PAPER

Provably Secure Gateway Threshold Password-Based Authenticated Key Exchange Secure against Undetectable On-Line Dictionary Attack**

Yukou KOBAYASHI[†*], *Nonmember*, Naoto YANAI[††], Kazuki YONEYAMA[†††], Takashi NISHIDE[†a)], *Members*, Goichiro HANAOKA[††††], *Nonmember*, Kwangjo KIM[†††††], *Member*, *and* Eiji OKAMOTO[†], *Fellow*

SUMMARY By using Password-based Authenticated Key Exchange (PAKE), a server can authenticate a user who has only the same password shared with the server in advance and establish a session key with the user simultaneously. However, in the real applications, we may have a situation where a user needs to share a session key with server A, but the authentication needs to be done by a different server B that shares the password with the user. Further, to achieve higher security on the server side, it may be required to make PAKE tolerant of a server breach by having multiple authentication servers. To deal with such a situation, Abdalla et al. proposed a variant of PAKE called Gateway Threshold PAKE (GTPAKE) where a gateway corresponds to the aforementioned server A being an online service provider and also a potential adversary that may try to guess the passwords. However, the schemes of Abdalla et al. turned out to be vulnerable to Undetectable On-line Dictionary Attack (UDonDA). In this paper, we propose the first GTPAKE provably secure against UDonDA, and in the security analysis, we prove that our GTPAKE is secure even if an adversary breaks into parts of multiple authentication servers.

key words: *password-based authenticated key exchange (PAKE), threshold cryptography*

1. Introduction

1.1 Background

Password-based authenticated key exchange (PAKE) [3]–[6], [13], [15] is a two-party protocol by which a user and an authentication server can establish a cryptographic key that is computationally hard to guess by sharing only the same human-memorable password in advance. PAKE can be useful because it does not assume a public-key infrastructure that requires the user to bring a device storing public-key related data. PAKE will also fit in well with computational environments where simple and secure ubiquitous access is needed.

Although PAKE is useful, there exist situations where (two-party) PAKE cannot be deployed as is. For instance, in the case of a global roaming service, a user (who is located abroad) may receive the service from an external server (called a gateway which can be considered as a service provider) by establishing a session key with the server, but in PAKE, it is assumed that the authentication server is the service provider at the same time. Hence, traditional PAKE will not function in such a context because the gateway is different from the authentication server. Furthermore, as the European Network and Information Security Agency (ENISA) [10] pointed out the potential threat of malicious servers, it is desirable to take into consideration the existence of malicious authentication servers to achieve high security. However, traditional PAKE will not be sufficient because only a single authentication server is supposed to be involved. Even if the server is trusted, software vulnerabilities can lead to a leakage of important data as shown by Heartbleed in April 2014 [11] where the vulnerability of OpenSSL can cause a leakage of the passwords stored in servers. Therefore, traditional PAKE cannot cope with such a threat.

To tackle those problems, Abdalla et al. proposed Gateway PAKE (GPAKE) [1] by introducing an entity called a gateway into PAKE and Gateway Threshold PAKE (GTPAKE) [1] by further having multiple authentication servers. However, as shown in [8], these schemes of Abdalla et al. turn out to be vulnerable to Undetectable On-line Dictionary Attack (UDonDA) in which an adversary mounts a password guessing attack in the on-line transaction and the attack cannot be detected by the authentication servers [9]. Although, in [2], it is mentioned that these protocols can be modified such that the authentication server can detect on-line dictionary attacks, the details of how it can be done and the security proof were not described. In the schemes of Abdalla et al., the authentication server responds and sends a message without authenticating the user, so the adversary can make on-line attempts to guess the password repeatedly. Due to the low entropy of the password, that attack becomes devastating. Hence, it is needed to propose a new GTPAKE scheme that overcomes UDonDA.

Manuscript received August 31, 2016.
Manuscript revised August 20, 2017.

[†] The authors are with University of Tsukuba, Tsukuba-shi, 305-8577 Japan.
[††] The author is with Osaka University, Suita-shi, 565-0871 Japan.
[†††] The author is with Ibaraki University, Hitachi-shi, 316-8511 Japan.
[††††] The author is with National Institute of Advanced Industrial Science and Technology, Tsukuba-shi, 305-8560 Japan.
[†††††] The author is with Korea Advanced Institute of Science and Technology, Korea.
[*] Presently, with LAC Co., Ltd.

[**] A preliminary version of this paper appeared in [16] and the detailed security analysis about the necessity of the commitments used in the proposed protocol was added.

a) E-mail: nishide@risk.tsukuba.ac.jp

DOI: 10.1587/transfun.E100.A.2991

Table 1 Comparison of the existing GPAKE and GTPAKE schemes.

Protocol	User	Gateway	Server	Message	Assumption	Model	Threshold	UDonDA
GPAKE [1]	$2e$	$2e$	$2e$	4	PCDDH	ROM	NO	NO
GTPAKE [1]	$2e$	$2e$	$(13n + 18)e$	$3n + 4$	PCDDH	ROM	YES	NO
GPAKE [18]	$3e$	$2e$	$2e$	6	DDH	ROM	NO	YES
GPAKE [20]	$5e + E$	$2e$	$5e + E$	6	DDH	Standard	NO	YES
GPAKE [19]	$3e$	$2e$	$2e$	9	CDH	ROM	NO	YES
Our GTPAKE	$6e$	$2e$	$(2n^2 + 19n + 11)e$	$4n + 11$	DDH	ROM	YES	YES

The computational costs for the user, gateway, and authentication server are estimated in the User, Gateway, and Server columns, respectively. Especially, the computational costs of [20] here are the ones recalculated by us. We use modular exponentiation denoted as "e" because modular exponentiation is the most expensive computation and "E" means the 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, the 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 the protocol tolerates the corruption of authentication servers. In the UDonDA column, it is shown whether the protocol can detect malicious login attempts for guessing the passwords of users.

1.2 Contribution

We propose new GTPAKE which has resistance of UDonDA and the corruption of authentication servers. We adapt the security model of GPAKE [1], [18] to that of GTPAKE and prove the security of our GTPAKE under standard assumptions in the random oracle model. The proposed scheme has the stronger security against a malicious gateway or authentication servers compared with existing schemes, and a global roaming service used for users regardless of places and devices is expected to be one of its applications.

A naive extension of GPAKE where the communication process is not changed and the process of authentication servers is adapted to the threshold secret sharing does not lead to GTPAKE. Even if the password is encrypted and the secret key for encrypting the password is shared among the authentication servers, the password itself needs to be decrypted to compare the registered password with the login password, which means the leakage of password to an adversary corrupting authentication servers. To overcome this problem, 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 [1]. However, Szydlo shows that the Chosen-basis Decisional Diffie-Hellman (CDDH) assumption on which the schemes of Abdalla et al. are based is already vulnerable to some attacks [17]. The security of our scheme is proven in the random oracle model with the DDH assumption. Similarly to GTPAKE of Abdalla et al., our scheme tolerates the corruption of some authentication servers. Furthermore, while the schemes of Abdalla et al. are vulnerable to UDonDA, our scheme is invulnerable to this attack, although our proof similar to [18] 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 Sect. 2, we introduce some background to understand this paper. In Sect. 3, we define the security model of GTPAKE. In Sect. 4, we describe the construction of our scheme. In Sect. 5, we prove the security of the proposed scheme. In Sect. 6, we make final remarks. Finally, we explain the reason why the proposed scheme needs a commitment scheme in Appendix A and the bound related to the birthday problem in Appendix B.

2. Preliminaries

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

2.1 Notation

We use the following notation throughout this paper. We denote by $\mathbb{Z}_q$ the set $\{0, 1, \ldots, q - 1\}$. $x \leftarrow A$ represents that x is chosen uniformly at random from set A. Let g be a generator of the subgroup $\mathbb{G}$ of order p over $\mathbb{Z}_q$ and $a \parallel b$ be the concatenation of elements a and b, which is able to be divided into its original elements. We denote by $\{0, 1\}^k$ the set of all binary strings of length k. Especially, $\{0, 1\}^*$ means the set of all binary strings of arbitrary length. The 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 sets of all users, gateways, and authentication servers, respectively, and n is the number of all authentication servers. 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 $C \in \mathcal{S}$. We set t $(< n/2 + 1)$ as the threshold value which is the minimum number of servers required to authenticate users.

