Efficient Threshold Password-Authenticated Secret Sharing Protocols for Cloud Computing

Xun Yi*

Computer Science and Software Engineering, RMIT University, Australia

Zahir Tari

Computer Science and Software Engineering, RMIT University, Australia

Feng Hao

Department of Computer Science, University of Warwick, United Kingdom

Liqun Chen

Department of Computer Science, University of Surrey, United Kingdom

Joseph K. Liu

Faculty of Information Technology, Monash University, Australia

Xuechao Yang

Computer Science and Software Engineering, RMIT University, Australia

Kwok-Yan Lam

School of Computer Science and Engineering, Nanyang Technological University, Singapore

Ibrahim Khalil

Computer Science and Software Engineering, RMIT University, Australia

Albert Y. Zomaya

School of Information Technologies, University of Sydney, Australia

*Corresponding author at: Computer Science and Software Engineering, School of Science, RMIT University, GPO Box 2476, Melbourne, Victoria 3001, Australia

Email address: `xun.yi@rmit.edu.au` (Xun Yi*)

Abstract

Threshold password-authenticated secret sharing (TPASS) protocols allow a client to distribute a secret s amongst n servers and protect it with a password pw, so that the client can later recover the secret s from any subset of t of the servers using the password pw. In this paper, we present two efficient TPASS protocols, one is built on two-phase commitment and has lower computation complexity, and another is based on zero-knowledge proof and has less communication rounds. Both protocols are in particular efficient for the client, who only needs to send a request and receive a response. In addition, we have provided rigorous proofs of security for the proposed protocols in the standard model. The experimental results have shown that the proposed two TPASS protocols are more efficient than Camenisch et al.'s protocols and save up to $85\% - 95\%$ total computational time and up to $65\% - 75\%$ total communication overhead.

Keywords: Threshold password-authenticated secret sharing protocol, ElGamal encryption scheme, Shamir secret sharing scheme, Diffie-Hellman problems

1. Introduction

Threshold password-authenticated secret sharing (TPASS) protocols consider a scenario [6], inspired by the movie "Memento" in which the main character suffers from short-term memory loss, leads to an interesting cryptographic problem, can a user securely recover his secrets from a set of servers, if all the user can or wants to remember is a single password and all of the servers may be adversarial? In particular, can he protect his previous password when accidentally trying to run the recovery with all-malicious servers? A solution for this problem can act as a natural bridge from human-memorisable passwords to strong keys for cryptographic tasks.

A typical application of TPASS is to protect user data in cloud computing environments, where a client encrypts his data with a random key before uploading it to a data server in the cloud, then secretly shares the random key together with a password with n key servers in the cloud. When he needs the key to decrypt his data downloaded from the data server, he recovers the key from any subset of t of the n key servers using his password. Therefore, to protect his data in the cloud, the client needs to remember his password only. This process can be illustrated in Fig. 1, where we assume that the gateway forwards messages between the client and key servers.

