

1. The server  $S$  chooses a random value  $y$ , sets  $Y = g^y$ , and then sends  $(Y, C)$  to the client  $C$ .
2. After receiving  $(Y, C)$  from  $S$ ,  $C$  chooses a random value  $x$  and sets  $X^* = (YX)^u$  with  $X = g^x$ . To obtain the DH value,  $C$  computes  $\alpha = Y^{u \cdot x}$  and sets  $V_C = H_0(C, S, X^*, Y, \alpha, \gamma)$  as the authenticator. We can see that  $X^*$  depends on the previous flow from  $S$ . Finally,  $((X^*, V_C), S)$  is sent to  $S$ .
3. On receiving  $((X^*, V_C), S)$ ,  $S$  computes the DH value  $\beta = (X^*/\gamma^y)^y$  and checks whether  $V_C \stackrel{?}{=} H_0(C, S, X^*, Y, \beta, \gamma)$ . If they are not equal,  $S$  aborts the session. Otherwise,  $S$  computes  $V_S = H_1(C, S, X^*, Y, \beta, \gamma)$  as its authenticator. Finally,  $(V_S, C)$  is sent to  $C$ . After this step,  $S$  computes the session key  $SK_S = H_2(C, S, X^*, Y, \beta, \gamma)$ .
4. On receiving the  $S$ 's authenticator,  $C$  checks whether  $V_S = H_1(C, S, X^*, Y, \alpha, \gamma)$ . If they are not equal,  $C$  aborts the session. Otherwise,  $C$  computes the session key  $SK_S = H_2(C, S, X^*, Y, \alpha, \gamma)$ .

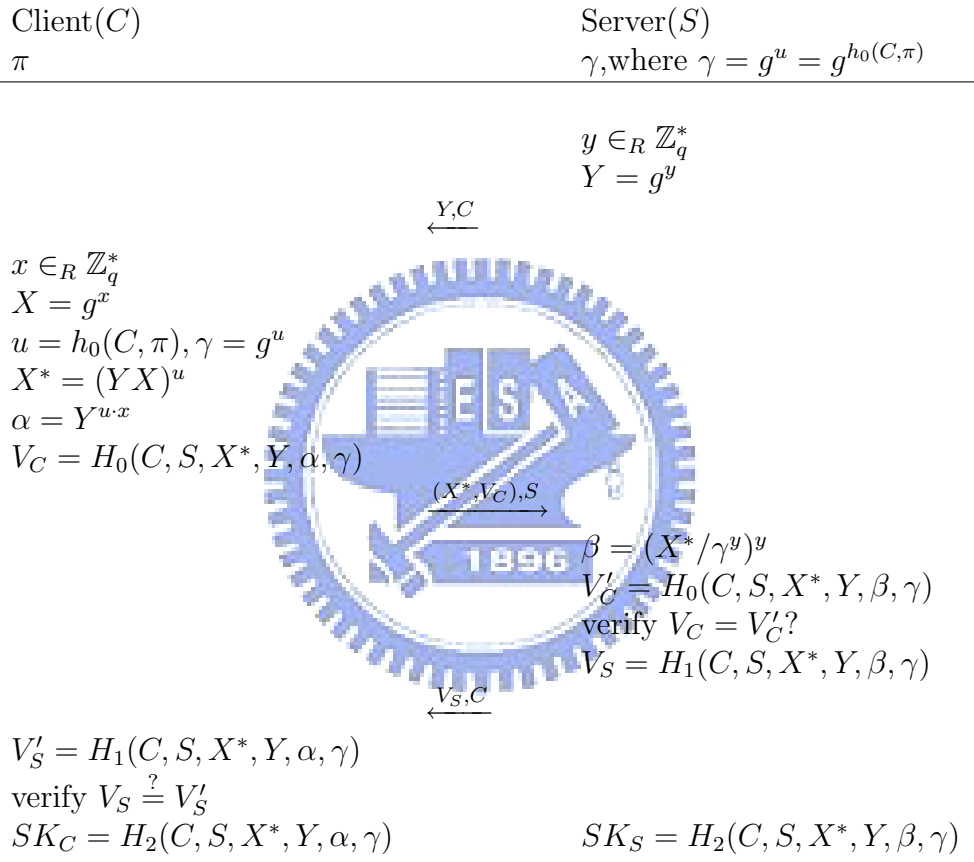


Figure 3.1: The ST-PAKE protocol.