[†] In the comparison here, we focus only on schemes with security proofs, and the discussion of schemes without security proofs can be found in [18].

2.2 Security Assumptions

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 that 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 the 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 that 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 the security parameter κ.

2.3 Building Block

We use the non-interactive zero-knowledge proof of equality of discrete logarithm as the building block in a similar manner to GTPAKE of Abdalla et al. [1]. We describe the proof system between a prover and a verifier as follows.
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 $c = H_0(q \parallel g_1 \parallel g_2 \parallel x_1 \parallel x_2 \parallel y_1 \parallel y_2)$ where H_0 is a hash function mapping $\{0, 1\}^* \rightarrow \mathbb{Z}_q$, the prover computes $z = xc + y \bmod q$ and sends (c, z) to the verifier. The verifier checks the equation $c = H_0(q \parallel g_1 \parallel g_2 \parallel x_1 \parallel x_2 \parallel g_1^z/x_1^c \parallel g_2^z/x_2^c)$.

3. Security Model

We describe the system model and security definitions of GTPAKE.

As Fig. 1 shows the communication model of GTPAKE, the user as the client connects to a gateway playing the role of the service provider. The gateway has a role of forwarding the message to one of the authentication servers named the combiner. Here the authentication servers play the role of authentication service provider. The combiner communicates with other servers to authenticate the user. Although the communication channel between the user and the gateway is insecure and under the control of an adversary, the channel between the gateway and the combiner is authenticated and the channel between the authentication servers is secure[†].

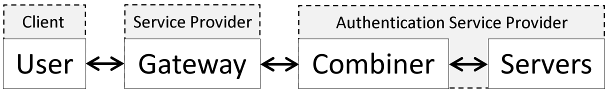

Fig. 1 Communication model of GTPAKE.

When an authentication 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 subprotocols.

- **Init**. Given the security parameter and the setup parameters, the public parameters *params* are output[††].
- **Regi**. The user registers his password with the authentication servers. If all authentication servers cannot register the password successfully, then outputs the error symbol $\perp$.
- **Auth**. The more than or equal to the threshold servers authenticate a user. If the user and gateway can establish the same session key while passing 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.

3.2.1 Existing Security Requirements

The security requirements for not only GPAKE but also GTPAKE are as follows (due to Wei et al. [18]).

- **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.

[†] To establish an authenticated or secure channel, both parties (i.e., the gateway and combiner, the combiner and authentication server, and pairs of authentication servers) have a common static key.

[††] Although we assume a trusted dealer distributing some parameters for simplicity, the authentication servers themselves can publish parameters by using the technique of the distributed key generation [12].

- **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 cannot guess a password by verifying its guess in the off-line manner.
- **Resistance to Undetectable On-line Dictionary Attack (UDonDA)**. The adversary cannot guess a password by verifying its guess in the on-line manner without being detected by uncorrupted participants.

3.2.2 Modified Security Requirements

We adapt the security requirements of GPAKE to that for the threshold setting. Unlike the single server setting where the adversary does not corrupt the authentication server in GPAKE, we take an active adversary corrupting less than threshold authentication servers into consideration in GTPAKE.

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

3.3 Oracles

We show the necessary oracles to define a stronger security model than that of GTPAKE [1]. To distinguish the session between participants, the ℓ-th instance for participant P is denoted by $P^{(\ell)}$. Let $U^{(\ell)}$, $G^{(j)}$, $C^{(k)}$, and $S_i^{(k)}$ be instances of a user, gateway, combiner, and i-th authentication server, respectively. The instance represents the state of a participant in a session during the progress of the protocol.

3.3.1 Existing Oracles [18]

The oracles used in the existing GPAKE are as follows.

- $\mathsf{Execute}(U^{(\ell)}, G^{(j)}, C^{(k)})$. This query models passive attacks. The output of this query consists of the message exchanged during the execution in the protocol among $U^{(\ell)}$, $G^{(j)}$, and $C^{(k)}$.
- $\mathsf{SendUser}(U^{(\ell)}, m)$. This query models active attacks against a user instance $U^{(\ell)\dagger}$. The output of this query consists of the message the user instance $U^{(\ell)}$ would generate on receipt of message m.
- $\mathsf{SendGateway}(G^{(j)}, m)$. This query models active attacks against a gateway instance $G^{(j)}$.† 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 $C^{(k)}$.† The output of this query consists of the message the combiner instance $C^{(k)}$ would generate on receipt of message m.
- $\mathsf{SessionKeyReveal}(P^{(\ell)})$. This query models leakage of session keys related to information of passwords. If the session key for the instance of participant $P^{(\ell)}$ is not defined, then return $\perp$. Otherwise, return the session key for $P^{(\ell)}$.
- $\mathsf{StaticKeyReveal}(P)$. 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, the published parameters for all authentication servers, and other secret keys for authenticated and secure channels.
- $\mathsf{EphemeralKeyReveal}(P^{(\ell)})$. This query models leakage of the ephemeral keys used by instance $P^{(\ell)}$. The output of this query consists of the ephemeral keys of $P^{(\ell)}$ such as the 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*.
- $\mathsf{Test}(P^{(\ell)})$. This query models the indistinguishability of the session key of $P^{(\ell)}$. At the beginning of the experiment the challenge bit b is chosen from $\{0, 1\}$. If the session key for $P^{(\ell)}$ is not defined, then return $\perp$. Otherwise, return session key of $P^{(\ell)}$ 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 the secrecy of the password held by an honest user U. If the guessed password pw'_U equals the registered password pw_U of the user U, then return 1. Otherwise, return 0. The adversary can ask this query only once at any time during the experiment.

3.3.2 Added Oracles

We add a new oracle to adapt the single server setting 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 the authentication server. By asking the query at the beginning of the protocol, the adversary can take full control of the authentication server S_i.

3.4 Security Definitions

We describe the security definitions of GTPAKE. The defi-

[†] The states of the oracle for an adversary in the same session are preserved. The SendUser and SendServer oracles halt if an adversary asks the oracle in the invalid order or the counter of incorrect login attempts exceeds a predetermined limit which is the upper limit of acceptable failed login attempts defined by the authentication service provider. The SendGateway oracle halts if an adversary asks the oracle in the invalid order. Although the SendUser and SendServer oracles interact with an adversary acting as a gateway, the SendGateway oracle interacts with an adversary acting as a user and a combiner.

nitions here are similar to those of GPAKE [1], [18], but the method of dealing with authentication servers is different. The Session ID (SID) is provided when the protocol is first initiated to determine the session uniquely. The Partner ID (PID) is taken to be an identifier of the instance intended to establish a session key.

Definition 3: (**Partnering** [1]). A user $U^{(\ell)}$ and gateway $G^{(j)}$ are partnered if the following conditions hold.

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

3.4.1 Session Key Security

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 $U^{(\ell)}$ and partnered gateway $G^{(j)}$ are *fresh* if the user is honest and none of the following conditions hold.

1. The adversary asks SessionKeyReveal($U^{(\ell)}$) or SessionKeyReveal($G^{(j)}$).
2. The adversary asks EphemeralKeyReveal($U^{(\ell)}$) or EphemeralKeyReveal($G^{(j)}$).
3. The adversary asks SendServer($C^{(k)}, m$) and either queries.
   a. StaticKeyReveal(G).
   b. StaticKeyReveal(C).
4. The adversary asks SendUser($U^{(\ell)}, m$) or SendGateway($G^{(j)}, m$) and either queries.
   a. StaticKeyReveal(U).
   b. EphemeralKeyReveal($U^{(\ell)}$) in any instance ℓ.
   c. Corrupt(S_i) for more than or equal to t authentication servers.

