Gateway Threshold Password-based Authenticated Key Exchange Secure against Undetectable On-line Dictionary Attack

Keywords: Password-based Authenticated Key Exchange (PAKE), Threshold Cryptography.

Abstract: Password-based Authenticated Key Exchange (PAKE) allows a server to authenticate a user and to establish a session key shared between the server and the user just by having memorable passwords. In PAKE, conventionally the server is assumed to have the authentication functionality and also provide on-line services simultaneously. However, in the real-life applications, this may not be the case, and the authentication server may be separate from on-line service providers. In such a case, there is a problem that a malicious service provider with no authentication functionality may be able to guess the passwords by interacting with other participants repeatedly. Abdalla et al. put forward a notion of the server password protection security to deal with this problem. However, their proposed schemes turned out to be vulnerable to Undetectable On-line Dictionary Attack (UDonDA). To cope with this situation, we propose the Gateway Threshold PAKE provably secure against this password guessing attack by also taking the corruption of authentication servers into consideration.

1 INTRODUCTION

1.1 Background

Password-based authenticated key exchange (PAKE) (Bellare et al., 2000; Bellovin et al., 1993; Bellovin et al., 1992; Boyko et al., 2000; Goldreich et al., 2001; Katz et al., 2001) is a two-party key exchange allowing users to utilize memorable passwords as secret information, where each password is shared between a user and an authentication server. Conventional authenticated key exchange protocols based on a public key cryptosystem need a key whose length is too long to remember. Since users cannot memorize such complicated information without devices, the users are unable to respond to any incident such as an emergency call unless the users have the devices. On the other hand, users can rely on PAKE protocols only with a short character string, and PAKE will also be suitable for cloud environments where ubiquitous access is important.

There are two cases where conventional PAKE cannot be deployed. The first case is that there exists a gateway between a user and an authentication server as in a global roaming service. The global roaming is a system where a user can receive the same services from an overseas provider even if the user is located outside service areas of the provider with which the user has made a contract. For example, when receiving on-line services via a foreign access point by using borrowed devices, a user has only to enter the password registered with the user's domestic provider. In the situation where a gateway called an access point plays a role of an on-line service provider, PAKE cannot be deployed in such environments because it is assumed that an authentication server provides services in PAKE.

The second case is that there exists a malicious authentication server. Actually, European Network and Information Security Agency (ENISA) (ENISA, 2009) pointed out that malicious providers could be a high threat with social influence. Even if a provider is honest, vulnerabilities on a system can also cause a problem such as Heartbleed in April 2014. In fact, the vulnerability of OpenSSL can cause a problem that the passwords stored in servers are leaked. ENISA pointed out that an adversary can potentially obtain the passwords of all users without authentication (ENISA, 2014). PAKE cannot defend against such a threat because a single trusted authentication server stores all the passwords.

Abdalla et al. (Abdalla et al., 2005; Abdalla et al., 2008) proposed schemes to solve those problems. Gateway PAKE (GPAKE) is a scheme addressing the first problem and Gateway Threshold PAKE (GTPAKE) is a scheme addressing both problems. However, their schemes are vulnerable to Undetectable On-line Dictionary Attack (UDonDA) (Ding et al., 1995), where an adversary guesses a password in on-line transaction and its password guessing attack is not detected by any authentication server. In their schemes, an authentication server returns a message without authenticating users, so the adversary can make unlimited attempts to guess a password. Due to the low entropy of the password, such a password guessing attack becomes a serious problem. Therefore, it is necessary to propose a new scheme that overcomes UDonDA.

1.2 Contribution

We propose new GTPAKE which has resistance of UDonDA and the corruption of authentication servers. We prove the security of our GTPAKE under standard assumptions in the random oracle model. The proposed scheme has the stronger security against a malicious provider compared with existing schemes, and a global roaming service used for users regardless of places and devices is expected as an application. Our scheme is an instantiation of GTPAKE, and the generalization of GPAKE and GTPAKE is left as future work.

A naive extension of GPAKE does not lead to GTPAKE with the property described in Section 1.1. The reason is as follows: If one authentication server holds plain passwords as in GPAKE, the server can just compare the received password with the corresponding plain password. However, as stored passwords should be hidden from authentication servers in GTPAKE, the authentication servers cannot easily verify logins of users. To overcome this problem, we encrypt the stored passwords by a public key of the authentication servers where the corresponding secret key is shared among the authentication servers. Furthermore, in the authentication process, the servers decrypt the encrypted password partially and authenticate a user simultaneously without revealing the password itself.

We compare the proposed scheme with other existing GPAKE and GTPAKE schemes. As shown in Table 1, the computation and communication costs of our protocol are not better than those of GTPAKE (Abdalla et al., 2005). However, their security proof reduces to the non-standard assumptions such as the Password-based Chosen-basis Decisional Diffie-Hellman (PCDDH) assumption, which is vulnerable to some attacks (Szydlo, 2006). Even if those flawed assumptions hold, their scheme is vulnerable to UDonDA, which is out of security model. The security of our scheme is proven in the random oracle model if the DDH assumption holds. Our scheme tolerates the corruption of some authentication servers as their scheme. Furthermore, while their scheme is vulnerable to UDonDA, our scheme is invulnerable to this attack, although our proof similar to (Wei et al., 2011) is given in the non-concurrent setting, which assumes that a new session does not begin until the previous session is finished.

The organization of the paper is as follows: In Section 2, we introduce some background to understand this paper. In Section 3, we define the security model of GTPAKE. In Section 4, we describe the construction of our scheme. In Section 5, we prove the security of the proposed scheme. Finally we make final remarks in Section 6.

2 PRELIMINARIES

We show the notation and the security assumption used in this paper.

Notation. We use the following notations throughout this paper. We denote by $\mathbb{Z}_q$ a set $\{0,1,\ldots,q-1\}$. $x \leftarrow A$ represents that x is chosen uniformly at random from a set A. Let g be a generator of subgroup $\mathbb{G}$ of order p over $\mathbb{Z}_q$ and $a \parallel b$ be a concatenation of elements a and b, which is able to be divided into the original elements. We denote by $\{0,1\}^k$ a set of all binary strings of length k. Especially, $\{0,1\}^*$ means a set of all binary strings of arbitrary length. A function negl is *negligible* if and only if for every positive integer c there exists an integer N such that $\mathsf{negl}(x) < 1/x^c$ for any $x > N$.

We represent a user as $U \in \mathcal{U}$, a gateway as $G \in \mathcal{G}$ and the i-th authentication server as $S_i \in \mathcal{S}$ for $i = 1,\ldots,n$ where $\mathcal{U}$, $\mathcal{G}$, and $\mathcal{S}$ are the set of all users, gateways, and authentication servers, respectively. Especially, we denote by P any participant in the set of all participants $\mathcal{P}$ $(= \mathcal{U} \cup \mathcal{G} \cup \mathcal{S})$. We call one representative of the authentication servers that communicates with a gateway a combiner C whose index is contained in the qualified index set L. The size of L is not less than threshold t.

Security Assumption. The following security assumptions are well-known:

Definition 1. (**Computational Diffie-Hellman (CDH) Assumption**). We define the Computational Diffie-Hellman (CDH) problem as the problem of computing g^{ab} from given (g, g^a, g^b) where g is a generator chosen at random from group $\mathbb{G}$ and $(a,b) \leftarrow \mathbb{Z}_q^2$. We say the CDH assumption holds in $\mathbb{G}$ if the advantage in solving the CDH problem defined as $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{CDH}}(\kappa) = \Pr[\mathcal{A}(g, g^a, g^b) = g^{ab}]$ is negligible for any probabilistic polynomial time algorithm $\mathcal{A}$ with respect to a security parameter κ.

Definition 2. (**Decisional Diffie-Hellman (DDH) Assumption**). We define the Decisional Diffie-Hellman (DDH) problem as the problem of distinguishing the distribution of (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) where g is a generator chosen at random from group $\mathbb{G}$ and $(a,b,c) \leftarrow \mathbb{Z}_q^3$. We say the DDH assumption holds in $\mathbb{G}$ if the advantage in solving the DDH problem defined as $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{DDH}}(\kappa) = |\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]|$ is negligible for any probabilistic polynomial time algorithm $\mathcal{A}$ with respect to a security parameter κ.

3 SECURITY MODEL

We describe the system model and security definitions of GTPAKE, which is a protocol which allows a legitimate user to establish a session key with a gateway with the help of multiple authentication servers. At the time of authentication, a user cannot directly communicate with an authentication server. Although the communication channel between a user and a gateway is insecure and under the control of an adversary, the channel between a gateway and a combiner is authenticated and the channel between the authentication servers is secure. When an authentication

Table 1: Comparison of the existing GPAKE and GTPAKE schemes.