# Chapter 4

## Security Proof

In this section, we prove that our protocol is secure in the random oracle model by using the CDH assumption and the S-CDH assumption. We use the security model mentioned before and accomplish our proof with the semantic security goal.

**Theorem 1.** *Let  $\mathcal{P}$  be the ST-PAKE protocol in Figure 3.1. with a uniformly distributed dictionary of size  $|\Lambda| = N$ .  $\mathcal{A}$  is the adversary who makes  $q_{send}$  queries to *Send* oracle,  $q_{exe}$  queries to *Execute* oracle, and  $q_{hash}$  queries to *Hash* oracles. Denote  $t'$  the running time of the CDH-solver and  $t''$  the running time of the S-CDH-solver. Then the advantage of the adversary  $\mathcal{A}$  is bounded by:*

$$\begin{aligned} \text{Adv}_{\mathcal{P}}^{\text{ake}}(\mathcal{A}) &\leq 2q_{ro}(q_{send} + q_{exe}) \cdot (\text{Adv}_{\mathbb{G}_q}^{\text{CDH}}(t', q_{ro}) + \text{Adv}_{\mathbb{G}_q}^{\text{S-CDH}}(t'', q_{ro})) \\ &\quad + \frac{2q_{send}}{N} + \frac{(q_{send} + q_{exe})^2}{q - 1} + \frac{(q_{ro} + q_{send})^2}{2^{k'}} \end{aligned}$$

*Proof.* We incrementally define a sequence of games starting at the real attack game  $\mathbf{G}_0$  and ending up at game  $\mathbf{G}_7$ . In the final stage, we give a bound for  $\text{Adv}_{\mathcal{P}}^{\text{ake}}(\mathcal{A})$  by using the triangle inequality to sum up the probability in each pair of neighboring games.

**Game  $\mathbf{G}_0$ :** This game represents the real attack game in the random oracle model. The oracles are defined previously that including four hash oracles  $(h_0, H_0, H_1, H_2)$  and all instances  $C^i$  and  $S^j$  can be available to adversary. We define the success event in the game  $\mathbf{G}_n$ :

- $\text{Succ}_n$ : This event occurs if  $b = b'$ , where  $b$  is the bit involved in the Test query, and  $b'$  is the output of the adversary after the Test game. In other word, this event means that the adversary wins the semantic game. By definition,

$$\text{Adv}_{\mathcal{P}}^{\text{ake}}(\mathcal{A}) \stackrel{\text{def}}{=} 2 \Pr[\text{Succ}_0] - 1.$$

**Game  $\mathbf{G}_1$ :** In this game, we simulate all the hash oracles for each query. We maintain three hash lists  $\mathcal{L}_h$ ,  $\mathcal{L}_H$  and  $\mathcal{L}_{\mathcal{A}}$  in order to the reason of consistency. The former two lists are used for random queries to hash oracles and the latter is used for hash queries asked by the adversary. Another list  $\mathcal{L}_{\mathcal{P}}$  is used to record the sending message between the instance  $C^i$  and  $S^j$ . We also simulate all the queries to any instance, including Send, Execute, Reveal, Corrupt and

**Test queries.** The rules for the simulation are described in Figure 4.2. Note that we omit descriptions of recording for oracle answers in the figure.

From this simulation, we can easily see that the transcript distribution of the game is identical to that of the real attack game in the random oracle model. Thus we have:

$$\Pr[\text{Succ}_1] = \Pr[\text{Succ}_0].$$

**Game  $\mathbf{G}_2$ :** This is the game to avoid the collision occurring in  $Y$  or  $X^*$  and in the hash queries asked by the adversary. In other words, if  $Y$  or  $X^*$  is randomly generated by honest parties and appeared in the old session of the protocol, the execution is aborted. Since either  $Y$  or  $X^*$  was simulated by choosing them randomly, according to the birthday paradox, the probability of collisions is bounded by  $\frac{(q_{send} + q_{exe})^2}{2(q-1)}$ . Also we modify the simulation of the hash oracle queries so that if  $\langle i, *, r \rangle \in \mathcal{L}_{\mathcal{A}}$  for a random value  $r \in \{0, 1\}^{k'}$ , then we abort the game with probability bounded by  $\frac{q_{ro}^2}{2^{k'+1}}$ .