To model the attacks which the adversary can mount through the game in 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 the adversary at the beginning of the experiment. In this situation, we define $Succ^{\mathrm{sks}}$ as the event where an adversary succeeds in guessing a challenge bit b in the Test oracle.

(1) Capturing the 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 non-target sessions. As described in the condition 2, LIS is reflected by allowing an adversary to obtain the ephemeral and static keys or intermediate results calculated by some authentication servers and by prohibiting the adversary from obtaining the 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 the 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 non-target sessions and by prohibiting the adversary from obtaining the password of the target user.

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

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

where the password is chosen at random from dictionary $\mathcal{D}$ of size $|\mathcal{D}|$. The maximum advantage among all adversaries that expend at most $\mathcal{T}$ time and $\mathcal{R}$ resources is defined as

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

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

3.4.2 Password Protection Security

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

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

1. StaticKeyReveal(U).
2. EphemeralKeyReveal($U^{(\ell)}$) in any instance ℓ.
3. Corrupt(S_i) for more than or equal to t authentication servers.

To model the attacks which the adversary can mount through the game in 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 the adversary at the beginning of the experiment. In this situation, we define $Succ^{\mathrm{pps}}$ as the event that an adversary succeeds in guessing the password pw_U in the TestPassword oracle.

(1) Capturing the 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 exceeds a predetermined limit. Also offDA is reflected by allowing an adversary to obtain internal information such as the ephemeral and static keys of 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 successful disturbance is not considered as the success of breaking the security of passwords.

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

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

where the password is chosen at random from dictionary $\mathcal{D}$ of size $|\mathcal{D}|$. The maximum advantage among all adversaries that expend at most $\mathcal{T}$ time and $\mathcal{R}$ resources is defined as

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

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

4. Our Scheme

4.1 How to Construct GTPAKE

We describe the problems and give an intuitive explanation of our construction. As described in Sect. 1.2, it is difficult to convert GPAKE of Wei et al. (which is secure against UDonDA) into GTPAKE. In fact, the naive extension of GPAKE of Wei et al. is the insecure scheme described in Appendix A. In addition, it seems difficult to make GTPAKE of Abdalla et al. secure against UDonDA. In the schemes of Abdalla et al., 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 the proposed scheme, it is possible for authentication servers to compute the result while keeping the password itself secret by realizing decryption and randomization simultaneously. We also use zero knowledge proofs 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 the authentication servers. Second, a user registers the ElGamal ciphertext $(PW \cdot pk^v, g^v)$ with the hash value PW of his password pw and a random number v. Third, the user sends g^r/PW to the authentication servers via a gateway where r is a random number. After a random number w is generated while hiding the random number among the authentication servers, the combiner sends g^w to the user. The user and authentication servers verify the validity of $H(g^{rw})$ with each other where H is a hash function. 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, the session key $H(g^{xy})$ is established between the user and the gateway.

4.3 Construction

We show our proposed scheme based on the system model defined in Sect. 3.1. A perspective of the proposed scheme is shown in Fig. 2.

First, we describe the following sub-protocol of initialization. If the number of authentication servers is modified, this sub-protocol is executed again.
Init. Let p be a κ-bit prime where κ is the given security parameter 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 $H_0 : \{0,1\}^* \to \mathbb{Z}_q$, $H_1 : \{0,1\}^* \to \mathbb{G}$, and $H_2, H_3 : \{0,1\}^* \to \{0,1\}^\kappa$ are chosen.
A trusted dealer computes the parameters as follows.
The trusted dealer computes another random generator h ($\neq g$) and the 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 the given threshold value and generates the polynomial $f(z) = s + a_1 z + \cdots + a_{t-1} z^{t-1} \bmod q$.
The trusted dealer sends a share $s_i = f(i) \bmod q$ to each authentication server S_i via a secure channel and publishes $g^{s_i} \bmod p$ 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 output.

Second, we describe the following sub-protocol of registration.
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 ciphertext $\mathsf{Enc}(pw_U) = (PW_U \cdot pk^v \bmod p, g^v \bmod p)$ 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 ciphertext of the password for the user U.
If any problems occur, then the error symbol $\perp$ is output.

Third, we describe the following sub-protocol of authentication composed of twelve steps.
Auth. If a processing request has come in the invalid order, the incorrect login attempt is counted and the session is rejected. After the counter of incorrect login attempts exceeds the predetermined limit, a processing request from the user

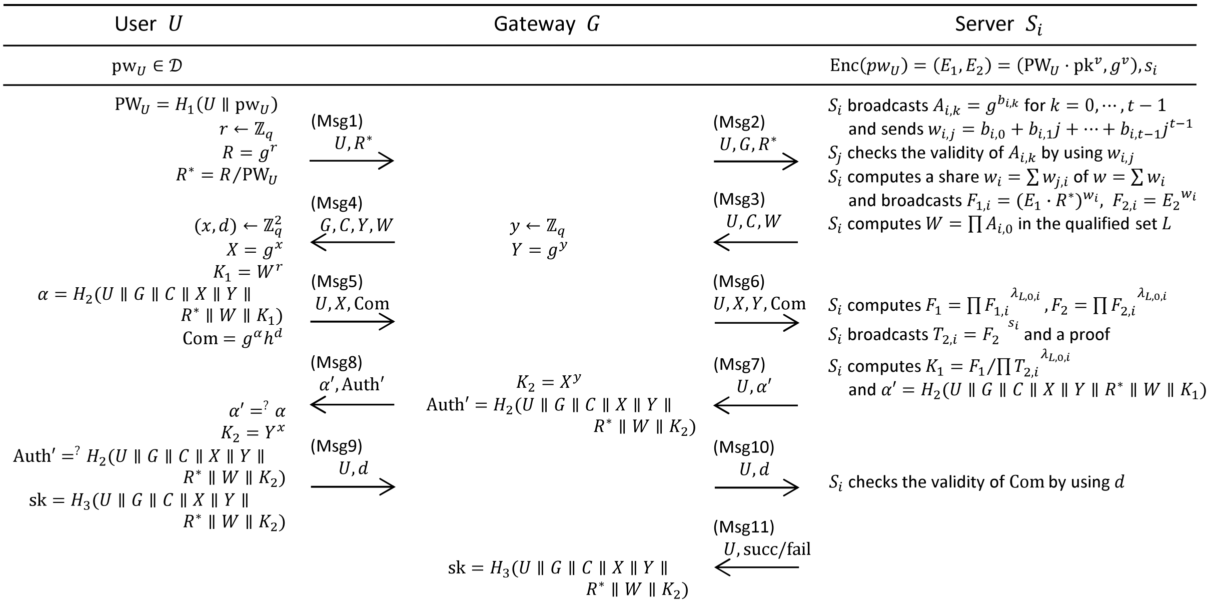
User U
Gateway G
Server S_i
$\mathrm{pw}_U \in \mathcal{D}$
$\mathrm{Enc}(pw_U) = (E_1, E_2) = (\mathrm{PW}_U \cdot \mathrm{pk}^v, g^v), s_i$
$\mathrm{PW}_U = H_1(U \parallel \mathrm{pw}_U)$
$r \leftarrow \mathbb{Z}_q$
$R = g^r$
$R^* = R/\mathrm{PW}_U$
(Msg1)
U, R^*
(Msg2)
U, G, R^*
S_i broadcasts $A_{i,k} = g^{b_{i,k}}$ 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, Y, W
$y \leftarrow \mathbb{Z}_q$
$Y = g^y$
$(x, d) \leftarrow \mathbb{Z}_q^2$
$X = g^x$
$K_1 = W^r$
$\alpha = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$
$\mathrm{Com} = g^{\alpha} h^d$
(Msg5)
U, X, Com
(Msg6)
U, X, Y, 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
(Msg8)
α', Auth'
$K_2 = X^y$
$\mathrm{Auth}' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$
(Msg7)
U, α'
S_i computes $K_1 = F_1/\prod T_{2,i}{}^{\lambda_{L,0,i}}$
and $\alpha' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$
$\alpha' =^? \alpha$
$K_2 = Y^x$
$\mathrm{Auth}' =^? H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$
$\mathrm{sk} = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$
(Msg9)
U, d
(Msg10)
U, d
S_i checks the validity of Com by using d
(Msg11)
U, succ/fail
$\mathrm{sk} = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$

Fig. 2 Overview of the proposed scheme.