Protocols	User	Gateway	Server	Message	Assumption	Model	Threshold	UDonDA
ACFP05	$2e$	$2e$	$2e$	4	PCDDH	ROM	NO	NO
ACFP05	$2e$	$2e$	$(13n+18)e$	$3n+4$	PCDDH	ROM	YES	NO
WMZ11	$3e$	$2e$	$2e$	6	DDH	ROM	NO	YES
WZM12	$5e+E$	$2e$	$5e+E$	6	DDH	Standard	NO	YES
WZM13	$3e$	$2e$	$2e$	9	CDH	ROM	NO	YES
Ours	$4e$	$2e$	$(2n^2+19n+11)e$	$4n+11$	DDH	ROM	YES	YES

The computational costs for a user, gateway, and each authentication server are estimated in the User, Gateway, and Server columns, respectively. We use a modular exponentiation denoted as "e" because a modular exponentiation is the most expensive computation and "E" means a cost of a public key encryption. For the sake of simplicity, n authentication servers participate in the authentication phase. In the Message column, the number of communications is shown and we evaluate a broadcast to all authentication servers as n communication costs. In the Assumption column, a hardness assumption is shown. In the Model column, the random oracle model or the standard model is shown. In the Threshold column, it is shown whether a protocol tolerates the corruptions of authentication servers. In the UDonDA column, it is shown whether a protocol can detect malicious login attempts for guessing the passwords of users.

process for a user (say, User A) is being processed, another login attempt from the same user (i.e., User A) is suspended until the preceding authentication process is finished. We assume a static adversary that corrupts the set of less than the threshold authentication servers before the protocol is executed.

3.1 System Model

The proposed scheme consists of the following three sub-protocols.

- **Init**. Given the security parameter and the setup parameters, public parameters *params* are outputted. Although we assume a trusted dealer distributing some parameters for simplicity, authentication servers themselves can publish parameters by using the technique of the distributed key generation (Gennaro et al., 2007). If the numbers of authentication servers are modified, this protocol phase is executed again.
- **Regi**. A user registers his password with authentication servers. If all authentication servers cannot register the password successfully, then outputs an error symbol $\perp$.
- **Auth**. The qualified servers the number of which is not less than the threshold authenticate a user. If the user and a gateway can establish the same session key while passing the authentication successfully, then outputs a session key *sk*, otherwise outputs $reject$.

3.2 Security Requirements

We describe the security requirements for GTPAKE. The technical details of reflecting these requirements are given in Definitions 4 and 6.

Existing Security Requirements. The security requirements for GPAKE are as follows (due to Wei et al. (Wei et al., 2011)):

- **Known-Key Security (KS)**. The adversary cannot distinguish a real session key from a random session key even if the adversary obtains other session keys.
- **Forward Secrecy (FS)**. The established session key before the adversary obtains the static keys of the user including passwords are still indistinguishable from a random session key.
- **Resistance to Basic Impersonation (BI)**. The adversary cannot impersonate a legitimate user unless the adversary obtains the password of the user.
- **Resistance to Off-line Dictionary Attack (offDA)**. The adversary acting as a malicious gateway cannot guess a password by verifying its guess in the off-line manner.
- **Resistance to Undetectable On-line Dictionary Attack (UDonDA)**. The adversary acting as a malicious gateway cannot guess a password by verifying its guess in the on-line manner without being detected by honest participants.

New Security Requirement. We add a security requirement necessary for the threshold setting. In GPAKE, an authentication server potentially acts only as a passive adversary, but in GTPAKE, we need to take an active adversary corrupting a set of less than the threshold authentication servers into consideration.

- **Resistance to leakage of internal information to servers (LIS)**. The adversary cannot distinguish a real session key from a random session key and cannot guess a password even if the adversary obtains internal information of some authentication servers.

3.3 Oracles

We show the necessary oracles to define the stronger security model than that of GTPAKE (Abdalla et al., 2005). To distinguish the session between participants, the i-th instance for participant P is denoted by $P^{(i)}$. Let $U^{(i)}$, $G^{(j)}$, $C^{(k)}$, and $S_i^{(k)}$ be instances of a user, a gateway, a combiner, and the i-th authentication server, respectively. The instance of these oracles defined here reflects the state during the progress of the protocol.

Existing Oracles (Wei et al., 2011). These oracles used in existing GPAKE are as follows:

- $\mathsf{Execute}(U^{(i)}, G^{(j)}, C^{(k)})$. This query models passive attacks. The output of this query consists of the message exchanged during the honest execution in the protocol among $U^{(i)}$, $G^{(j)}$, and $C^{(k)}$.
- $\mathsf{SendUser}(U^{(i)}, m)$. This query models active attacks against a user instance $U^{(i)}$.[1] The output of this query consists of the message the user instance $U^{(i)}$ would generate on receipt of message m.
- $\mathsf{SendGateway}(G^{(j)}, m)$. This query models active attacks against a gateway instance $G^{(j)}$.[1] The output of this query consists of the message the gateway instance $G^{(j)}$ would generate on receipt of message m.
- $\mathsf{SendServer}(C^{(k)}, m)$. This query models active attacks against a combiner instance $C^{(k)}$.[1] The output of this query consists of the message the combiner instance $C^{(k)}$ would generate on receipt of message m.
- $\mathsf{SessionKeyReveal}(P^{(i)})$. This query models misuses of session keys which are the intermediate result calculated from ephemeral keys. If the session key for the instance of participant $P^{(i)}$ is not defined, then return $\perp$. Otherwise, return the session key for $P^{(i)}$.
- $\mathsf{StaticKeyReveal}(P)$. This query models leakage of the static secrets of participant P. If P is a user, then return the password. If P is a gateway, then return secret keys for authenticated channels. If P is an authentication server, then return the encrypted passwords for all users, the share of a secret key, and other secret keys for authenticated and secure channels.
- $\mathsf{EphemeralKeyReveal}(P^{(i)})$. This query models leakage of the ephemeral keys used by instance $P^{(i)}$. The output of this query consists of the ephemeral keys of $P^{(i)}$ such as chosen random numbers.
- $\mathsf{EstablishParty}(U, pw_U)$. This query models that an adversary registers a password pw_U on behalf of a user U. The users against whom the adversary has not ask this query are called *honest*.

[1] This oracle halts if an adversary asks the oracle in the invalid order or the counter of incorrect login attempts exceeds the predetermined limit. The states of the oracle for an adversary in the same session are preserved. Although the SendUser and SendServer oracles interacts with an adversary acting as a gateway, the SendGateway oracle interacts with an adversary acting as a user and a combiner.

- $\mathsf{Test}(P^{(i)})$. This query models the indistinguishability of the session key of $P^{(i)}$. At the beginning of an experiment the challenge bit b is chosen. If the session key for $P^{(i)}$ is not defined, then return $\perp$. Otherwise, return session key of $P^{(i)}$ if $b = 1$ or a random key of the same size if $b = 0$. The adversary can ask this query only once at any time during the experiment.
- $\mathsf{TestPassword}(U, pw'_U)$. This query models secrecy of the password held by an honest user U. If the guessed password pw'_U equals the registered password pw_U, then return 1. Otherwise, return 0. The adversary can ask this query only once at any time during the experiment.

Added Oracles. We add a new oracle to adapt to the threshold setting where an adversary can obtain internal information of authentication servers by corruption.

- $\mathsf{Corrupt}(S_i)$. This query models intrusion into authentication servers. By asking the query at the beginning of the protocol, the adversary can take full control of an authentication server S_i.

3.4 Security Definitions

We describe the security definitions of GTPAKE. The definitions here are similar to GPAKE, but the method of dealing with authentication servers is different. The Session ID (SID) and Partner ID (PID) are used to define a partner sharing the session key in PAKE. The SID is an identifier to determine a session uniquely and the PID is an instance considered to share a session key.

Definition 3. (**Partnering** (Abdalla et al., 2005)). A user $U^{(i)}$ and a gateway $G^{(j)}$ are partnered if the following conditions hold.

1. $U^{(i)}$ and $G^{(j)}$ are accepted.
2. $U^{(i)}$ and $G^{(j)}$ have the same SID.
3. The PID of $U^{(i)}$ is $G^{(j)}$ and the PID of $G^{(j)}$ is $U^{(i)}$.
4. No other instances have the same PID of $U^{(i)}$ or $G^{(j)}$.

For the security of session keys, an adversary can ask the Test oracle once against a *fresh* participant. In the following definition, the adversary is restricted such that the adversary cannot ask queries that break the security of the protocol trivially.

Definition 4. (**Freshness in Session Key Security**). A user $U^{(i)}$ and a partnered gateway $G^{(j)}$ are *fresh* if the user is honest and none of the following conditions hold.

1. The adversary asks $\mathsf{SessionKeyReveal}(U^{(i)})$ or $\mathsf{SessionKeyReveal}(G^{(j)})$.
2. The adversary asks $\mathsf{EphemeralKeyReveal}(U^{(i)})$ or $\mathsf{EphemeralKeyReveal}(G^{(j)})$.