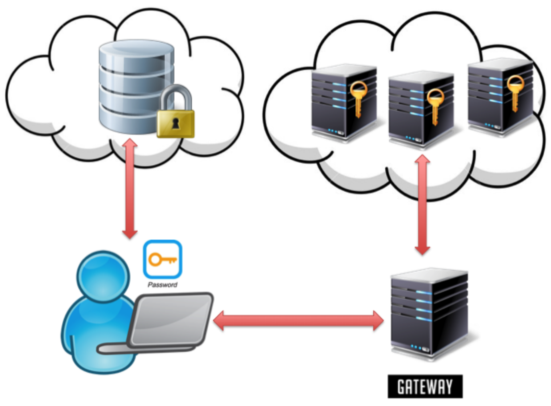


Figure 1: Cloud Security with TPASS

The first TPASS protocol was given by Bagherzandi et al. [2] in 2011. It is built on the PKI model, secure under the decisional Diffie-Hellman assumption, using non-interactive zero-knowledge proofs. The basic idea is as follows: a client initially generates an ElGamal private and public key pairs $(sk, pk = g^{sk})$ [8] and secret-shares sk among servers using an t-out-of-n secret sharing [16] and outputs public parameters including the public key pk and the encryptions $E(g^{\mathsf{pw}}, pk)$ and $E(s, pk)$ of password pw and secret s, respectively, under the public key pk. When retrieving the secret from the servers, the client encrypts the password pw' he remembers and send the encryption $E(g^{\mathsf{pw}'}, pk)$ to the servers, each of which computes and returns $A_i = [E(g^{\mathsf{pw}}, pk)/E(g^{\mathsf{pw}'}, pk)]^{r_i} = E(g^{r_i(\mathsf{pw}-\mathsf{pw}')}, pk)$, where r_i is randomly chosen. The client then computes $A = \prod_{i=1}^{n} A_i$ and sends it to the servers. In the end, t servers cooperate to decrypt $B = E(s, pk)A = E(sg^{\sum r_i(\mathsf{pw}-\mathsf{pw}')}, pk)$ and sends partial decryptions to the client through secure channels, respectively. When $\mathsf{pw}' = \mathsf{pw}$, the client is able to retrieve the secret s by combining t partial decryptions. This protocol is secure against honest-but-curious adversaries but not malicious adversaries. A protocol against malicious adversaries was also given by Bagherzandi et al. [2] using non-interactive zero-knowledge proofs. In Bagherzandi et al. protocol, it is easy to see that the client must correctly remember the public key pk and the exact set of servers, as he or she sends out an encryption of his or her password attempt pw' he or she remembers. If pk can be tampered with and changed so that the adversary knows the decryption key, then the adversary can decrypt pw'. Although the protocol actually encrypts $g^{\mathsf{pw}'}$, the malicious servers can perform an offline dictionary attack on $g^{\mathsf{pw}'}$ to obtain the password pw'. In addition, even if the client can correctly remember the public key pk, the malicious servers can cheat the client with a different secret s' by sending the partial decryptions of $B' = E(s', pk)A$ instead of $B = E(s, pk)A$ to the client. An 1-out-of-2 TPASS was given by Camenisch et al. [5] in 2012. This protocol also leaks the password when the client tries to retrieve his or her secret from a set of all-malicious servers.

Authenticating to the wrong servers is a common scenario when users are tricked in phishing attacks. To overcome this shortcoming, Camenisch et al. [6] proposed the first t-out-of-n TPASS protocol for any $n > t$ in 2014. The protocol requires the client to only remember a username and a password, assuming that a PKI is available. If the client misremembers his or her list of servers and tries to retrieve his or her secret from corrupt servers, the protocol prevents the servers from learning anything about the password or secret, as well as from planting a different secret into the user's mind than the secret that he or she stored earlier. The construction of Camenisch et al. protocol is inspired by Bagherzandi et al. protocol based on a homomorphic threshold encryption scheme, but the crucial difference is that in the retrieval protocol of Camenisch et al., the client never sends out an encryption of his or her password attempt. Instead, the client derives an encryption of the (randomised) quotient of the password used at setup and the password attempt. The servers then jointly decrypt the quotient and verify whether it yields "1", indicating that both passwords matched. In case the passwords were not the same, all the servers learn is a random value.

Camenisch et al. TPASS protocol [6], proved to be secure in the UC framework, requires the client to involve in many communication rounds so that it becomes impractical for the client. The client has to do $5n+15$ exponentiations in $\mathbb{G}$ for the setup protocol and $14t + 24$ exponentiations in the retrieval protocol. Each server has to perform $n + 18$ and $7t + 28$ exponentiations in these respective protocols.

Recently, Abdalla et al. [1] proposed new robust password-protected secret sharing protocols which are significantly more efficient than the existing ones. Their protocols have been proven in the random-oracle model, because their construction requires random non-malleable fingerprints, which is provided by an ideal hash function. In addition, Jarecki et al. [11] constructed password-protected secret sharing protocols based on oblivious pseudorandom functions, formulated as a universally composable (UC) functionality.