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 $R = g^r$ and $R^* = R/PW_U$.
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, C, R^*) to all authentication servers.
The authentication server S_i generates two polynomials $g_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t-1}z^{t-1} \bmod q$ and $g'_i(z) = b'_{i,0} + b'_{i,1}z + \cdots + b'_{i,t-1}z^{t-1} \bmod q$ where $(b_{i,k}, b'_{i,k}) \leftarrow \mathbb{Z}_q^2$ for $k = 0, \ldots, t-1$.
S_i broadcasts $B_{i,k} = g^{b_{i,k}} h^{b'_{i,k}} \bmod p$ for $k = 0, \ldots, t-1$ and sends $w_{i,j} = g_i(j) \bmod q, w'_{i,j} = g'_i(j) \bmod q$ 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 more than or equal to the threshold t complaints is marked as disqualified.
S_j publishing the complaint against S_i's values $(w_{i,j}, w'_{i,j})$ satisfying the confirmation is also marked as disqualified.
The set L is defined as the subscript set of more than or equal to the threshold qualified 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 chooses $y \leftarrow \mathbb{Z}_q$ and computes $Y = g^y$.
G sends (G, C, Y, W) to U.

Step 5. The user U chooses $(x, d) \leftarrow \mathbb{Z}_q^2$.
U computes $X = g^x$, $K_1 = W^r$, $\alpha = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$, and the commitment $Com = g^{\alpha} h^{d}$[†].
U sends (U, X, Com) to G.

Step 6. The gateway G sends (U, X, Y, Com) to C.

Step 7. The combiner C broadcasts (U, X, Y, 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 the Lagrange coefficient.
S_i broadcasts $T_{2,i} = F_2^{s_i}$ and the 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 $\alpha' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$.
C sends (U, α') to G.

[†] The reason why our scheme uses a commitment scheme here (although it seems unnecessary) is explained in Appendix A. Without this commitment, it enables a malicious gateway to guess a password by combining subtle on-line and off-line dictionary attacks. We believe this shows the complexity that arises in considering the extension (i.e., GTPAKE) of traditional PAKE where more than two parties are involved.

Step 8. The gateway G computes $K_2 = X^y$ and an authenticator $Auth' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$.
G sends $(\alpha', Auth')$ 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, U increments the counter of incorrect login attempts for G, *reject* is output, and this sub-protocol is terminated[†].
Otherwise, U computes the session key $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$ 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, S_i increments the counter of incorrect login attempts for U, *reject* is output, and C sends $(U, failure)$ to G where *failure* means that the authentication process failed.
Otherwise, C sends $(U, success)$ to G where *success* means that the authentication process succeeded.

Step 12. The gateway G computes the session key $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$ and sk is output if *success* is received.
Otherwise (i.e., *failure* is received), the session is rejected.

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

$$\begin{aligned}\mathsf{Adv}^{\mathrm{sks}}_{\mathcal{L},\mathcal{D}}(\mathcal{A}) \le\ & \frac{q_{H_0}^2 + q_{H_1}^2 + (q_{exe} + q_{send})^2}{2q} + \frac{q_{H_2}^2 + q_{H_3}^2}{2^{\kappa+1}} \\ & + q_{exe} \cdot (q_{H_2} + q_{H_3}) \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathcal{A}}(\kappa) \\ & + \frac{q_{H_2} + q_{H_3}}{2^{\kappa}} + \frac{q_{sends} + q_{sendu}}{|\mathcal{D}|},\end{aligned}$$

where $q_{H_0}, q_{H_1}, q_{H_2}$, and q_{H_3} are the number of hash queries to the oracles H_0, H_1, H_2, and 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 number of queries to the SendServer and SendUser oracles, respectively, i.e., the equation $q_{sendu} + q_{sends} \le q_{send}$ holds, and $|\mathcal{D}|$ is the size of the dictionary $\mathcal{D}$.

Proof 1: We define $Succ^{\mathrm{sks}}$ in Game n as $Succ^{\mathrm{sks}}_n$.

Game 0. This experiment corresponds to a real attack by the adversary in the random oracle model. By Definition 5, and we have

$$\mathsf{Adv}^{\mathrm{sks}}_{\mathcal{L},\mathcal{D}}(\mathcal{A}) = |\Pr[Succ^{\mathrm{sks}}_0] - 1/2|.$$

Game 1. In this experiment, we simulate the hash functions and the oracle defined in Sect. 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 simulations 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 Execute, SendUser, SendGateway, SendServer, SessionKeyReveal, StaticKeyReveal, EphemeralKeyReveal, EstablishParty, Corrupt, and Test oracles can be simulated as follows.

- On a query SendUser$(U^{(\ell)}, *)$, we proceed as follows. If a query StaticKeyReveal(U), EphemeralKeyReveal$(U^{(\ell)})$ in any instance ℓ, or Corrupt(S_i) for more than or equal to t authentication servers has been asked by the adversary, then do nothing. If the processing request has come in the invalid order or the counter of incorrect login attempts exceeds the predetermined limit, we increment the counter of incorrect login attempts and do not reply.
  1. On a query SendUser$(U^{(\ell)}, \text{start})$, we proceed as follows.
     $PW_U = H_1(U \parallel pw_U)$; $r \leftarrow \mathbb{Z}_q$;
     $R = g^r$; $R^* = R/PW_U$; then return (U, R^*);
  2. On a query SendUser$(U^{(\ell)}, (G, C, Y, W))$, we proceed as follows.
     $(x, d) \leftarrow \mathbb{Z}_q^2$; $X = g^x$; $K_1 = W^r$;
     $\alpha = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$;
     $Com = g^{\alpha} h^{d}$; then return (U, X, Com);

[†] 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 a corrupted gateway can cause such malicious login failures.

3. On a query $\mathsf{SendUser}(U^{(\ell)}, (\alpha', Auth'))$, 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, $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$;
   then return (U, d);

- On a query $\mathsf{SendServer}(C^{(k)}, *)$, we proceed as follows.
  If a query $\mathsf{StaticKeyReveal}(G)$ or $\mathsf{StaticKeyReveal}(C)$ has been asked by the adversary, then do nothing. If the processing request has come in the invalid order or the counter of incorrect login attempts exceeds the predetermined limit, we increment the counter of incorrect login attempts and do not reply. In the following queries, the adversary does the process on behalf of the corrupted authentication server S_j.

  1. On a query $\mathsf{SendServer}(C^{(k)}, (U, G, R^*))$, 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 the adversary;
     S_i checks the validity of $w_{j,i}, w'_{j,i}$ by using $B_{j,k}$;
     S_i computes the 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 $\mathsf{SendServer}(C^{(k)}, (U, X, Y, Com))$, 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 the 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 $\alpha' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$;
     then return (U, α');
  3. On a query $\mathsf{SendServer}(C^{(k)}, (U, d))$, 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 $\mathsf{SendGateway}(G^{(j)}, *)$, we proceed as follows.
  If a query $\mathsf{StaticKeyReveal}(U)$, $\mathsf{EphemeralKeyReveal}(U^{(\ell)})$ in any instance ℓ, or $\mathsf{Corrupt}(S_i)$ for more than or equal to t authentication servers has been asked by the adversary, then do nothing. If the processing request has come in the invalid order, then do not reply.