3. The adversary asks $\mathsf{SendServer}(C^{(k)}, m)$ and either queries.
   (a) $\mathsf{StaticKeyReveal}(G)$.
   (b) $\mathsf{StaticKeyReveal}(C)$.
4. The adversary asks $\mathsf{SendUser}(U^{(i)}, m)$ or $\mathsf{SendGateway}(G^{(j)}, m)$ and either queries.
   (a) $\mathsf{StaticKeyReveal}(U)$.
   (b) $\mathsf{EphemeralKeyReveal}(U^{(i)})$ in any instance i.
   (c) $\mathsf{Corrupt}(S_i)$ for not less than t authentication servers.

In the game to prove the security of session keys, an adversary is allowed to ask the Execute, SendUser, SendGateway, SendServer, SessionKeyReveal, StaticKeyReveal, EphemeralKeyReveal, Corrupt, EstablishParty, and Test oracles. The list of participants is given to an adversary at the beginning of the experiment. In this situation, we define $Succ^{\text{sks}}$ as the event that an adversary succeeds in guessing a challenge bit b in the Test oracle.

Capturing Security properties of session keys. As described in the condition 1 of Definition 4, KS is reflected by allowing an adversary to obtain session keys in the non-target session. As described in the condition 2, LIS is reflected by allowing an adversary to obtain internal information of some authentication servers and by prohibiting the adversary from obtaining ephemeral keys of users and gateways in the target session. As described in the condition 3, FS is reflected by allowing an adversary to obtain static keys of the users and by prohibiting the adversary from obtaining static keys of partnered gateways and the combiner. As described in the condition 4, BI is reflected by allowing an adversary to ask queries in the non-target session and by prohibiting the adversary from obtaining the password of the target user.

Definition 5. (**Session Key Security**). In a GTPAKE protocol $\mathcal{L}$, an advantage of an adversary $\mathcal{A}$ for the session key security is defined as

$$\mathsf{Adv}^{\text{sks}}_{\mathcal{L}}(\mathcal{A}) = |\Pr[Succ^{\text{sks}}] - 1/2|.$$

The session key security meets the equation $\mathsf{Adv}^{\text{sks}}_{\mathcal{L}}(\mathcal{A}) \le \mathsf{negl}(\kappa)$ for any probabilistic polynomial time adversaries where κ is a security parameter.

For the security of passwords, an adversary can ask the TestPassword oracle once against a *fresh* password.

Definition 6. (**Freshness in Password Protection Security**). A password of a user U is *fresh* if the user is honest and an adversary does not ask the following queries.

1. $\mathsf{StaticKeyReveal}(U)$.
2. $\mathsf{EphemeralKeyReveal}(U^{(i)})$ in any instance i.
3. $\mathsf{Corrupt}(S_i)$ for not less than t authentication servers.

In the game to prove the security of passwords, an adversary is allowed to ask the SendUser, SendServer, SessionKeyReveal, StaticKeyReveal, EphemeralKeyReveal, EstablishParty, Corrupt, and TestPassword oracles. The list of participants is given to an adversary at the beginning of the experiment. In this situation, we define $Succ^{\text{pps}}$ as the event that an adversary succeeds in guessing the password pw_U in the TestPassword oracle.

Capturing Security properties of passwords. As described in Definition 6, UDonDA is reflected by allowing an adversary to ask the SendUser and SendServer oracles until the number of incorrect login attempts does not exceed the predetermined limit. Also offDA is reflected by allowing an adversary to obtain internal information such as ephemeral or static keys of the non-target users and a set of less than the threshold corrupted authentication servers. The corrupted combiner can disturb the communication between a user and honest authentication servers, but cannot obtain information about the password of the user.

Definition 7. (**$(\mathcal{T}, \mathcal{R})$-Password Protection Security**). In a GTPAKE protocol $\mathcal{L}$, an advantage of an adversary $\mathcal{A}$ for the $(\mathcal{T}, \mathcal{R})$-password protection security is defined as

$$\mathsf{Adv}^{\text{pps}}_{\mathcal{L},\mathcal{D}}(\mathcal{A}) = \Pr[Succ^{\text{pps}}],$$

where the password is chosen at random from a dictionary $\mathcal{D}$ whose size is denoted as $|\mathcal{D}|$. A max advantage with at most a time $\mathcal{T}$ and a resource $\mathcal{R}$ is defined as

$$\mathsf{Adv}^{\text{pps}}_{\mathcal{L},\mathcal{D}}(\mathcal{T}, \mathcal{R}) = \max_{\mathcal{A}}\{\mathsf{Adv}^{\text{pps}}_{\mathcal{L},\mathcal{D}}(\mathcal{A})\}.$$

The $(\mathcal{T}, \mathcal{R})$-password protection security meets the equation $\mathsf{Adv}^{\text{pps}}_{\mathcal{L},\mathcal{D}}(\mathcal{T}, \mathcal{R}) \le (q_{send} + 1)/|\mathcal{D}| + \mathsf{negl}(\kappa)$ for any probabilistic polynomial time adversaries where q_{send} $(< |\mathcal{D}|)$ is the number of queries to the SendUser or SendServer oracles and κ is a security parameter.

4 OUR PROPOSED SCHEME

4.1 How to construct GTPAKE

We describe the problems and give an intuitive explanation of our construction. As described in Section 1.2, it is difficult to make GTPAKE of Abdalla et al. secure against UDonDA. In their scheme, it is impossible for an authentication server to terminate the protocol when an incorrect login attempt is made because the message made by an honest user is indistinguishable from that by the adversary. In addition, it seems difficult to convert GPAKE of Wei et al. (which is secure against UDonDA) into GTPAKE in a naive way. In the proposed scheme, it is possible for authentication servers to compute the result while keeping a password secret by realizing decryption and randomization simultaneously. We also use a zero knowledge proof in communication among authentication servers to prevent an adversary from showing incorrect shares.

4.2 Overview of our scheme

We describe the flow of our proposed scheme. First, a trusted dealer generates some public system parameters such as the public key pk of authentication servers. Second, a user registers the ElGamal encryption $(PW \cdot pk^v, g^v)$ with the hash value PW of his password pw and a random number v. Third, the user sends g^r/PW to authentication servers via a gateway where r is a random number. After a random number w is generated while hiding the random number among authentication servers, a combiner sends g^w to the user. The user and authentication servers verify the validity of $\mathsf{H}(g^{rw})$ with each other. The user sends g^x to the gateway where x is a random number. The gateway sends g^y to the user where y is a random number. Finally, a session key $\mathsf{H}(g^{xy})$ is established between the user and the gateway.

4.3 Construction

We show our proposed scheme based on the system model defined in Section 3.1. A perspective of the proposed scheme is shown in Figure 1. First, we describe the initialization process as below.

Init. Let p be a prime whose bit length is the given security parameter κ and q be a large prime dividing $p-1$. Let us denote a generator of subgroup $\mathbb{G}$ of order q over $\mathbb{Z}_p$ by g. The hash functions $H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_q$, $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$, and $H_2, H_3 : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ are chosen. A trusted dealer computes the parameters as follows: The trusted dealer computes another random generator h $(\neq g)$ and a public key $pk = g^s$ of a secret key $s \leftarrow \mathbb{Z}_q$ for the ElGamal encryption. The trusted dealer chooses $a_k \leftarrow \mathbb{Z}_q$ for $k = 1,\ldots,t-1$ where t is a given threshold value and generates a polynomial $f(z) = s + a_1 z + \cdots + a_{t-1} z^{t-1} \mod q$. The trusted dealer sends a share $s_i = f(i) \mod q$ to each authentication server S_i via a secure channel and publishes $g^{s_i} \mod p$ among the authentication servers. Finally, the public system parameters $params = (p, q, \mathbb{G}, g, h, pk, H_0, H_1, H_2, H_3)$ are outputted.

Second, we describe the registration process as below.

Regi. A new user chooses a password pw_U at random from a dictionary $\mathcal{D}$ and computes $PW_U = H_1(U \parallel pw_U)$ by using his identification U. After generating the ElGamal encryption $\mathsf{Enc}(pw_U) = (PW_U \cdot pk^v \mod p, g^v \mod p)$ where $v \leftarrow \mathbb{Z}_q$, the user sends $(U, \mathsf{Enc}(pw_U))$ to the authentication servers. This information is stored in all authentication servers as the encryption of the password for the user U. If any problems occur, then an error symbol $\perp$ is outputted.

We use the non-interactive zero-knowledge proof of equality of discrete logarithm as the building block in a similar manner to Abdalla's GTPAKE (Abdalla et al., 2005). We describe the proof system between a prover and a verifier as follows: The two generators (g_1, g_2) over a group $\mathbb{G}$ are given and let $\mathsf{EDLog}_{(g_1,g_2)}$ be the language pairs $(x_1, x_2) \in \mathbb{G}^2$ where there exists a random number $x \in \mathbb{Z}_q$ such that $x_1 = g_1^x$ and $x_2 = g_2^x$. The prover chooses $y \leftarrow \mathbb{Z}_q$ and computes $y_1 = g_1^y$ and $y_2 = g_2^y$. After computing $c = H_0(q \parallel g_1 \parallel g_2 \parallel x_1 \parallel x_2 \parallel y_1 \parallel y_2)$, the prover computes $z = xc + y \mod q$ and sends (c, z) to the verifier. The verifier checks the equation $c = H_0(q \parallel g_1 \parallel g_2 \parallel x_1 \parallel x_2 \parallel g_1^z/x_1^c \parallel g_2^z/x_2^c)$.