Our Contribution. To improve the efficiency of TPASS, we propose two new t-out-of-n TPASS protocols for any $n > t$ in this paper. One protocol is built on two-phase commitment and has lower computation complexity. Another protocol is built on zero-knowledge proof and has less communication rounds. Both protocols are in particular efficient for the client, who only needs to send a request and receive a response.

The basic idea is as follows: a client initially secret-shares a password, a secret and the digest of the secret with n servers, such as t out of the n servers can recover the secret. When retrieving the secret from the servers, the client submits to the servers $A = g_1^r g_2^{\mathsf{pw}_C}$, where r is randomly chosen and pw_C is the password, and then t servers cooperate to generate and return an ElGamal encryption of the secret and an ElGamal encryption of the digest of the secret, both under the public key g_1^r. We use two-phase commitment and zero-knowledge proof to prevent the collusion attack from up to $t-1$ malicious servers. At the end, the client then decrypts the two ciphertexts and accepts the secret if one decrypted value is another's digest.

The proposed protocols are significantly more efficient than Camenisch et al. protocol [6] in terms of computational and communication complexities. In the proposed protocols, the client only needs to send a request and receive a response. In addition, the client needs to do $3n$ evaluations of polynomials of degree $t-1$ in $\mathbb{Z}_q$ for the initialization and 7 exponentiations for the retrieval protocol. Each server only needs to do $t+10$ (in two-phase commitment case) or $3t+10$ (in one-phase zero-knowledge proof case) exponentiations in the retrieval protocol. The computation and communication complexities for the client are independent of the number of the servers n and the threshold t.

We have provided rigorous proof of security for the proposed protocols in the standard model. Like Camenisch et al. protocol [6], the proposed protocols can protect the password of the client even if he or she communicates with all-malicious servers by mistake. In addition, they prevent the servers from planting a different secret into the user's mind than the secret that he stored earlier.

This paper is an extended version of our conference paper [20]. In this extension, we have added a new TPASS protocol based on zero-knowledge proof, which reduces the communications among servers from two phases to one phase.

In addition, we provide a rigorous proof of security for the new protocol in the standard model, and performed some experiments on the proposed two protocols.

Related Works. A close work related to TPASS is threshold password - authenticated key exchange (TPAKE), which lets the client agree on a fresh session key with each of the servers, but does not allow the client to store and recover a secret. Depending on the desired security properties, one can build a TPASS scheme from a TPAKE scheme by using the agreed-upon session keys to transmit the stored secret shares over secure channels [2].

The first TPAKE protocols, due to Ford and Kaliski [9] and Jablon [10], were not proved secure. The first provably secure TPAKE protocol, a t-out-of-n protocol in a PKI setting, was proposed by MacKenzie et al. [14]. The 1-out-of-2 protocol of Brainard et al. [4] is implemented in EMC's RSA Distributed Credential Protection. Both protocols either leak the password or allow an offline dictionary attack when the retrieval is performed with corrupt servers. The t-out-of-n TPAKE protocols by Di Raimondo and Gennaro [15] and the 1-out-of-2 protocol by Katz et al. [13] are proved secure in a hybrid password-only/PKI setting, where the user does not know any public keys, but the servers and an intermediate gateway do have a PKI. These protocols actually remain secure when executed with all-corrupt servers, but are restricted to the cases that $n > 3t$ and $(t, n) = (1, 2)$. Based on identity-based encryption (IBE), an 1-out-of-2 protocol where the client is required to remember the identities of the two servers besides his or her password, was proposed by Yi et al. [19].

Organization. The rest of our paper is organised as follows. We give the security definition in Section 2, describe the proposed TPASS protocols in Section 3, provide the security proof for the proposed protocols in Section 4, analyze the performance of the proposed protocols in Section 5. Conclusions are drawn in the last section.