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

- On a query $\mathsf{Execute}(U^{(\ell)}, G^{(j)}, C^{(k)})$, we proceed as follows.
  $(U, R^*) \leftarrow \mathsf{SendUser}(U^{(\ell)}, \text{start})$;
  $(U, G, R^*) \leftarrow \mathsf{SendGateway}(G^{(j)}, (U, R^*))$;
  $(U, C, W) \leftarrow \mathsf{SendServer}(C^{(k)}, (U, G, R^*))$;
  $(G, C, Y, W) \leftarrow \mathsf{SendGateway}(G^{(j)}, (U, C, W))$;
  $(U, X, Com) \leftarrow \mathsf{SendUser}(U^{(\ell)}, (G, C, Y, W))$;
  $(U, X, Y, Com) \leftarrow \mathsf{SendGateway}(G^{(j)}, (U, X, Com))$;
  $(U, \alpha') \leftarrow \mathsf{SendServer}(C^{(k)}, (U, X, Y, Com))$;
  $(\alpha', Auth') \leftarrow \mathsf{SendGateway}(G^{(j)}, (U, \alpha'))$;
  $(U, d) \leftarrow \mathsf{SendUser}(U^{(\ell)}, (\alpha', Auth'))$;
  $(U, d) \leftarrow \mathsf{SendGateway}(G^{(j)}, (U, d))$;
  $(U, success/failure) \leftarrow \mathsf{SendServer}(C^{(k)}, (U, d))$;
  $\mathsf{SendGateway}(G^{(j)}, (U, success/failure))$;
  then return $(U, G, C, X, Y, R^*, W, \alpha, \alpha', Auth, Auth', Com, d, success/failure)$;
- On a query $\mathsf{SessionKeyReveal}(P^{(\ell)})$, we proceed as follows.
  If the session key sk is defined for the user or gateway instance $P^{(\ell)}$ then return sk, else return $\perp$;
- On a query $\mathsf{StaticKeyReveal}(P)$, 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 the secret key for the authenticated channels between the gateway and authentication servers;
  If P is an authentication server S_i, then return the encrypted password $\mathsf{Enc}(pw) = (PW \cdot pk^v, g^v)$ 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 the authenticated channels between the authentication server and a gateway and secure channels among authentication servers;

- On a query EphemeralKeyReveal($P^{(\ell)}$), we proceed as follows.
  If there are already the ephemeral keys generated by the instance $P^{(\ell)}$, then return the ephemeral keys, else return $\perp$;
- On a query Corrupt(S_i), we proceed as follows.
  We return all 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($U^{(\ell)}$), we proceed as follows.
  $sk \leftarrow$ SessionKeyReveal($P^{(\ell)}$);
  If $sk = \perp$, then return $\perp$, else $b \leftarrow \{0,1\}$;
  If $b = 1$ (resp. $b = 0$), $sk' = sk$ (resp. $sk' \leftarrow \{0,1\}^\kappa$), then return sk';

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

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

Game 2. We halt this experiment when a 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 using the bound described in Appendix B (i.e., the value n corresponds to q or 2^κ and the value m corresponds to $q_{H_0}, q_{H_1}, q_{H_2}, q_{H_3}$, or $(q_{exe} + q_{send})$), we have

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

Game 3. We change the simulation of queries to the Execute oracle, especially queries to the SendUser and SendGateway oracles on a test session. When a query SendGateway($G^{(j)}, (U, \alpha')$), SendUser($U^{(\ell)}, (\alpha', Auth')$), or SendGateway($G^{(j)}, (U, success/failure)$) is asked, we compute the authenticator $Auth'$ and the session key sk as $H_2'(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W)$ and $H_3'(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W)$ 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 $(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$ 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

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

To evaluate the probability of this event, we construct an algorithm to solve the CDH problem. The algorithm obtains the CDH tuple (M, N) and chooses the session $(U^{(\ell)}, G^{(j)})$. We set $X = M^{u_1} g^{u_2}$ for a query SendUser($U^{(\ell)}, (G, C, Y, W)$) and $Y = N^{u_3} g^{u_4}$ for a query SendGateway($G^{(j)}, (U, C, W)$) where $(u_1, u_2, u_3, u_4) \leftarrow \mathbb{Z}_q^4$. We compute α and $Auth'$ as $H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W)$ and sk as $H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W)$ for queries SendUser($U^{(\ell)}, (G, C, Y, W)$), SendUser($U^{(\ell)}, (\alpha', Auth')$), SendGateway($G^{(j)}, (U, C, W)$), SendGateway($G^{(j)}, (U, \alpha')$), and SendGateway($G^{(j)}, (U, success/failure)$). All other queries are handled in the same way as the previous game. The simulator picks a test session with the probability $1/q_{exe}$. Although the simulator cannot obtain the 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 $(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$ to the hash function H_2 or H_3 in the session $(U^{(\ell)}, G^{(j)})$, we can compute $K_2 = \mathrm{CDH}(M^{u_1} g^{u_2}, N^{u_3} g^{u_4}) = \mathrm{CDH}(M^{u_1}, N^{u_3}) \cdot \mathrm{CDH}(M^{u_1}, g^{u_4}) \cdot \mathrm{CDH}(g^{u_2}, N^{u_3}) \cdot \mathrm{CDH}(g^{u_2}, g^{u_4}) = \mathrm{CDH}(M, N)^{u_1 u_3} \cdot M^{u_1 u_4} \cdot N^{u_2 u_3} \cdot g^{u_2 u_4}$ 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 in [7]. In this way, we can extract the value $\mathrm{CDH}(M, N)$ from the given tuple (M, N).

In Game 3, the Diffie-Hellman key K_2 is random and independent from any other ephemeral and static keys such as passwords. In this situation, only three ways to distinguish a session key from a random key are left as follows.

Case 1. The adversary asks the query $(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$ to the hash function H_2 or H_3.

The probability that Case 1 occurs is $(q_{H_2} + q_{H_3})/2^\kappa$.

Case 2. The adversary asks the SendGateway or SendServer oracle except for SendGateway($G^{(j)}, m$) and successfully shares the session key with $G^{(j)}$.

In order for the adversary to share the session key with the gateway, the adversary needs to convince the gateway by sending valid commitments Com and d to the combiner and giving $Succ$ to the gateway. Also we note that an authenticated channel is established between the gateway and combiner in our security model. Therefore, even if the adversary tries to send Com and d to the combiner directly (i.e., not via the gateway) or to send $Succ$ to the gateway directly (i.e., not via the combiner), this attack fails because the adversary is not allowed to obtain the static key for establishing the authenticated channel by Definition 4. Thus, the adversary has no choice but to do active attacks to authentication servers via the gateway through queries to the SendServer oracle. Therefore, the probability that the adversary successfully gives $Succ$ to the gateway without communicating with the gateway and combiner is negligible. However, the adversary cannot obtain information of passwords unless active attacks are done as Theorem 2, and thus, the probability that Case 2 occurs is at most $q_{sends}/|\mathcal{D}|$ where q_{sends} is the number of queries to the SendServer oracle.

Case 3. The adversary asks the SendUser oracle except for SendUser($U^{(\ell)}, m$) and successfully shares the session key with $U^{(\ell)}$.

The adversary needs to send valid authenticators α' and $Auth'$ to impersonate G to U. However, the adversary cannot obtain information of passwords unless active attacks are done as Theorem 2. According to Definition 4, the adversary is not allowed to obtain static keys and ephemeral keys of the user and corrupt more than or equal to t authentication servers. The probability that Case 3 occurs is at most $q_{sendu}/|\mathcal{D}|$ where q_{sendu} is the number of queries to the SendUser oracle.

As a result, the probability that the adversary succeeds in guessing the challenge bit b at random, we have

$$\Pr[Succ_3^{\mathrm{sks}}] = \frac{1}{2} + \frac{q_{H_2} + q_{H_3}}{2^{\kappa}} + \frac{q_{sends} + q_{sendu}}{|\mathcal{D}|}.$$

□

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 scheme and $\mathcal{A}$ be a probabilistic polynomial time adversary that corrupts at most $(t-1)(< n/2)$ authentication servers in advance. Then the advantage of $\mathcal{A}$ for the password protection security in $\mathcal{L}$ with at most $\mathcal{T}$ time and $\mathcal{R}$ resources is

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

where $q_{H_0}, q_{H_1}, q_{H_2}$, and q_{H_3} are the number of hash queries to the oracles H_0, H_1, H_2, and 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 number of queries to the SendServer and SendUser oracles, respectively, i.e., the equation $q_{sendu} + q_{sends} \leq q_{send}$ holds, and $|\mathcal{D}|$ is the size of the dictionary $\mathcal{D}$.