The two games  $\mathbf{G}_2$  and  $\mathbf{G}_1$  are identical except for the above-mentioned collisions case. Thus the distance between  $\mathbf{G}_2$  and  $\mathbf{G}_1$  is:

$$|\Pr[\text{Succ}_2] - \Pr[\text{Succ}_1]| \leq \frac{(q_{send} + q_{exe})^2}{2(q-1)} + \frac{q_{ro}^2}{2^{k'+1}}.$$

**Game  $\mathbf{G}_3$ :** In this game, we exclude the case that the adversary has been lucky in guessing the authenticator  $V_C$  or  $V_S$  without querying the corre-

$-H_i(q)$ with $i \in \{0, 1, 2\}$	$-\text{Execute}(C^i, S^j)$
If $\langle i, q, r \rangle \notin \mathcal{L}_h$ $r \xleftarrow{R} \{0, 1\}^{k'}$ Add $\langle i, q, r \rangle$ to $\mathcal{L}_h$ If the query was asked by $\mathcal{A}$ Add $\langle i, q, r \rangle$ to $\mathcal{L}_A$ Return $r$	$(Y, C^i) \leftarrow \text{Send}(\text{Start}, S^j)$ $((X^*, V_C), S^j) \leftarrow \text{Send}(Y, C^i)$ $(V_S, C^i) \leftarrow \text{Send}((X^*, V_C), S^j)$ $\text{Send}(V_S, C^i)$ Return $((Y, C^i), ((X^*, V_C), S^j), (V_S, C^i))$
$-\text{Send}(\text{Start}, S^j)$	$-\text{Send}((X^*, V_C), S^j)$
$y \xleftarrow{R} \mathbb{Z}_{q^*}$ $Y \leftarrow g^y$ Return $(Y, C^i)$	$\beta \leftarrow (X^*/\gamma^y)^y$ $V'_C \leftarrow H_1(C, S, X^*, \beta, \gamma)$ If $V_C = V'_C$ $V_S = H_2(C, S, X^*, Y, \beta, \gamma)$ Return $(V_S, C^i)$ $SK_S \leftarrow H_3(C, S, X^*, Y, \beta, \gamma)$ Terminate
$-\text{Send}(Y, C^i)$	$-\text{Send}(V_S, C^i)$
$x \xleftarrow{R} \mathbb{Z}_{q^*}$ $X \leftarrow g^x$ $u \leftarrow H_0(C, \pi)$ $X^* \leftarrow (YX)^u$ $\alpha \leftarrow Y^{u \cdot x}$ $V_C \leftarrow H_1(C, S, X^*, Y, \alpha, \gamma)$ Return $((X^*, V_C), S^j)$	$V'_S = H_2(C, S, X^*, Y, \alpha, \gamma)$ If $V_S = V'_S$ $SK_C \leftarrow H_2(C, S, X^*, Y, \alpha, \gamma)$ Terminate
$-\text{Test}(U^i)$	$-\text{Corrupt}(U)$
$SK_U \leftarrow \text{Reveal}(U)$ $b \xleftarrow{R} \{0, 1\}$ $SK_U \xleftarrow{R} \{0, 1\}^{k'}$ Return $SK_U$	If $U = C$ Return $\pi$ Else if $U = S$ $u \leftarrow H_0(C, \pi)$ $\gamma \leftarrow g^u$ Return $\gamma$
$-\text{Reveal}(U^i)$	
Return $SK_U$	

Figure 4.2: Simulation of the oracle queries.

sponding random oracles. For this reason, we slightly modify the rule of the **Send** queries.

When invoking the **Send** $((X^*, V_C), S^j)$  query, the server first checks whether  $\langle 1, C, S, X^*, Y, \beta, \gamma, V_C \rangle \in \mathcal{L}_A$  or  $\langle (Y, C), ((X^*, V_C), S), * \rangle \in \mathcal{L}_P$ . If both tests fail, the server terminates without accepting.

Similarly, when invoking the **Send** $((V_S, C^i)$  query, the client first checks whether if  $\langle 1, C, S, X^*, Y, \alpha, \gamma, V_S \rangle \in \mathcal{L}_A$  or  $\langle (Y, C), ((X^*, V_C), S), (V_S, C) \rangle \in \mathcal{L}_P$ . If both tests fail, the client terminates without accepting. This modification ensures that the authenticator  $V_C$  and  $V_S$  must be generated by the simulator via correct random oracle queries.