Third, we describe the authentication process composed of twelve steps as below.

Auth. If a processing request has come in the invalid order, the request is recorded as an incorrect login attempt and the protocol terminates. When the counter of incorrect login attempts exceeds the predetermined limit, a processing request from the gateway to a user or authentication servers is rejected.

Step 1. A user U computes $PW_U = H_1(U \parallel pw_U)$ by using his password pw_U. U chooses $r \leftarrow \mathbb{Z}_q$ and computes $R^* = g^r/PW_U$. U sends (U, R^*) to a gateway G.

Step 2. The gateway G sends (U, G, R^*) to a combiner C.

Step 3. The combiner C publishes (U, G, R^*) to all authentication servers. The authentication server S_i generates two polynomials $g_i(z) = b_{i,0} + b_{i,1} z + \cdots + b_{i,t-1} z^{t-1} \mod q$ and $g'_i(z) = b'_{i,0} + b'_{i,1} z + \cdots + b'_{i,t-1} z^{t-1} \mod q$ where $(b_{i,k}, b'_{i,k}) \leftarrow \mathbb{Z}_q^2$ for $k = 0,\ldots,t-1$. S_i broadcasts $B_{i,k} = g^{b_{i,k}} h^{b'_{i,k}} \mod p$ for $k = 0,\ldots,t-1$ and sends $w_{i,j} = g_i(j) \mod q, w'_{i,j} = g'_i(j) \mod q$ to S_j for $j = 1,\ldots,n$. S_j checks the equation $g^{w_{i,j}} h^{w'_{i,j}} = \prod_{k=0}^{t-1} (B_{i,k})^{j^k}$ for $i = 1,\ldots,n$. If the confirmation does not hold for index i, S_j publishes a complaint against S_i. An authentication server receiving not less than the threshold t complaints is marked as disqualified. S_i receiving a complaint from S_j publishes $w_{i,j}, w'_{i,j}$ satisfying the confirmation. An authentication server publishing the complaint with values satisfying the confirmation is also marked as disqualified. According to the index set L of not less than the threshold t non-disqualified authentication servers, S_i whose index is included in L computes as follows: S_i computes $w_i = \sum_{j \in L} w_{j,i}$ and broadcasts $A_{i,k} = g^{b_{i,k}}$ for $k = 0,\ldots,t-1$. S_j checks the equation $g^{w_{i,j}} = \prod_{k=0}^{t-1} (A_{i,k})^{j^k}$ for $i \in L$. If the confirmation does not hold for index i, S_j publishes a complaint. Otherwise S_i computes $W = \prod_{i \in L} A_{i,0}$. Using the encrypted password $\mathsf{Enc}(pw_U) = (E_1, E_2) = (PW_U \cdot pk^v, g^v)$, S_i broadcasts $F_{1,i} = (E_1 \cdot R^*)^{w_i}$, $F_{2,i} = E_2^{w_i}$ and proofs $\mathsf{EDLog}_{(E_1 \cdot R^*, g)}(F_{1,i}, \prod_{j \in L} \prod_{k=0}^{t-1} (A_{j,k})^{i^k})$, $\mathsf{EDLog}_{(E_2, g)}(F_{2,i}, \prod_{j \in L} \prod_{k=0}^{t-1} (A_{j,k})^{i^k})$. After S_i checks the proofs, C sends (U, C, W) to G.

Step 4. The gateway G sends (G, C, W) to U.

Step 5. The user U computes $K_1 = W^r$ and $\alpha = H_2(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$. U chooses $(x,d) \leftarrow \mathbb{Z}_q^2$ and computes $X = g^x$ and a commitment $Com = g^{\alpha}h^d$. U sends (U, X, Com) to G.
Step 6. The gateway G sends (U, Com) to C.
Step 7. The combiner C broadcasts (U, Com) among S_i whose index is included in the set L. S_i computes $F_1 = \prod_{i \in L} F_{1,i}^{\lambda_{L,0,i}}$ and $F_2 = \prod_{i \in L} F_{2,i}^{\lambda_{L,0,i}}$ where $\lambda_{L,i,j} = \prod_{\{k \in L \wedge k \neq j\}}(i-k)/(j-k)$ is a Lagrange coefficient. S_i broadcasts $T_{2,i} = F_2^{s_i}$ and a proof $\mathsf{EDLog}_{(F_2,g)}(T_{2,i}, g^{s_i})$. After checking the proof, S_i computes $K_1 = F_1 / \prod_{i \in L} T_{2,i}^{\lambda_{L,0,i}}$. S_i computes $\alpha' = H_2(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$. C sends (U, α') to G.
Step 8. The gateway G chooses $y \leftarrow \mathbb{Z}_q$ and computes $Y = g^y$. G computes $K_2 = X^y$ and an authenticator $Auth' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$. G sends $(Y, \alpha', Auth')$ to U.
Step 9. The user U checks the validity of α' by using α. U computes $K_2 = Y^x$ and checks the validity of $Auth'$ by using K_2. If one of the two confirmations is false, the user increments the counter of incorrect login attempts for G, $reject$ is outputted, and the protocol is terminated.[2] Otherwise, U computes a session key $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$ and sends (U, d) to G.
Step 10. The gateway G sends (U, d) to C.
Step 11. The combiner C broadcasts (U, d) among S_i whose index is included in the set L. S_i checks the validity of Com by using d and α'. If one of the confirmations is false, the authentication server increments the counter of incorrect login attempts for U, $reject$ is outputted, and the combiner sends $(U, failure)$ to G where $failure$ means that the authentication process failed. Otherwise, the combiner sends $(U, success)$ to G where $success$ means that the authentication process succeeded.
Step 12. The gateway G computes a session key $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$ and sk is outputted if $success$ is received. Otherwise (i.e., $failure$ is received), the session is rejected.

Remark. We explain the reason why the proposed scheme uses the commitment scheme in Step 5 although it seems unnecessary. If a user sends α with X in Step 5 without hiding α, a malicious gateway can guess the password of a target user with a combination of the on-line dictionary attack and the off-line dictionary attack as follows: First, the active adversary chooses the random number $\overline{w} \leftarrow \mathbb{Z}_q$ and sends $(G, C, g^{\overline{w}})$ to the user and obtains $\overline{\alpha}$. We note that this attack in the on-line manner is detected eventually by honest authentication servers. Second, the passive adversary chooses a password $\overline{PW}$ and checks whether $H_2(U \parallel G \parallel C \parallel R^* \parallel g^{\overline{w}} \parallel (R^* \cdot \overline{PW})^{\overline{w}})$ equals the hash value $\overline{\alpha}$ until the password satisfying this equation (that is the correct password) is detected.

[2] An honest user also counts the number of the login failures if the login attempt failed although a correct password was used, and will report that the gateway is corrupted if the counter exceeds the predetermined limit. This is because the corrupted gateway can cause such login failures.

5 SECURITY ANALYSIS

5.1 Session Key Security

We prove the security of the session key in the proposed scheme under the CDH assumption in the random oracle model.

Theorem 1. (**Session Key Security**). Let $\mathcal{L}$ be our GTPAKE protocol and $\mathcal{A}$ be a probabilistic polynomial time adversary that corrupts at most $t-1$ authentication servers in advance where $(t-1) < n/2$. Then the advantage of $\mathcal{A}$ for the session key security in $\mathcal{L}$ is

$$\mathsf{Adv}_{\mathcal{L}}^{\mathrm{sks}}(\mathcal{A}) \leq \frac{q_{H_0}^2 + q_{H_1}^2 + (q_{exe} + q_{send})^2}{2q} + \frac{q_{H_2}^2 + q_{H_3}^2}{2^{\kappa+1}}$$
$$+ q_{exe} \cdot (q_{H_2} + q_{H_3}) \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathrm{CDH}}(\kappa) + \frac{q_{H_2} + q_{H_3}}{q},$$

where $q_{H_0}, q_{H_1}, q_{H_2}, q_{H_3}$ are the numbers of hash queries to the oracles H_0, H_1, H_2, H_3, respectively, q_{exe} is the number of queries to the $\mathsf{Execute}$ oracle and q_{send} is the number of queries to the $\mathsf{SendUser}$, $\mathsf{SendGateway}$, and $\mathsf{SendServer}$ oracles.