Proof 2: We define $Succ^{\mathrm{pps}}$ in Game n as $Succ_n^{\mathrm{pps}}$.

Game 0. This experiment corresponds to a real attack by the adversary in the random oracle model. By Definition 7, and we have

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

Game 1. As the proof in Sect. 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 follows.

- On a query TestPassword(U, pw'_U), we proceed as follows.
  $pw_U \leftarrow$ 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

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

Game 2. We halt this experiment when a collision on the outputs of the hash oracles and the transcripts occurs. By using the bound described in Appendix B (i.e., the value n corresponds to q or 2^{κ} and the value m corresponds to q_{H_0}, q_{H_1}, q_{H_2}, q_{H_3}, or $(q_{exe} + q_{send})$), we have

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

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 SendServer$(C^{(k)}, (U, X, Y, Com))$ or Corrupt(S_i) is asked, we replace the secret part needed to decrypt the stored ElGamal ciphertext 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

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

We suppose that we have a successful distinguisher between Games 2 and 3, 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 in the well-defined set L. The simulator controls the other uncorrupted authentication servers as usual.

First, we deal with the simulation about the public key in the sub-protocol of setup 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 knowing 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 (M, N, Z), the corresponding share for S_n is set as $g^{s_n} = M^{\lambda_{L,n,0}} \cdot \prod_{j \in (L \setminus \{n\})} (g^{s_j})^{\lambda_{L,n,j}}$ where $\lambda_{L,i,j} = \prod_{\{k \in L \wedge k \neq j\}} (i-k)/(j-k)$ is the Lagrange coefficient. Due to the honest majority setting, the simulator can compute all the share s_j of the 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 the authentication servers. In this case, we can simulate the public key by hitting the intended value similar to the distributed key generation technique [12].

Second, we deal with the simulation about the stored passwords in the sub-protocol of registration 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 Corrupt(S_i) is asked, we embed the DDH triple into all encrypted passwords. The simulator returns $(PW_{U_i} \cdot Z^{v_i}, N^{v_i})$ 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 sub-protocol of authentication Auth. We need to simulate the uncorrupted authentication servers for SendServer($C^{(k)}, (U, X, Y, Com)$). The share $T_{2,n}$ for U_i can be computed as $Z^{w \cdot v_i \cdot \lambda_{L,n,0}} \cdot \prod_{j \in (L \backslash \{n\})} N^{w \cdot v_i \cdot s_j \cdot \lambda_{L,n,j}}$ without s_n. Other parameters generated from the 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. 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 (M, N, Z). The simulator is able to solve the DDH problem by using the difference of success probability that an adversary guesses the registered password of one honest user through the TestPassword oracle because all the encrypted passwords for all honest users include the DDH triple. In the case of $Z = \mathrm{CDH}(M, N)$, the environment for the distinguisher corresponds to Game 2. In the case of $Z \neq \mathrm{CDH}(M, N)$, 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 Execute oracle, especially the SendUser and SendServer oracles on a test session. When a query SendUser($U^{(\ell)}$, start), SendUser($U^{(\ell)}, (G, C, Y, W)$), or SendServer($C^{(k)}, (U, G, R^*)$) is asked, we replace the Diffie-Hellman key in the authenticators α and α' with random elements. 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 (M, N, Z) and chooses the session $(U^{(\ell)}, C^{(k)})$. The distinguisher picks a test session with the probability $1/q_{exe}$. We show the simulation of the SendUser and SendServer oracles without ephemeral keys $\log_g R$ and $\log_g W$ as follows.

We set $R = M^{u_1}$ for a query SendUser($U^{(\ell)}$, start) and $K_1 = Z^{u_1 u_2}$ for a query SendUser($U^{(\ell)}, (G, C, Y, W)$) where $(u_1, u_2) \leftarrow \mathbb{Z}_q^2$. When a query SendServer($C^{(k)}, (U, G, R^*)$) is asked, we need to simulate the process of honest authentication servers sending and receiving information privately and publicly for corrupted parties. 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 assign $A_{i,k} = g^{b_{i,k}}$ for $i \in (L \backslash \{n\}), k = 0, \ldots, t-1$ and compute $A_{n,0} = N^{u_2} \cdot \prod_{i \in (L \backslash \{n\})} (A_{i,0})^{-1}$. 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 $A_{n,k}$, $B_{n,k} = A_{n,k} h^{b'_{n,k}} \bmod p$ for $k = 0, \ldots, t-1$, $F_{1,n} = (N^{u_2 sv} \cdot Z^{u_1 u_2})^{\lambda_{L,n,0}} \cdot \prod_{j \in (L \backslash \{n\})} (M^{u_1} \cdot pk^v)^{w_j \cdot \lambda_{L,n,j}}$, and $F_{2,n} = (N^{u_2 v})^{\lambda_{L,n,0}} \cdot \prod_{j \in (L \backslash \{n\})} (E_2)^{w_j \cdot \lambda_{L,n,j}}$. All other processes are handled in the same way as 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. We note that the password obtained via the 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 SendUser and 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 $Z = \mathrm{CDH}(M, N)$, the environment for the distinguisher corresponds to Game 3. In the case of $Z \neq \mathrm{CDH}(M, N)$, 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 SendServer oracle. When a query SendServer($C^{(k)}, (U, X, Y, Com)$) is asked, we compute the authenticator α' as $H'_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W)$ 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 $(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$ to the hash function H_2 where $K_1 = \mathrm{CDH}(R^* \cdot PW_U, W)$.

There is at most one hash value PW such that $K_1 = \mathrm{CDH}(R^* \cdot PW, W)$ for some pair (R^*, W) as a result of Lemma 2 in [7]. Therefore, the probability that this bad event occurs in the non-concurrent setting is bounded by the probability that the adversary guesses the password at random through the SendServer oracle to which q_{sends} is the number of queries with at most $\mathcal{T}$ time and $\mathcal{R}$ resources 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 a query SendUser($U^{(\ell)}$, start) is

[†] The reason why the simulator assigns $(R, W, \mathrm{CDH}(R, W))$ for the triple not (M, N, Z) but $(M^{u_1}, N^{u_2}, Z^{u_1 u_2})$ is to differ the outputs of the SendUser and SendServer oracles in each session.

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 the adversary trying 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 by at most one password in the non-concurrent setting. Then the success probability with at most $\mathcal{T}$ time and $\mathcal{R}$ resources is at most $q_{sendu}/|\mathcal{D}|$ where q_{sendu} is the number of queries to the SendUser oracle, we have

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

In this game, information of the password obtained via the SendUser and SendServer oracles is not used in sessions, so the adversary 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 to be one of its applications. Our scheme is an instantiation of GTPAKE, and the generic construction of GPAKE and GTPAKE is left as future work.

Acknowledgements

We would like to thank anonymous reviewers and Shin-Akarui-Angou-Benkyou-Kai for their valuable comments. This work was supported in part by JSPS KAKENHI Grant Number 17K00178, Institute for Information and & communications Technology Promotion (IITP) grant (2017-0-00555), and National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2015R1A-2A2A01006812).

References

[1] M. Abdalla, O. Chevassut, P.A. Fouque, and D. Pointcheval, “A simple threshold authenticated key exchange from short secrets,” ASIACRYPT 2005, LNCS, vol.3788, pp.566–584, 2005.

[2] M. Abdalla, M. Izabachene, and D. Pointcheval, “Anonymous and transparent gateway-based password-authenticated key exchange,” CANS 2008, LNCS, vol.5339, pp.133–148, 2008.

[3] M. Bellare, D. Pointcheval, and P. Rogaway, “Authenticated key exchange secure against dictionary attacks,” EUROCRYPT 2000, LNCS, vol.1807, pp.139–155, 2000.

[4] S. Bellovin and M. Merritt, “Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise,” ACM CCS 1993, pp.244–250, 1993.