Games  $\mathbf{G}_3$  and  $\mathbf{G}_2$  are identical unless the server (client) aborts without accepting the authenticator  $V_C(V_S)$ . This event happens only if the adversary has been lucky in guessing the correct value  $V_C(V_S)$  without querying the corresponding random oracle. So the distance between  $\mathbf{G}_3$  and  $\mathbf{G}_2$  is bounded by:

$$|\Pr[\text{Succ}_3] - \Pr[\text{Succ}_2]| \leq \frac{q_{send}}{2^{k'}}.$$

**Game  $\mathbf{G}_4$ :** In this game, we exclude the case that the adversary has been lucky in guessing the verifier  $u$  without querying the corresponding random oracle. We slightly modify the process of the hash oracle by adding an extra test before returning the random value.



When the  $H_i(C, S, X^*, Y, *, \gamma)(i \in \{0, 1, 2\})$  query is asked by the adversary and  $\langle 0, C, \pi, u \rangle \in \mathcal{L}_h$ , it must be checked whether  $\langle 0, C, \pi, u \rangle \notin \mathcal{L}_A$ . In this case, the game will abort. This game ensures that the value  $u$  must be derived from  $\pi$  by the random oracle. Games  $\mathbf{G}_4$  and  $\mathbf{G}_3$  are identical unless the adversary has correctly guessed the value  $u$  without asking the oracle. So we have:

$$|\Pr[\text{Succ}_4] - \Pr[\text{Succ}_3]| \leq \frac{q_{ro}}{2^{k'}}.$$

**Game  $\mathbf{G}_5$ :** This game simulates the case that a correct password guess is made by the adversary without using a `Corrupt` query and he furthermore asks `Execute` queries, then he succeeds in solving the session key by asking the hash query. We use a reduction from the CDH assumption to prove that the probability of this case is negligible.

Given a random CDH input  $(A, B)$  with  $A \leftarrow g^a$  and  $B \leftarrow g^b$ , the CDH-solver uses  $\mathcal{A}$  as a subroutine to solve the CDH problem and outputs  $\text{CDH}(A, B) = g^{ab}$ . We archive this goal by modifying the oracle answers, as shown in Figure 4.3. Note that the list  $\mathcal{L}_{\text{CDH}}$  is used to record the candidates of the CDH output.

Under the random oracle model, the adversary must query the hash oracle to win the game. We except that the element of the hash oracle input  $\Omega$  is

$\text{-Execute}(C^i, S^j)$	$H_i(C, S, X^*, Y, \Omega, \gamma)$ with $i \in \{0, 1, 2\}$
$Y \leftarrow A$	If $\langle (Y, C), ((X^*, V_C), S), (V_S, C) \rangle \in \mathcal{L}_{\mathcal{P}}$
$\varphi \xleftarrow{R} \mathbb{Z}_{q^*}$	and $\langle 0, C, \pi, u \rangle \in \mathcal{L}_{\mathcal{A}}$ without asking <b>Corrupt</b> ( $C$ )
$X^* \leftarrow (AB)^\varphi$	Adding $Z = \Omega^{\frac{-1}{\varphi}}$ to the $\mathcal{L}_{\text{CDH}}$
$V_C, V_S \xleftarrow{R} \{0, 1\}^{k'}$	Else
Return $\langle (Y, C), ((X^*, V_C), S), (V_S, C) \rangle$	The game aborts

Figure 4.3: Simulation of th game  $\mathbf{G}_5$ .

equal to  $\text{CDH}(A, B)^\varphi$ . The correct CDH output  $Z$  is picked up from the list  $\mathcal{L}_{\text{CDH}}$  with the probability at least  $\frac{1}{q_{ro}}$ . We denote  $t'$  the running time of the CDH-solver. Games  $\mathbf{G}_5$  and  $\mathbf{G}_4$  are indistinguishable unless the game aborts.

So we have:

$$|\Pr[\text{Succ}_5] - \Pr[\text{Succ}_4]| \leq q_{ro} \cdot \text{Adv}_{\mathbf{G}_q}^{\text{CDH}}(t', q_{ro}).$$

**Game  $\mathbf{G}_6$ :** In this game, we exclude the case that if the adversary has been lucky in guessing the password  $\pi$  without using a **Corrupt** query and he furthermore asks the **Send** queries. Since the password is the only secret information in the PAKE protocol, we disallow this case to happen. For this reason, we slightly modify the process of the **Send** query by adding an extra test.