Proof. We define $Succ^{\mathrm{sks}}$ in Game n as $Succ_n^{\mathrm{sks}}$.
Game 0. This experiment corresponds to a real attack by the adversary in the random oracle model. By Definition 5, and we have

$$\mathsf{Adv}_{\mathcal{L}}^{\mathrm{sks}}(\mathcal{A}) = |\Pr[Succ_0^{\mathrm{sks}}] - 1/2|.$$

Game 1. In this experiment, we simulate the hash functions and the oracle defined in Section 3.3. We simulate the random oracles H_0, H_1, H_2, and H_3 by maintaining hash lists Λ_0, Λ_1, Λ_2, and Λ_3 as follows:

- On a hash query $H_0(m)$, if there already exists a record (m, r), then we return r; Otherwise, we choose $r \leftarrow \mathbb{Z}_q$, add the record (m, r) in the hash list Λ_0, and return r;
- On a hash query $H_1(m)$, if there already exists a record (m, r), then we return r; Otherwise, we choose $r \leftarrow \mathbb{G}$, add the record (m, r) in the hash list Λ_1, and return r;
- On a hash query $H_2(m)$ (resp. $H_3(m)$), if there already exists a record (m, r), then we return r; Otherwise, we choose $r \leftarrow \{0,1\}^{\kappa}$, add the record (m, r) in the hash list Λ_2 (resp. Λ_3) and return r;

For the simulation in the later games, we also prepare the hash functions H_2' and H_3' by maintaining hash lists Λ_2' and Λ_3'.

The $\mathsf{Execute}$, $\mathsf{SendUser}$, $\mathsf{SendGateway}$, $\mathsf{SendServer}$, $\mathsf{SessionKeyReveal}$, $\mathsf{StaticKeyReveal}$, $\mathsf{EphemeralKeyReveal}$, $\mathsf{EstablishParty}$, $\mathsf{Corrupt}$, and Test oracles can be simulated as below.

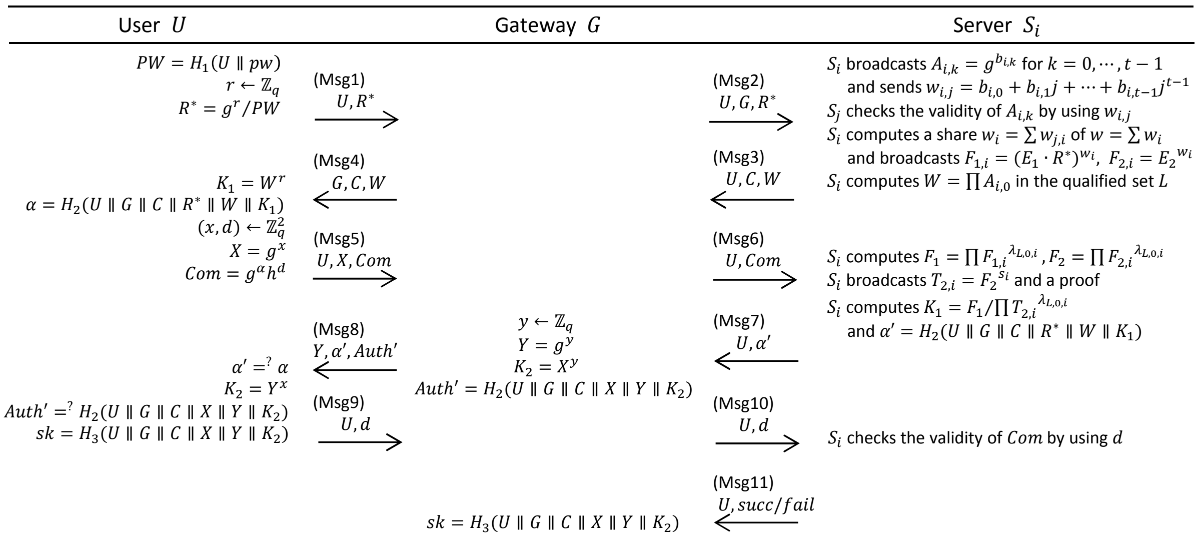
User U
Gateway G
Server S_i
$PW = H_1(U \parallel pw)$
$r \leftarrow \mathbb{Z}_q$
$R^* = g^r/PW$
(Msg1) U, R^*
(Msg2) U, G, R^*
S_i broadcasts $A_{i,k} = g^{b_{i,k}}$ for $k = 0, \cdots, t-1$
and sends $w_{i,j} = b_{i,0} + b_{i,1}j + \cdots + b_{i,t-1}j^{t-1}$
S_j checks the validity of $A_{i,k}$ by using $w_{i,j}$
S_i computes a share $w_i = \sum w_{j,i}$ of $w = \sum w_i$
and broadcasts $F_{1,i} = (E_1 \cdot R^*)^{w_i}$, $F_{2,i} = E_2{}^{w_i}$
S_i computes $W = \prod A_{i,0}$ in the qualified set L
(Msg3) U, C, W
(Msg4) G, C, W
$K_1 = W^r$
$\alpha = H_2(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$
$(x, d) \leftarrow \mathbb{Z}_q^2$
$X = g^x$
$Com = g^\alpha h^d$
(Msg5) U, X, Com
(Msg6) U, Com
S_i computes $F_1 = \prod F_{1,i}{}^{\lambda_{L,0,i}}$, $F_2 = \prod F_{2,i}{}^{\lambda_{L,0,i}}$
S_i broadcasts $T_{2,i} = F_2{}^{s_i}$ and a proof
S_i computes $K_1 = F_1/\prod T_{2,i}{}^{\lambda_{L,0,i}}$
and $\alpha' = H_2(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$
(Msg7) U, α'
$y \leftarrow \mathbb{Z}_q$
$Y = g^y$
$K_2 = X^y$
$Auth' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$
(Msg8) $Y, \alpha', Auth'$
$\alpha' =^? \alpha$
$K_2 = Y^x$
$Auth' =^? H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$
$sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$
(Msg9) U, d
(Msg10) U, d
S_i checks the validity of Com by using d
(Msg11) $U, succ/fail$
$sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$

Figure 1: Overview of the proposed scheme.

- On a query SendUser$(U^{(i)}, *)$, we proceed as follows:
  If a query StaticKeyReveal(U) or EphemeralKeyReveal$(U^{(i)})$ in any instance i or a query Corrupt(S_i) for not less than t authentication servers has been asked by the adversary or G is a member of invalid gateways for which the counter of incorrect login attempts exceeds the predetermined limit or a processing request has come in the invalid order, we increment the counter of incorrect login attempts, then do nothing.

  1. On a query SendUser$(U^{(i)}, \text{start})$, we proceed as follows: $PW_U = H_1(U \parallel pw_U)$; $r \leftarrow \mathbb{Z}_q$; $R^* = g^r/PW_U$; then return (U, R^*);
  2. On a query SendUser$(U^{(i)}, (G, C, W))$, we proceed as follows: $K_1 = W^r$; $\alpha = H_2(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$; $(x, d) \leftarrow \mathbb{Z}_q^2$; $X = g^x$; $Com = g^\alpha h^d$; then return (U, X, Com);
  3. On a query SendUser$(U^{(i)}, (Y, \alpha', Auth'))$, we proceed as follows: We check the validity of α'; $K_2 = Y^x$; We check the validity of $Auth'$; If one of the two confirmations is false, we increment the counter of incorrect login attempts for G and return *abort*; Otherwise, $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$; then return (U, d);

- On a query SendServer$(C^{(k)}, *)$, we proceed as follows:
  If a query StaticKeyReveal(G) or StaticKeyReveal(C) has been asked by the adversary or U is a member of invalid users for which the counter of incorrect login attempts exceeds the predetermined limit or a processing request has come in the invalid order, we increment the counter of incorrect login attempts, then do nothing. In the following queries, an adversary does the process on behalf of the corrupted authentication server S_j.

  1. On a query SendServer$(C^{(k)}, (U, G, R^*))$, we proceed as follows: The uncorrupted authentication server S_i broadcasts $B_{i,k} = g^{b_{i,k}} h^{b'_{i,k}}$ and sends $w_{i,j} = g_i(j), w'_{i,j} = g'_i(j)$ secretly to the authentication server S_j corrupted by an adversary; S_i checks the validity of $w_{j,i}, w'_{j,i}$ by using $B_{j,k}$; S_i computes a share $w_i = \sum_{j \in L} w_{j,i}$ and broadcasts $A_{i,k} = g^{b_{i,k}}$; S_i checks the validity of $A_{j,k}$ by using $w_{j,i}$; S_i broadcasts $F_{1,i} = (E_1 \cdot R^*)^{w_i}$, $F_{2,i} = E_2^{w_i}$ and proofs; S_i checks the proofs and computes $W = \prod_{j \in L} A_{i,0}$; then return (U, C, W);
  2. On a query SendServer$(C^{(k)}, (U, Com))$, we proceed as follows: The uncorrupted authentication server S_i computes $F_1 = \prod_{i \in L} F_{1,i}^{\lambda_{L,0,i}}$ and $F_2 = \prod_{i \in L} F_{2,i}^{\lambda_{L,0,i}}$; S_i broadcasts $T_{2,i} = F_2^{s_i}$ and a proof; S_i checks the proof and computes $K_1 = F_1 / \prod_{i \in L} T_{2,i}^{\lambda_{L,0,i}}$; S_i computes $\alpha' = H_2(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$; then return (U, α');
  3. On a query SendServer$(C^{(k)}, (U, d))$, we proceed as follows: The uncorrupted authentication server S_i checks the validity of Com; If one of the confirmations is false, we increment the counter of incorrect login attempts for U and return $(U, failure)$; Otherwise, return $(U, success)$;

- On a query SendGateway$(G^{(j)}, *)$, we proceed as follows:
  If a query StaticKeyReveal(U) or EphemeralKeyReveal$(U^{(i)})$ in any instance i or a query Corrupt(S_i) for not less than t authentication servers has been asked by the adversary or a processing request has come in the invalid order, then do nothing.