[5] S. Bellovin and M. Merritt, “Encrypted key exchange: Password based protocols secure against dictionary attacks,” Proc. IEEE Symposium on Research in Security and Privacy, pp.72–84, 1992.

[6] V. Boyko, P. MacKenzie, and S. Patel, “Provably secure password-authenticated key exchange using Diffie-Hellman,” EUROCRYPT 2000, LNCS, vol.1807, pp.156–171, 2000.

[7] E. Bresson, O. Chevassut, and D. Pointcheval, “New security results on encrypted key exchange,” PKC 2004, LNCS, vol.2947, pp.145–158, 2004.

[8] J.W. Byun, D.H. Lee, and J.I. Lim, “Security analysis and improvement of a gateway-oriented password-based authenticated key exchange protocol,” IEEE Commun. Lett., vol.10, no.9, pp.683–685, 2006.

[9] Y. Ding and P. Horster, “Undetectable on-line password guessing attacks,” Operating Systems Review, vol.29, no.4, pp.77–86, 1995.

[10] European Network and Information Security Agency, “Cloud computing risk assessment,” 2009.

[11] European Network and Information Security Agency, “Heartbleed Wake Up Call,” 2014.

[12] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Secure distributed key generation for discrete-log based cryptosystems,” J. Cryptology, vol.20, no.1, pp.51–83, 2007.

[13] O. Goldreich and Y. Lindell, “Session-key generation using human passwords only,” CRYPTO 2001, LNCS, vol.2139, pp.408–432, 2001.

[14] S. Goldwasser and M. Bellare, “Lecture notes on cryptography,” Summer Course “Cryptography and computer security” at MIT, 1999:1999, 1996.

[15] J. Katz, R. Ostrovsky, and M. Yung, “Efficient password-authenticated key exchange using human-memorable passwords,” EUROCRYPT 2001, LNCS, vol.2045, pp.475–494, 2001.

[16] Y. Kobayashi, N. Yanai, K. Yoneyama, T. Nishide, G. Hanaoka, K. Kim, and E. Okamoto, “Gateway threshold password-based authenticated key exchange secure against undetectable on-line dictionary attack,” SECRYPT 2015, pp.39–52, 2015.

[17] M. Szydlo, “A note on chosen-basis decisional Diffie-Hellman assumptions,” FC 2006, LNCS, vol.4107, pp.166–170, 2006.

[18] F. Wei, C. Ma, and Z. Zhang, “Gateway-oriented password-authenticated key exchange protocol with stronger security,” ProvSec 2011, LNCS, vol.6980, pp.366–379, 2011.

[19] F. Wei, Z. Zhang, and C. Ma, “Analysis and enhancement of an optimized gateway-oriented password-based authenticated key exchange protocol,” IEICE Trans. Fundamentals, vol.E96-A, no.9, pp.1864–1871, Sept. 2013.

[20] F. Wei, Z. Zhang, and C. Ma, “Gateway-oriented password-authenticated key exchange protocol in the standard model,” J. Systems and Software, vol.85, no.3, pp.760–768, 2012.

Appendix A: Security Analysis of the Scheme without Commitments

A.1 Construction

We mention the insecure construction to explain the reason why the proposed scheme needs a commitment scheme, although it seems unnecessary. We describe the following sub-protocol of authentication composed of nine steps. We note that the sub-protocol of setup and registration in this scheme are the same as those of the proposed scheme in Sect. 4.3. A perspective of the insecure scheme is shown in Fig. A· 1.

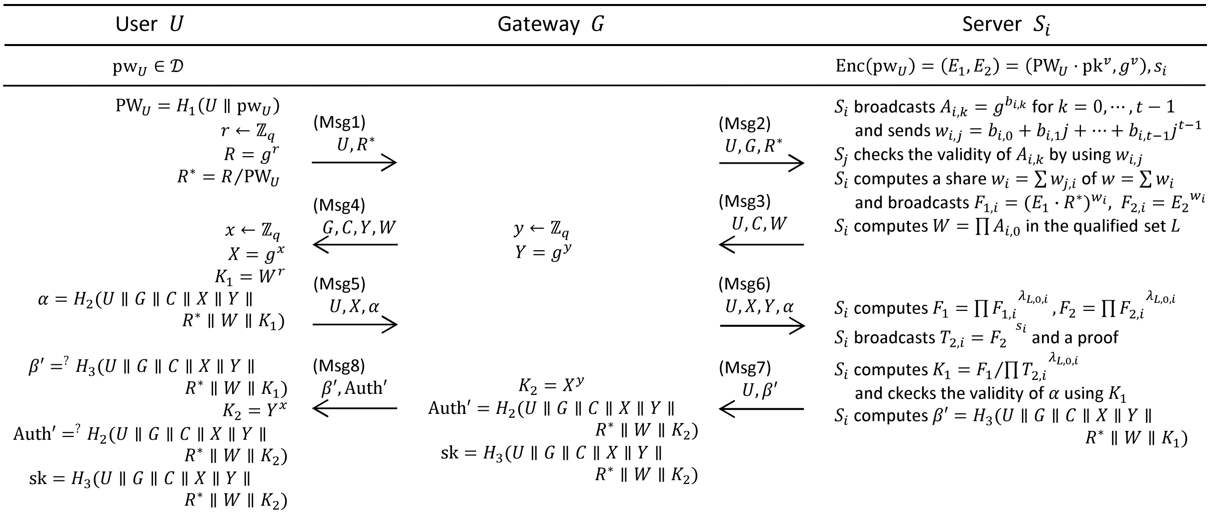
User U
Gateway G
Server S_i
$\mathrm{pw}_U \in \mathcal{D}$
$\mathrm{Enc}(\mathrm{pw}_U) = (E_1, E_2) = (\mathrm{PW}_U \cdot \mathrm{pk}^v, g^v), s_i$
$\mathrm{PW}_U = H_1(U \parallel \mathrm{pw}_U)$
$r \leftarrow \mathbb{Z}_q$
$R = g^r$
$R^* = R/\mathrm{PW}_U$
(Msg1) U, R^*
(Msg2) U, G, R^*
S_i broadcasts $A_{i,k} = g^{b_{i,k}}$ 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, Y, W
$y \leftarrow \mathbb{Z}_q$
$Y = g^y$
$x \leftarrow \mathbb{Z}_q$
$X = g^x$
$K_1 = W^r$
$\alpha = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$
(Msg5) U, X, α
(Msg6) U, X, Y, α
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 ckecks the validity of α using K_1
S_i computes $\beta' = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$
(Msg7) U, β'
$K_2 = X^y$
$\mathrm{Auth}' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$
$\mathrm{sk} = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$
(Msg8) β', Auth'
$\beta' =^? H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$
$K_2 = Y^x$
$\mathrm{Auth}' =^? H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$
$\mathrm{sk} = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$

Fig. A· 1 Overview of the insecure scheme.

Auth. If a processing request has come in the invalid order, the incorrect login attempt is counted and the session is rejected. After the counter of incorrect login attempts exceeds the predetermined limit, a processing request from the user 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 $R = g^r$ and $R^* = R/PW_U$.
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 $g_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t-1}z^{t-1} \bmod q$ and $g'_i(z) = b'_{i,0} + b'_{i,1}z + \cdots + b'_{i,t-1}z^{t-1} \bmod q$ where $(b_{i,k}, b'_{i,k}) \leftarrow \mathbb{Z}_q^2$ for $k = 0, \ldots, t-1$.
S_i broadcasts $B_{i,k} = g^{b_{i,k}}h^{b'_{i,k}} \bmod p$ for $k = 0, \ldots, t-1$ and sends $w_{i,j} = g_i(j) \bmod q, w'_{i,j} = g'_i(j) \bmod q$ 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 more than or equal to the threshold t complaints is marked as disqualified.
S_j publishing the complaint against S_i's values $(w_{i,j}, w'_{i,j})$ satisfying the confirmation is also marked as disqualified.
The set L is defined as the subscript set of more than or equal to the threshold qualified 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 chooses $y \leftarrow \mathbb{Z}_q$ and computes $Y = g^y$.
G sends (G, C, Y, W) to U.