When invoking the **Send**(( $X^*, V_C$ ),  $S^j$ ) query, the server first checks that if

$\langle 1, C, S, X^*, Y, \beta, \gamma, V_C \rangle \in \mathcal{L}_A$  or  $\langle (Y, C), ((X^*, V_C), S), * \rangle \in \mathcal{L}_P$ . If both tests fail, the server terminates without accepting. Otherwise, check if  $\langle 0, C, \pi, u \rangle \in \mathcal{L}_A$  and the **Corrupt** query has not been asked. If this is the case, we abort the game.

Similarly, when invoking the **Send** $((V_S, C^i)$  query, the client first checks that if  $\langle 1, C, S, X^*, Y, \alpha, \gamma, V_S \rangle \in \mathcal{L}_A$  or  $\langle (Y, C), ((X^*, V_C), S), (V_S, C) \rangle \in \mathcal{L}_P$ . If both tests fail, the client terminates without accepting. Otherwise, check if  $\langle 0, C, \pi, u \rangle \in \mathcal{L}_A$  and the **Corrupt** query has not been asked. If this is the case, we abort the game. This modification ensures that the correct password must be obtained via **Corrupt** query. The two games  $\mathbf{G}_6$  and  $\mathbf{G}_5$  are perfectly indistinguishable unless the adversary has correctly guessed the password without asking a **Corrupt** query and made the **Send** query. Hence,

$$|\Pr[\mathbf{Succ}_6] - \Pr[\mathbf{Succ}_5]| \leq \frac{q_{send}}{N}.$$

Now we partition the game  $\mathbf{G}_7$  into two independent sub-games for the two secure requirements, forward secrecy and server compromise.

**Game  $\mathbf{G}_{7.1}$  (forward secrecy):** In this game, we consider the forward secrecy. We need to ensure that old session keys still security after the LL-key is compromised. By the definition of freshness, the **Corrupt** $(C)$  query was made

by an adversary after the **Test** query but in the old game. We modify the current game to another corrupted game in a way to reply with old transcripts when an **Execute** query is asked. This situation is similar to  $\mathbf{G}_5$ . The only different thing is that how does the adversary obtain the password. In this game, the adversary obtains the password by means of making a **Corrupt**( $C$ ) query, instead of lucky guessing in  $\mathbf{G}_5$ . Since the simulator needs to consider the overhead of choosing the transcripts from the old sessions, the success probability of the adversary is bounded by  $(q_{send} + q_{exe})q_{ro} \cdot \text{Adv}_{\mathbb{G}_q}^{\text{CDH}}(t', q_{ro})$ . So we have:

$$|\Pr[\text{Succ}_{7.1}] - \Pr[\text{Succ}_6]| \leq (q_{send} + q_{exe})q_{ro} \cdot \text{Adv}_{\mathbb{G}_q}^{\text{CDH}}(t', q_{ro}).$$



**Game  $\mathbf{G}_{7.2}$  (server compromise):** In this game, we consider the case that server is compromised. In order to resist to server compromise, we allow the adversary to make a **Corrupt**( $S$ ) query before the **Test** query (**Test**( $S^j$ ) only) in our proof by the definition of semi-freshness. We modify the current game to another semi-corrupted game in the following ways. We discuss two cases that the adversary invokes the **Execute** queries or the **Send** queries. Note that the adversary obtains the value  $\gamma$  in the semi-corrupted game.

- If the adversary invokes the **Execute** query, then he succeeds in solving the session key by asking the hash query. We use a reduction from

the CDH problem to prove that the probability of this case occurs is negligible.

Given a random CDH input  $(A, B)$  with  $A \leftarrow g^a$  and  $B \leftarrow g^b$ , the CDH-solver uses  $\mathcal{A}$  as a subroutine to solve CDH and outputs  $\text{CDH}(A, B) = g^{ab}$ . We achieve this goal by modifying the oracle answers, as shown in Figure 4.4.

Under the random oracle model, the adversary must query the hash oracle to win the game. We except that the element of the hash oracle input  $\Omega$  is equal to  $\text{CDH}(A, B)^u$ . It is similar to the game  $\mathbf{G}_5$ , we could observe that this event may happen with probability bounded by  $q_{ro} \cdot \text{Adv}_{\mathbb{G}_q}^{\text{CDH}}(t', q_{ro})$ .