1. On a query SendGateway$(G^{(j)}, (U, R^*))$, then return (U, G, R^*);
2. On a query SendGateway$(G^{(j)}, (U, C, W))$, then return (G, C, W);
3. On a query SendGateway$(G^{(j)}, (U, X, Com))$, then return (U, Com);
4. On a query SendGateway$(G^{(j)}, (U, \alpha'))$, $y \leftarrow \mathbb{Z}_q$; $Y = g^y$; $K_2 = X^y$; $Auth' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$; then return $(Y, \alpha', Auth')$;
5. On a query SendGateway$(G^{(j)}, (U, d))$, then return (U, d);
6. On a query SendGateway$(G^{(j)}, (U, success/failure))$, If $success$ is received, $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$; If $failure$ is received, then do nothing;

- On a query Execute$(U^{(i)}, G^{(j)}, C^{(k)})$, we proceed as follows:
$(U, R^*) \leftarrow$ SendUser$(U^{(i)}, \text{start})$; $(U, G, R^*) \leftarrow$ SendGateway$(G^{(j)}, (U, R^*))$; $(U, C, W) \leftarrow$ SendServer$(C^{(k)}, (U, G, R))$; $(G, C, W) \leftarrow$ SendGateway$(G^{(j)}, (U, C, W))$; $(U, X, Com) \leftarrow$ SendUser$(U^{(i)}, (G, C, W))$; $(U, Com) \leftarrow$ SendGateway$(G^{(j)}, (U, X, Com))$; $(U, \alpha') \leftarrow$ SendServer$(C^{(k)}, (U, Com))$; $(Y, \alpha', Auth') \leftarrow$ SendGateway$(G^{(j)}, (U, \alpha'))$; $(U, d) \leftarrow$ SendUser$(U^{(i)}, (Y, \alpha', Auth'))$; $(U, d) \leftarrow$ SendGateway$(G^{(j)}, (U, d))$; $(U, success/failure) \leftarrow$ SendServer$(C^{(k)}, (U, d))$; SendGateway$(G^{(j)}, (U, success/failure))$; then return $(U, G, C, R^*, W, X, Y, \alpha, \alpha', Auth, Auth', Com, d, success/failure)$;
- On a query SessionKeyReveal$(P^{(i)})$, we proceed as follows:
If the session key sk is defined for the user or the gateway instance $P^{(i)}$ then return sk, else return $\perp$;
- On a query StaticKeyReveal(P), we proceed as follows:
If P is a user U, then return the registered password pw_U; If P is a gateway G, then return a secret key for authenticated channels between the gateway and authentication servers; If P is an authentication server S_i, then return the encrypted password Enc$(pw) = (PW \cdot pk^v, g^v)$ for all users, the share s_i of the secret key for P, the published parameter g^{s_i} for all authentication servers, and other secret keys for authenticated channels between the authentication server and a gateway and secure channels among authentication servers; else return $\perp$;
- On a query EphemeralKeyReveal$(P^{(i)})$, we proceed as follows:
If there are already the ephemeral keys generated by the instance $P^{(i)}$, then return the ephemeral keys, else return $\perp$;
- On a query Corrupt(S_i), we proceed as follows:
We return all the information obtained by the authentication server S_i such as static keys, ephemeral keys, and all the intermediate values of the computation in S_i.
- On a query EstablishParty(U, pw_U), we proceed as follows:
If there is already a user U, then do nothing, else establish a new user U with the password pw_U;
- On a query Test$(U^{(i)})$, we proceed as follows:
$sk \leftarrow$ SessionKeyReveal$(P^{(i)})$; if $sk = \perp$, then return $\perp$; else $b \leftarrow \{0,1\}$; if $b = 1$ then $sk' = sk$ else $sk' \leftarrow \{0,1\}^\kappa$; then return sk';

This experiment is perfectly indistinguishable from the previous experiment, and we have

$$\Pr[Succ_1^{\text{sks}}] = \Pr[Succ_0^{\text{sks}}].$$

Game 2. We halt this experiment when the collision on the outputs of the hash oracles and the transcripts occurs. We set the number of queries to the hash oracle H_i as q_{H_i} for $i = 0, 1, 2, 3$, that to the Execute oracle as q_{exe} and that to the SendUser, SendGateway, and SendServer oracles as q_{send}. By the birthday paradox, and we have

$$|\Pr[Succ_2^{\text{sks}}] - \Pr[Succ_1^{\text{sks}}]| \leq \frac{q_{H_0}^2 + q_{H_1}^2 + (q_{exe} + q_{send})^2}{2q} + \frac{q_{H_2}^2 + q_{H_3}^2}{2^{\kappa+1}}.$$

Game 3. We change the simulation of queries to the Execute oracle or queries sent by the oracle instances to SendUser and SendGateway oracles on the selected session. When a query SendGateway$(G^{(j)}, (U, \alpha'))$ or SendUser$(U^{(i)}, (Y, \alpha', Auth'))$ or SendGateway$(G^{(j)}, (U, success/failure))$ is asked, we compute the authenticators $Auth$ (and $Auth'$) and the session key sk as $H_2'(U \parallel G \parallel C \parallel X \parallel Y)$ and $H_3'(U \parallel G \parallel C \parallel X \parallel Y)$ by using the private hash oracles H_2' and H_3', respectively. The difference between this experiment and the previous one is indistinguishable unless the adversary asks the query $(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$ to the hash function H_2 or H_3. The difference of the following probabilities is negligible as long as the CDH assumption holds, and we have

$$|\Pr[Succ_3^{\text{sks}}] - \Pr[Succ_2^{\text{sks}}]| \leq q_{exe} \cdot (q_{H_2} + q_{H_3}) \cdot \mathsf{Adv}_{\mathcal{A}}^{\text{CDH}}(\kappa).$$

To evaluate the probability of this event, we construct an algorithm to solve the CDH problem. The algorithm obtains the CDH tuple (U, V) and chooses the session $(U^{(i)}, C^{(k)})$. We set $X = U^{u_1} g^{u_2}$ for a query SendUser$(U^{(i)}, (G, C, W))$ and $Y = V^{u_3} g^{u_4}$ for a query SendGateway$(G^{(j)}, (U, \alpha'))$ where $(u_1, u_2, u_3, u_4) \leftarrow \mathbb{Z}_q^4$. All other queries are handled in the same way as in Game 2. The simulator picks a selected session as test one with the probability $1/q_{exe}$. Although the simulator cannot obtain ephemeral keys of a user and a gateway by Definition 4, the simulator can simulate queries to all oracles without $\log_g X$ and $\log_g Y$.

If the adversary asks the query $(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$ to the hash function H_2 or H_3 in the session $(U^{(i)}, G^{(j)})$, we can compute $K_2 = \mathrm{CDH}(U^{u_1}g^{u_2}, V^{u_3}g^{u_4}) = \mathrm{CDH}(U^{u_1}, V^{u_3}) \cdot \mathrm{CDH}(U^{u_1}, g^{u_4}) \cdot \mathrm{CDH}(g^{u_2}, V^{u_3}) \cdot \mathrm{CDH}(g^{u_2}, g^{u_4}) = \mathrm{CDH}(U,V)^{u_1u_3} \cdot U^{u_1u_4} \cdot V^{u_2u_3} \cdot g^{u_2u_4}$ from hash lists Λ_2 and Λ_3 where CDH is a function to return g^{ab} from (g^a, g^b) in terms of the generator g in the same way as (Bresson et al., 2004). In this way, we can extract the value $\mathrm{CDH}(U,V)$ from the given tuple (U,V). As a result, the only way to distinguish a session key from a random key is to ask the query $(U \parallel G \parallel C \parallel X \parallel Y \parallel K_2)$ to the hash function H_2 or H_3. With the probability that the adversary succeeds in guessing the challenge bit b at random, we have

$$\Pr[Succ_3^{\text{sks}}] = \frac{1}{2} + \frac{q_{H_2} + q_{H_3}}{2^\kappa}.$$

$\square$

5.2 Password Protection Security

We prove the security of the password in the proposed scheme under the DDH assumption in the random oracle model.