Step 5. The user U chooses $x \leftarrow \mathbb{Z}_q$ and computes $X = g^x$, $K_1 = W^r$, and $\alpha = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$.
U sends (U, X, α) to G.

Step 6. The gateway G sends (U, X, Y, α) to C.

Step 7. The combiner C broadcasts (U, X, Y, α) 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 the Lagrange coefficient.
S_i broadcasts $T_{2,i} = F_2^{s_i}$ and the 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 checks the validity of α using K_1.
If the confirmation is false, S_i increments the counter of incorrect login attempts for U, *reject* is output, and this subprotocol is terminated.
Otherwise, C computes $\beta' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_1)$ and sends (U, β') to G.

Step 8. The gateway G computes $K_2 = X^y$, an authenticator $Auth' = H_2(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$ and the session key $sk = H_3(U \parallel G \parallel C \parallel X \parallel Y \parallel R^* \parallel W \parallel K_2)$.
G sends $(\beta', Auth')$ to U.

Step 9. The user U checks the validity of β' by using K_1.
U computes $K_2 = Y^x$ and checks the validity of $Auth'$ by using K_2.
If one of the two confirmations is false, U increments the counter of incorrect login attempts for G and *reject* is output.
Otherwise, U computes the session key $sk = H_3(U \| G \| C \| X \| Y \| R^* \| W \| K_2)$ and sk is output.

A.2 Attack against Password Protection Security

We describe the following method for an adversary acting as a malicious gateway to guess the password of a target user by a combination of the on-line and off-line dictionary attacks in the scheme described in Sect. A.1. To avoid this attack, the proposed scheme in Sect. 4.3 uses a commitment scheme that can hide the calculated intermediate results among the authentication servers.

1. The active adversary interacts with a target user U once as below where instances of a user, a malicious gateway, and a combiner are $U^{(\ell)}$, $G'^{(j)}$, and $C^{(k)}$ respectively. We note that this attack in the on-line manner is detected by the honest authentication servers correctly because the value generated by the adversary at random does not pass the verification with overwhelming probability. However, we also note that this attack is quite powerful in the sense that this only one probe makes the whole password guessing attack possible.
   a. The adversary (i.e., the malicious gateway G') obtains R^* in the message (U, R^*) from U.
   b. The adversary chooses the random numbers $(y, w') \leftarrow \mathbb{Z}_q^2$ and computes $Y = g^y, W' = g^{w'}$. The adversary sends (G', C, Y, W') to U and obtains α' in the response message (U, X, α') from U.
2. The passive adversary guesses the password pw_U in the off-line manner by using the obtained data as below.
   a. The adversary chooses the password pw'_U from the dictionary $\mathcal{D}$ and computes $R' = R^* \cdot H_1(U \| pw'_U)$.
   b. The adversary checks whether $H_2(U \| G' \| C \| X \| Y' \| R^* \| W' \| (R')^{w'})$ equals the hash value α'.
   c. The adversary repeats Steps (a) and (b) until the password satisfying the equation in (b) (that is the correct password) is detected.

Appendix B: The Bound Related to the Birthday Problem

We explain how to derive the bound of the probability of the collision on the outputs of the hash function used in the proof of Theorems 1 and 2. This probability is widely discussed as the birthday problem [14], which asks for the probability that at least one pair in a set of m people will have the same birthday. We compute the probability $p(n, m)$ that there is at least one collision in the set of m hash values computed from hash function H which outputs n kinds of hash values as below.

When the first value is chosen, the probability of no collision is 1. When the second value is chosen, the probability of no collision is $1 - 1/n$. When the third value is chosen, the probability of no collision is $1 - 2/n$. Similarly, when the m-th value is chosen, the probability of no collision is $1 - (m - 1)/n$. The probability that at least one collision happens is

$$p(n, m) = 1 - (1 - 1/n)(1 - 2/n) \cdot (1 - (m - 1)/n).$$

By using a first-order approximation for e^x where e is Napier's constant and $|x| \ll 1$, we have $\mathrm{e}^x \approx (1 + x)$. Thus when the equation $m \ll n$ holds,

$$\begin{aligned} p(n, m) &\approx 1 - \mathrm{e}^{-1/n} \cdot \mathrm{e}^{-2/n} \cdots \mathrm{e}^{-(m-1)/n} \\ &= 1 - \mathrm{e}^{-(1+2+\cdots+(m-1))/n} \\ &= 1 - \mathrm{e}^{-m(m-1)/2n} \\ &\approx 1 - (1 - m(m - 1)/2n) \\ &= m(m - 1)/2n \\ &< m^2/2n. \end{aligned}$$

Yukou Kobayashi received the B.S. degree from the School of Informatics at University of Tsukuba in 2013 and the M.E. degree from the School of Systems and Information Engineering at University of Tsukuba in 2015. He is currently working as a security engineer at LAC Co., Ltd.

Naoto Yanai received B.E. degree in Electrical Engineering from Ichinoseki National College of Technology, Japan, in 2009, and M.E. and Ph.D. in Graduate School of Information Engineering from University of Tsukuba, Japan, in 2011 and 2014, respectively. He is currently an assistant professor at Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University. His research interests are cryptography and information security.

Kazuki Yoneyama received the B.E., M.E. and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 2004, 2006 and 2008, respectively. He was a researcher of NTT Secure Platform Laboratories from 2009 to 2015. He is presently engaged in research on cryptography at the Ibaraki University, since 2015. He is a member of the International Association for Cryptologic Research (IACR), IPSJ and JSIAM.

Takashi Nishide received B.S. degree from the University of Tokyo in 1997, M.S. degree from the University of Southern California in 2003, and Dr.E. degree from the University of Electro-Communications in 2008. From 1997 to 2009, he had worked at Hitachi Software Engineering Co., Ltd. developing security products. From 2009 to 2013, he had been an assistant professor at Kyushu University and from 2013 he is an associate professor at University of Tsukuba. His research is in the areas of cryptography and information security.

Goichiro Hanaoka Graduated from the Department of Engineering, The University of Tokyo in 1997. Received Ph.D. degree from The University of Tokyo in 2002. Joined AIST in 2005. Currently, Leader, Advanced Cryptosystems Research Group, Information Technology Research Institute, AIST. Engages in the R&Ds for encryption and information security technologies including the efficient design and security evaluation of public key cryptosystem. Received the Wilkes Award (2007), British Computer Society; Best Paper Award (2008), The Institute of Electronics, Information and Communication Engineers; Innovative Paper Award (2012, 2014), Symposium on Cryptography and Information Security (SCIS); Award of Telecommunication Advancement Foundation (2005); 20th Anniversary Award (2005), SCIS; Best Paper Award (2006), SCIS; Encouragement Award (2000), Symposium on Information Theory and its Applications (SITA); and others.

Kwangjo Kim received the B.S. and M.S. degrees of Electronic Engineering in Yonsei University, Korea in 1980 and 1883, respectively and Ph.D. of Div. of Electrical and Computer Engineering in Yokohama National University, Japan in 1991. He has served board member of International Association for Cryptologic Research (IACR) from 2000 to 2004, chairperson of Asiacrypt Steering Committee from 2005 to 2008 and President of Korea Institute of Information Security and Cryptography (KIISC) in 2009. Also he was a visiting professor in MIT (2005), UCSD (2005), KUSTAR, UAE (2012) and an education specialist in ITB, Indonesia (2013). He is currently a professor at School of Computing in KAIST, Korea, Korea Representative to IFIP TC-11 and Honorable President of KIISC. His research interests include the theory of cryptology and information security and its application. He is members of IEICE, IEEE, IACR and ACM. He is a Fellow of the IACR.

Eiji Okamoto received his B.S., M.S. and Ph.D. degrees in electronics engineering from the Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He worked and studied communication theory and cryptography for NEC central research laboratories since 1978. In 1991 he became a professor at Japan Advanced Institute of Science and Technology, then at Toho University. Now he is a professor at Faculty of Engineering, Information and Systems, University of Tsukuba. His research interests are cryptography and information security. He is members of IEEE and ACM.