$\text{-Execute}(C^i, S^j)$	$H_i(C, S, X^*, Y, \Omega, \gamma)$ with $i \in \{0, 1, 2\}$
$Y \leftarrow A$ $X^* \leftarrow (AB)^u$ $V_C, V_S \xleftarrow{R} \{0, 1\}^{k'}$ Return $\langle (Y, C), ((X^*, V_C), S), (V_S, C) \rangle$	If $\langle (Y, C), ((X^*, V_C), S), (V_S, C) \rangle \in \mathcal{L}_{\mathcal{P}}$ and $\text{Corrupt}(S)$ has been queried Adding $Z = \Omega^{\frac{1}{u}}$ to the $\mathcal{L}_{\text{CDH}}$ Else Same as the previous rules.

Figure 4.4: Simulation of the case 1 of the game  $\mathbf{G}_{7.2}$ .

- If the adversary invokes the **Send** query, we use a reduction from the S-CDH problem.

Given a random S-CDH input  $(U, V)$  with  $A \leftarrow (g^u)$  and  $B \leftarrow (g^v)$ ,

the S-CDH-solver uses  $\mathcal{A}$  as a subroutine to solve S-CDH and outputs  $\text{S-CDH}(U, V) = (Q, T)$  with  $Q = (\frac{T}{g^{uv}})^u$  and  $T$  is a arbitrary value. We achieve this goal by modifying the oracle answers, as shown in Figure 4.5. We denote  $t''$  the running time of the S-CDH-solver. It is clear that this event may happen with probability bounded by  $q_{ro} \cdot \text{Adv}_{\mathbb{G}_q}^{\text{S-CDH}}(t'', q_{ro})$ .


-Corrupt( $U$ )	$H_i(C, S, T, Y, Q, \gamma)$ with $i \in \{0, 1, 2\}$
If $U = C$ abort Else if $U = S$ $\gamma \leftarrow U$ Return $\gamma$	 If $\langle (V, C), ((T, Q), S), * \rangle \notin \mathcal{L}_{\mathcal{P}}$ Adding $Z = Q$ to the $\mathcal{L}_{\text{S-CDH}}$ Else Same as the previous rules.
-Send( $Start, S^j$ )	
$Y \leftarrow V$ Return( $Y, C^i$ )	

Figure 4.5: Simulation of the case 2 of the game  $\mathbf{G}_{7.2}$ .

Combine these two cases, we have :

$$|\Pr[\text{Succ}_{7.2}] - \Pr[\text{Succ}_6]| \leq q_{ro}(\text{Adv}_{\mathbb{G}_q}^{\text{CDH}}(t', q_{ro}) + \text{Adv}_{\mathbb{G}_q}^{\text{S-CDH}}(t'', q_{ro})).$$

**Summary:** By the triangle inequality, we have  $\text{Adv}_{\mathcal{P}}^{\text{ake}}(\mathcal{A}) \leq 2 \sum_{i=0}^7 |\text{Pr}[\text{Succ}_{i+1}] - \text{Pr}[\text{Succ}_i]|$  and, thus:

$$\begin{aligned} \text{Adv}_{\mathcal{P}}^{\text{ake}}(\mathcal{A}) &\leq 2q_{ro}(q_{send} + q_{exe}) \cdot (\text{Adv}_{\mathbb{G}_q}^{\text{CDH}}(t', q_{ro}) + \text{Adv}_{\mathbb{G}_q}^{\text{S-CDH}}(t'', q_{ro})) \\ &\quad + \frac{2q_{send}}{N} + \frac{(q_{send} + q_{exe})^2}{q-1} + \frac{(q_{ro} + q_{send})^2}{2^{k'}} \end{aligned}$$

This is the complete proof. □



# Chapter 5

## Discussion

### 5.1 Other security issues

The traditional off-line dictionary attacks are mounted by a passive adversary who eavesdrops protocol messages and then goes off-line to perform the password search. In this section we consider another kind of off-line dictionary attacks. We call it the active dictionary attacks. This attack happens when the adversary impersonates the client (server) and communicates with the server (client). Then the adversary can obtain the messages from the server (client) and try to mount the off-line dictionary attacks by using the received information.

We consider the adversary impersonates the server and communicate with an honest client in our ST-PAKE protocol. The adversary can mount the active dictionary attacks by the following steps.

1. The adversary guesses a password  $\pi'$  and computes  $\gamma' = g^{u'} = g^{h_0(C, \pi')}$ .
2. In the first flow, the adversary chooses a random  $y'$  and sends  $Y' = g^{y'}$



to the client.