Theorem 2. (**$(\mathcal{T}, \mathcal{R})$-Password Protection Security**). Let $\mathcal{L}$ be our GTPAKE protocol and $\mathcal{A}$ be a probabilistic polynomial time adversary that corrupts at most $t-1$ authentication servers in advance where $(t-1) < n/2$. Then the advantage of $\mathcal{A}$ for the password protection security in $\mathcal{L}$ with at most time $\mathcal{T}$ and resource $\mathcal{R}$ is

$$\mathsf{Adv}_{\mathcal{L}}^{\text{pps}}(\mathcal{A}) \leq \frac{q_{H_0}^2 + q_{H_1}^2 + (q_{exe} + q_{send})^2}{2q} + \frac{q_{H_2}^2 + q_{H_3}^2}{2^{\kappa+1}} + (q_{exe}+1) \cdot \mathsf{Adv}_{\mathcal{A}}^{\text{DDH}}(\kappa) + \frac{q_{sends} + q_{sendu} + 1}{|\mathcal{D}|},$$

where $q_{H_0}, q_{H_1}, q_{H_2}, q_{H_3}$ are the numbers of hash queries to the oracles H_0, H_1, H_2, H_3, respectively, q_{exe} is the number of queries to the Execute oracle, q_{send} is the number of queries to the SendUser, SendGateway, and SendServer oracles, q_{sends} and q_{sendu} are the numbers of queries to the SendServer and SendUser oracle, respectively, and $|\mathcal{D}|$ is the size of the dictionary $\mathcal{D}$.

Proof. We define $Succ^{\text{pps}}$ in Game n as $Succ_n^{\text{pps}}$.
Game 0. This experiment corresponds to a real attack by the adversary in the random oracle model. By Definition 7, and we have

$$\mathsf{Adv}_{\mathcal{L},\mathcal{D}}^{\text{pps}}(\mathcal{A}) = \Pr[Succ_0^{\text{pps}}].$$

Game 1. As in the proof of Section 5.1, we can simulate the hash functions H_i for $i = 0,1,2,3$, the Execute, SendUser, SendGateway, SendServer, SessionKeyReveal, StaticKeyReveal, EphemeralKeyReveal, EstablishParty, and Corrupt oracles. We simulate the TestPassword oracle as below.

- On a query $\mathsf{TestPassword}(U, pw'_U)$, we proceed as follows: $pw_U \leftarrow \mathsf{StaticKeyReveal}(U)$; If $pw'_U = pw_U$, then return 1, else return 0;

This experiment is perfectly indistinguishable from the previous experiment, and we have

$$\Pr[Succ_1^{\text{pps}}] = \Pr[Succ_0^{\text{pps}}].$$

Game 2. We halt this experiment when the collision on the outputs of the hash oracles and the transcripts occurs. By the birthday paradox, and we have

$$|\Pr[Succ_2^{\text{pps}}] - \Pr[Succ_1^{\text{pps}}]| \leq \frac{q_{H_0}^2 + q_{H_1}^2 + (q_{exe} + q_{send})^2}{2q} + \frac{q_{H_2}^2 + q_{H_3}^2}{2^{\kappa+1}}.$$

Game 3. We change the simulation of the queries to the SendServer oracle for all sessions and the Corrupt oracle for authentication servers. When a query $\mathsf{SendServer}(C^{(k)}, (U, Com))$ or $\mathsf{Corrupt}(S_i)$ is asked, we replace the secret part needed to decrypt the stored ElGamal encryption of the password with a random element for honest users. The difference between this experiment and the previous one is the stored passwords which are not generated by EstablishParty. The difference of the following probabilities is negligible as long as the DDH assumption holds, and we have

$$|\Pr[Succ_3^{\text{pps}}] - \Pr[Succ_2^{\text{pps}}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\text{DDH}}(\kappa).$$

Suppose a successful distinguisher between Games 2 and 3, and we construct an algorithm to solve the DDH problem to prove the above.

The simulator needs to change the process done by uncorrupted authentication servers to be consistent with the intended values. We assume w.l.o.g. that S_n is in the set of uncorrupted authentication servers participating in the authentication process. The simulator controls the other uncorrupted authentication servers as usual.

First, we deal with the simulation about the public key in the setup phase Init. In this proposed model, the trusted dealer computes a secret key s, the public key $pk = g^s$, and the corresponding share g^{s_i}. Given g^s without s, the simulator needs to publish g^{s_n} as a trusted dealer such that g^s is the ElGamal public key. Using the DDH triple (U,V,Z), the corresponding share for S_n is set as $g^{s_n} = U^{\lambda_{S,n,0}} \cdot \prod_{j \in (L \setminus \{n\})} (g^{s_j})^{\lambda_{S,n,j}}$ where $\lambda_{S,i,j} = \prod_{\{k | S_k \in S \wedge k \neq j\}} (i-k)/(j-k)$ is a Lagrange coefficient. Due to the honest majority setting, the simulator can compute all the share s_i of secret key s.

To simplify the system, we assume that the trusted dealer distributes the shares of the secret key corresponding to the ElGamal public key. We can construct the proposed scheme without the trusted dealer by producing some parameters among authentication servers. In this case, we can simulate the public key by hitting the intended value similar to the distributing key generation technique (Gennaro et al., 2007).

Second, we deal with the simulation about the stored passwords in the registration phase Regi. Since all users register the encrypted passwords by sending the encrypted passwords to the authentication servers, the simulator knows all the passwords for honest users. When a query $\mathsf{Corrupt}(S_i)$ is asked, we embed the DDH triple into all encrypted passwords as follows. The simulator returns $(PW_{U_i} \cdot Z^{v_i}, V^{v_i})$ where $v_i \leftarrow \mathbb{Z}_q$ for all honest users and $(PW_{U_i} \cdot pk^v, g^v)$ for the other users. Other internal information is given to the adversary as usual.

Third, we deal with the simulation about a partial decryption in the authentication phase Auth. We need to simulate the uncorrupted authentication servers for $\mathsf{SendServer}(C^{(k)}, (U, Com))$. The share $T_{2,n}$ for U_i can be computed as $Z^{w \cdot v_i \cdot \lambda_{L,n,0}} \cdot \prod_{j \in (L \backslash \{n\})} V^{w \cdot v_i \cdot s_j \cdot \lambda_{S,n,j}}$ without s_n. Other parameters generated from uncorrupted servers can be simulated similarly to the authentication process.

To detect malicious authentication servers that publish incorrect shares, the non-interactive zero-knowledge proof of equality of discrete logarithm is used in our scheme. In the random oracle model, the simulator can simulate this proof without knowing the secret keys as follows. The simulator chooses $(c, z) \leftarrow \mathbb{Z}_q^2$ and sends (c, z) as the proof. The hash oracle H_0 returns c from the hash list Λ_0 if the adversary asks the query $(q, g_1, g_2, u_1, u_2, g_1^z/u_1^c, g_2^z/u_2^c)$.

Therefore, the simulator assigns $(pk, g^v, \mathrm{CDH}(pk, g^v))$ for the given DDH triple (U, V, Z). In Definition 7, the task of an adversary is to guess the password of a target user. The simulator is able to solve the DDH problem by using the difference of success probability between each game because all the encrypted passwords for all honest users includes the DDH triple. In the case of $Z = \mathrm{CDH}(U, V)$, the environment for the distinguisher corresponds to Game 2. In the case of $Z \neq \mathrm{CDH}(U, V)$, the environment for the distinguisher corresponds to Game 3. If the distinguisher decides that the distinguisher interacted with Game 2, the algorithm outputs 1, otherwise 0.

Game 4. We change the simulation of queries to the $\mathsf{Execute}$ oracle or queries sent by the oracle instances to $\mathsf{SendUser}$ and $\mathsf{SendServer}$ oracles on the selected session. When a query $\mathsf{SendUser}(U^{(i)}, (G, C, W))$ or $\mathsf{SendServer}(C^{(k)}, (U, Com))$ is asked, we replace the Diffie-Hellman key in the authenticators α and α' with a random element. The difference of the following probabilities between this experiment and the previous one is negligible as long as the DDH assumption holds, and we have

$$|\Pr[Succ_4^{\mathrm{pps}}] - \Pr[Succ_3^{\mathrm{pps}}]| \leq q_{exe} \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathrm{DDH}}(\kappa).$$

To evaluate the probability of this event, we construct an algorithm to solve the DDH problem. The algorithm obtains the DDH triple (U, V, Z) and chooses the session $(U^{(i)}, C^{(k)})$. The distinguisher picks a selected session as test one with the probability $1/q_{exe}$. We want to compute $R = U^{u_1}$ for a query $\mathsf{SendUser}(U^{(i)}, \mathrm{start})$ and $W = V^{u_2}$ for a query $\mathsf{SendServer}(C^{(k)}, (U, G, R^*))$ where $(u_1, u_2) \leftarrow \mathbb{Z}_q^2$.

When a query $\mathsf{SendUser}(U^{(i)}, \mathrm{start})$ is asked, we compute $R^* = U^{u_1}/PW$ instead of $R^* = g^r/PW$. Accordingly, we set K_1 as $W^{u_1 u_2}$ on a query $\mathsf{SendUser}(U^{(i)}, (G, C, W))$.

When a query $\mathsf{SendServer}(C^{(k)}, (U, G, R^*))$ is asked, we describe the process of honest servers sending and receiving the information privately and publicly for corrupted parties without ephemeral keys $\log_g R$ and $\log_g W$ as below. The simulator knows all the shares $w_{i,j}, w'_{i,j}$, the coefficients $b_{i,k}, b'_{i,k}$, and the public values $B_{i,k}$ due to the honest majority setting. We assume w.l.o.g. that S_n is in the set of uncorrupted authentication servers in the well-defined set L. We compute $A_{i,k} = g^{b_{i,k}}$ for $i \in (L \backslash \{n\}), k = 0, \ldots, t-1$ and set $A^*_{n,0} = V^{u_2} \prod_{i \in (L \backslash \{n\})} (A_{i,0})^{-1}$. We assign $w^*_{n,j} = g_n(j)$ for $j \in (L \backslash \{n\})$ and compute $A^*_{n,k} = (A^*_{n,0})^{\lambda_{L,k,0}} \cdot \prod_{j \in (L \backslash \{n\})} (g^{w^*_{n,j}})^{\lambda_{L,k,j}}$ for $k = 1, \ldots, t-1$. We broadcast $A_{i,k}$ for uncorrupted authentication servers S_i and $A^*_{n,k}$ for $k = 0, \ldots, t-1$. We broadcast $F_{1,n} = (V^{u_2 sv} \cdot Z^{u_1 u_2})^{\lambda_{L,n,0}} \cdot \prod_{j \in (L \backslash \{n\})} (U^{u_1} \cdot pk^v)^{w_j \cdot \lambda_{L,n,j}}$ and $F_{2,n} = (V^{u_2 v})^{\lambda_{L,n,0}} \cdot \prod_{j \in (L \backslash \{n\})} (E_2)^{w_j \cdot \lambda_{L,n,j}}$. All other processes are handled in the same way as in the previous game.

Since the simulation in the generation of $\log_g W$ is identical to the action in the previous game, the set L can be the same as the real protocol at the end of the protocol. Accordingly, other parameters such as $w'_{i,j}$ are defined without contradiction. More details can be found in (Gennaro et al., 2007). We note that the password obtained via the $\mathsf{Execute}$ oracle is information-theoretically hidden in sessions because R, X, and K_1 are relatively independent in every session. On the other hand, the passwords obtained via the $\mathsf{SendUser}$ and $\mathsf{SendServer}$ oracles are still used in sessions.

Therefore, the simulator assigns $(R, W, \mathrm{CDH}(R, W))$ for the triple $(U^{u_1}, V^{u_2}, Z^{u_1 u_2})$. In the case of $Z = \mathrm{CDH}(U, V)$, the environment for the distinguisher corresponds to Game 3. In the case of $Z \neq \mathrm{CDH}(U, V)$, the environment for the distinguisher corresponds to Game 4. If the distinguisher decides that the distinguisher interacted with Game 3, the algorithm outputs 1, otherwise 0.

Game 5. We change the simulation of the queries to the $\mathsf{SendServer}$ oracle. When a query $\mathsf{SendServer}(C^{(k)}, (U, Com))$ is asked, we compute the authenticator α' as $H'_2(U \parallel G \parallel C \parallel R^* \parallel W)$ by using the private hash oracle H'_2. The difference between this experiment and the previous one is indistinguishable unless the adversary asks a query $(U \parallel G \parallel C \parallel R^* \parallel W \parallel K_1)$ to the hash function H_2 where $K_1 = \mathrm{CDH}(R^* \cdot PW, W)$.

There is at most one password such that $K_1 = \mathrm{CDH}(R^* \cdot PW, W)$ for some pair (R^*, W) as a result of Lemma 2 in (Bresson et al., 2004). Therefore, the probability that this bad event occurs in the non-concurrent setting is bounded by the probability that an adversary guesses the password at random through the $\mathsf{SendServer}$ oracle to which q_{sends} is the number of queries with at most time $\mathcal{T}$ and resource $\mathcal{R}$ where $|\mathcal{D}|$ is the size of the dictionary $\mathcal{D}$, we have

$$|\Pr[Succ_5^{\mathrm{pps}}] - \Pr[Succ_4^{\mathrm{pps}}]| \leq \frac{q_{sends}}{|\mathcal{D}|}.$$

Game 6. We change the simulation of the queries to the SendUser oracle. When SendUser$(U^{(i)}, \text{start})$ is asked, we compute R^* as g^{r^*} where $r^* \leftarrow \mathbb{Z}_q$ without using the password pw_U. R^* has been simulated, but W has been generated by an adversary who tries to impersonate a combiner to a user. To succeed in passing the verification done by the user, the adversary needs to send an authenticator α which is computed with at most one password in the non-concurrent setting. Then the success probability with at most time $\mathcal{T}$ and resource $\mathcal{R}$ is at most $q_{sendu}/|\mathcal{D}|$ where q_{sendu} is the number of queries to the SendUser oracle, we have

$$|\Pr[Succ_6^{\mathrm{pps}}] - \Pr[Succ_5^{\mathrm{pps}}]| \leq \frac{q_{sendu}}{|\mathcal{D}|}.$$

We note that if a user sends the hash value α without using the commitment *Com*, the adversary can compute the password of the honest user as described in Section 4.3. This means the proof cannot go through if we do not use the commitment *Com* due to the existence of such an adversary distinguishing between the previous game and this game. In the proposed scheme, however, α is information-theoretically hidden by the commitment, so the difference between this experiment and the previous one is indistinguishable. We note also that the user reveals α to the authentication servers in Step 9 eventually only when the authentication servers already sent the same α' to the user, so this does not cause any harm in terms of security.

In this game, the information of the password obtained via the SendUser and SendServer oracles is not used in sessions, so the malicious gateway cannot do much better than guessing the password at random. Finally, the adversary guesses the password through the TestPassword oracle once, we have

$$\Pr[Succ_6^{\mathrm{pps}}] = \frac{1}{|\mathcal{D}|}.$$

□

6 CONCLUDING REMARKS

We proposed new GTPAKE which has resistance of UDonDA and the corruption of authentication servers. We proved the security of our GTPAKE under standard assumptions in the random oracle model. The proposed scheme has the stronger security against a malicious provider compared with existing schemes, and a global roaming service used for users regardless of places and devices is expected as an application.

REFERENCES

Abdalla, M., Chevassut, O., Fouque, P.A., Pointcheval, D.: A Simple Threshold Authenticated Key Exchange from Short Secrets. ASIACRYPT 2005, LNCS vol.3788, pp.566-584 (2005)

Abdalla, M., Izabachene, M., Pointcheval, D.: Anonymous and Transparent Gateway-Based Password-Authenticated Key Exchange. CANS 2008, LNCS vol.5339, pp.133-148 (2008)

Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. EUROCRYPT 2000, LNCS vol.1807, pp.139-155, (2000)

Bellovin, S., Merritt, M.: Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. ACM CCS 1993, pp.244-250, (1993)

Bellovin, S., Merritt, M.: Encrypted Key Exchange: Password based protocols secure against dictionary attacks. Proceedings of the IEEE Symposium on Research in Security and Privacy, pp.72-84, (1992)

Boyko, V., MacKenzie, P., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. EUROCRYPT 2000, LNCS vol.1807, pp.156-171, (2000)

Bresson, E., Chevassut, O., Pointcheval, D.: New Security Results on Encrypted Key Exchange. PKC 2004, LNCS vol.2947, pp.145-158 (2004)

Byun, J.W., Lee, D.H., Lim, J.I.: Security analysis and improvement of a gateway-oriented password-based authenticated key exchange protocol. IEEE Communications Letters vol.10 no.9, pp.683-685 (2006)

Ding, Y., Horster, P.: Undetectable On-line Password Guessing Attacks. Operating Systems Review vol.29 no.4, pp.77-86 (1995)

European Network and Information Security Agency.: Cloud computing risk assessment. (2009)

European Network and Information Security Agency.: Heartbleed Wake Up Call. (2014)

Gennaro, R., Jarecki, S.,Krawczyk, H., Rabin, T.: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. J. Cryptology vol.20 no.1, pp.51-83 (2007)

Goldreich, O., Lindell, Y.: Session-Key Generation Using Human Passwords Only. CRYPTO 2001, LNCS vol.2139, pp.408-432, (2001)

Katz, J., Ostrovsky, R., Yung, M.: Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. EUROCRYPT 2001, LNCS vol.2045, pp.475-494, (2001)

Szydlo, M.: A Note on Chosen-Basis Decisional Diffie-Hellman Assumptions. FC 2006, LNCS vol.4107, pp.166-170 (2006)

Wei, F., Ma, C., Zhang, Z.: Gateway-Oriented Password-Authenticated Key Exchange Protocol with Stronger Security. ProvSec 2011, LNCS vol.6980, pp.366-379 (2011)

Wei, F., Zhang, Z., Ma, C.: Analysis and Enhancement of an Optimized Gateway-Oriented Password-Based Authenticated Key Exchange Protocol. IEICE Transactions vol.96-A no.9, pp.1864-1871 (2013)

Wei, F., Zhang, Z., Ma, C.: Gateway-oriented password-authenticated key exchange protocol in the standard model. Journal of Systems and Software vol.85 no.3, pp.760-768 (2012)