3. In the second flow, the client responds the  $X^*$  and  $V_C$  to the adversary.
4. The adversary computes the  $V_C' = H_0(C, S, X^*, Y', (\frac{X^*}{\gamma'^{y'}})^{y'}, \gamma')$  and compares it with  $V_C$ .
5. The adversary repeats the step 1 and 4 until he finds a password  $\pi$  such that  $V_C' = V_C$ .

We can see that the adversary mounts an active attack and obtains the password information from an honest client. Then he can try all possible passwords and verify them in private by using the received password information. This is a successful off-line dictionary attack.

The active dictionary attacks can be partitioned into two cases, the adversary impersonates the client or the server. It depend on the first authentication value is sent by the client or the server. If the first authentication value is sent by the client, then the adversary can try to impersonate the server. The ST-PAKE is in this case. Because the first authentication value  $V_C$  is sent by the client. The adversary can obtain this information and mount the dictionary attacks.

We modify our ST-PAKE to the ST-PAKE-A which can resist to the active dictionary attacks. In the first flow, the server sends  $Y^* = Y \oplus \gamma$  instead of  $Y$ . And the client computes  $Y = Y^* \oplus \gamma$  after receiving  $Y^*$ . This modification can ensure that the server must know  $\gamma$  otherwise he can't mount

the active dictionary attacks successfully. He only can probably eliminate at most one candidate password from consideration per **Send** query. The complete ST-PAKE-A is shown in Figure 5.1.

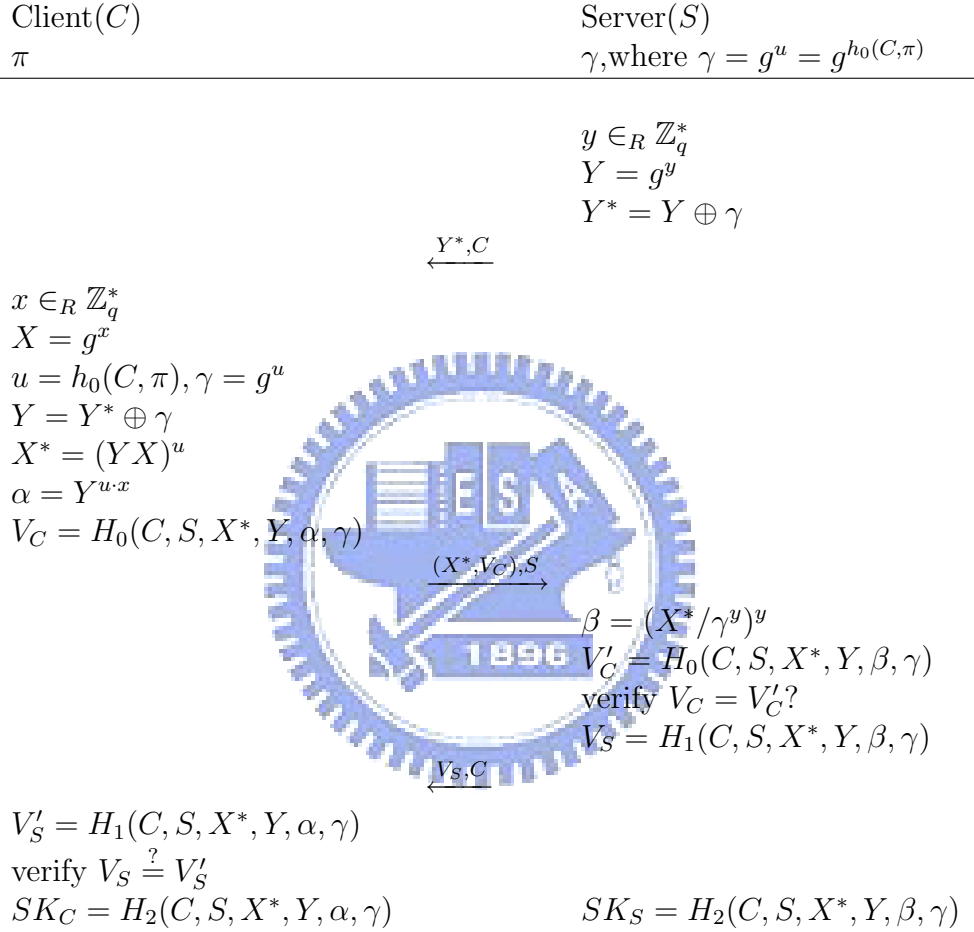


Figure 5.1: The ST-PAKE-A protocol.

## 5.2 Comparison

The comparison with other protocols is given in Figure 5.2. Only the asymmetric protocols are compared. We consider three factors: the number of flows, the number of exponentiations executed by the client and the server.

The mentioned protocols are all completed in three rounds or four rounds. We can see that AMP [Wu98] is the most efficient protocol because it has the minimum computational overhead mostly. But it doesn't have a formal security proof. TP-AMP [Kwo04] reduces one-round from the original four-round AMP and gave a formal security proof. Our protocol is a little bit inefficient than the TP-AMP protocol and same as the SRP [Wu98]. PAK-Y [Mac02] using the Schnorr signature for the authentication. So it needs more exponential operations for the signature. SCWL07 [SCWL07] is a PAKE protocol without public information. The all public information such as the prime, the group and the generator will be chosen by the client and sent to the server on the first flow. So this protocol is inefficient than other PAKE protocols.

The above protocols are all triggered by the client. However, our protocol is triggered by the server. This is the most different feature from others.

	Number of flows	Number of Exp. (client)	Number of Exp. (server)	Security proof	Triggering side
AMP [Wu98]	4	2	3	No	client
SRP [Kwo01]	4	3	3	No	client
PAK-Y [Mac02]	3	5	3	No	client
TP-AMP [Kwo04]	3	2	3	Yes	client
SCWL07 [SCWL07]	4	4	5	Yes	client
ST-PAKE	3	3	3	Yes	server

Figure 5.2: Comparison of PAKEs.

# Chapter 6

## Conclusion

In this paper, we propose a practical server-triggering password-based authenticated key exchange protocol. Different from most previous protocols, our protocol is the new scheme in which the server generates the short-term information first. This idea has a special feature that is useful for the client-server communication architecture. We also consider the active dictionary attack and given an improved version. Furthermore, we provide a formal security proof of our scheme under the CDH assumption and the S-CDH assumption in the random oracle model.

# Bibliography

- [AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2005.
- [BCP03] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 241–250. ACM, 2003.
- [BCP04] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New security results on encrypted key exchange. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 145–158. Springer, 2004.
- [BM92] Steve M. Bellare and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In

*IEEE Computer Society Symposium on Research in Security and Privacy, May 1992, Oakland, CA*, pages 72–84, 1992.

- [BM93] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security*, pages 244–250, 1993.
- [BM03] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag Berlin Heidelberg New York, 2003.
- [BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *EUROCRYPT*, pages 156–171, 2000.
- [BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pages 139–155, 2000.
- [BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

- [CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [GLNS93] Li Gong, T. Mark A. Lomas, Roger M. Needham, and Jerome H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.
- [Jab97] David P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *WETICE*, pages 248–255. IEEE Computer Society, 1997.
- [Kwo01] Taekyoung Kwon. Authentication and key agreement via memorable passwords. In *NDSS*. The Internet Society, 2001.
- [Kwo04] Taekyoung Kwon. Practical authenticated key agreement using passwords. In Kan Zhang and Yuliang Zheng, editors, *ISC*, volume 3225 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2004.



- [LGSN89] T. Mark A. Lomas, Li Gong, Jerome H. Saltzer, and Roger M. Needham. Reducing risks from poorly chosen keys. In *SOSP*, pages 14–18, 1989.
- [Mac01] Philip D. MacKenzie. More efficient password-authenticated key exchange. In David Naccache, editor, *CT-RSA*, volume 2020 of *Lecture Notes in Computer Science*, pages 361–377. Springer, 2001.
- [Mac02] Philip MacKenzie. The pak suite: Protocols for password-authenticated key exchange. In *IEEE P1363.2*, 2002.
- [PNKW07] Sangjoon Park, Junghyun Nam, Seungjoo Kim, and Dongho Won. Efficient password-authenticated key exchange based on rsa. In Masayuki Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2007.
- [SCWL07] Jun Shao, Zhenfu Cao, Licheng Wang, and Rongxing Lu. Efficient password-based authenticated key exchange without public information. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2007.
- [Wu98] Thomas D. Wu. The secure remote password protocol. In *NDSS*. The Internet Society, 1998.

- [WZ06] Shuhua Wu and Yuefei Zhu. Practical password-based authenticated key exchange protocol. In Yuping Wang, Yiu ming Cheung, and Hailin Liu, editors, *CIS*, volume 4456 of *Lecture Notes in Computer Science*, pages 523–533. Springer, 2006.