2. Definition of Security

In this section, we define the security for TPASS protocol on the basis of the security models for PAKE [3, 12].

Participants, Initialization, Passwords, Secrets. A TPASS protocol involves three kinds of protocol participants: (1) A group of clients (denoted as Client), each of which requests TPASS services from t servers on the network; (2) A group of n servers $\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_n$ (denoted as $\mathsf{Server} = \{\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_n\}$), which cooperate to provide TPASS services to clients on the network; (3) A gateway (GW), which coordinates TPASS. We assume that $\mathsf{User} = \mathsf{Client} \bigcup \mathsf{Server}$ and $\mathsf{Client} \bigcap \mathsf{Server} = \emptyset$. When the gateway GW coordinates TPASS, it simply forwards messages between a client and t servers.

Prior to any execution of the protocol, we assume that an initialization phase occurs. During initialization, the n servers cooperate to generate public parameters for the protocol, which are available to all participants.

We assume that the client C chooses its password pw_C independently and uniformly at random from a "dictionary" $\mathsf{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ of size N, where N is a fixed constant which is independent of any security parameter. The client then secretly shares the password with the n servers such that any t servers can restore the password.

In addition, we assume that the client C chooses its secret s_C independently and uniformly at random from $\mathbb{Z}_q^*$, where q is a public parameter. The client then secretly shares the secret with the n servers such that any t servers can recover the secret.

We assume that at least $n - t + 1$ servers are trusted not to collude to determine the password and the secret of the client. The client C needs to remember pw_C only to retrieve its secret s_C.

Execution of the Protocol. A protocol determines how users behave in response to input from their environments. In the formal model, these inputs are provided by the adversary. Each user is assumed to be able to execute the protocol multiple times (possibly concurrently) with different partners. This is modeled by allowing each user to have unlimited number of instances with which to execute the protocol. We denote instance i of user U as U^i. A given instance may be used only once. The adversary is given oracle access to these different instances. Furthermore, each instance maintains (local) state which is updated during the course of the experiment. In particular, each instance U^i is associated with the following variables, initialized as NULL or FALSE (as appropriate) during the initialization phase.

- sid_U^i is a variable containing the session identity for an instance U^i. The session identity is simply a way to keep track of the different executions of a particular user U. Without loss of generality, we simply let this be the (ordered) concatenation of all messages sent and received by instance U^i.
- s_C^i is a variable containing the secret s_C for a client instance C^i. Retrieval of the secret is, of course, the ultimate goal of the protocol.
- acc_U^i and term_U^i are boolean variables denoting whether a given instance U^i has been accepted or terminated, respectively. Termination means that the given instance has done receiving and sending messages, acceptance indicates successful termination. When an instance U^i has been accepted, sid_U^i is no longer NULL. When a client instance C^i has been accepted, s_C^i is no longer NULL.
- state_U^i records any state necessary for execution of the protocol by U^i.
- used_U^i is a boolean variable denoting whether an instance U^i has begun executing the protocol. This is a formalism which will ensure each instance is used only once.

The adversary $\mathcal{A}$ is assumed to have complete control over all communications in the network (between the clients and servers, and between servers and servers) and the adversary's interaction with the users (more specifically, with various instances) is modelled via access to oracles. The state of an instance may be updated during an oracle call, and the oracle's output may depend upon the relevant instance. The oracle types include:

- $\mathsf{Send}(C, i, M)$ – This sends message M to a client instance C^i. Assuming $\mathsf{term}_C^i = \mathsf{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of C^i (i.e., the message sent by the instance) is given to the adversary, who receives the updated values of $\mathsf{sid}_C^i, \mathsf{acc}_C^i$, and term_C^i. This oracle call models an active attack to the protocol. If M is empty, this query represents a prompt for C to initiate the protocol.

- $\mathsf{Send}(S, j, U, M)$ – This sends message M to a server instance S^j, supposedly from a user U (either a client or a server) or even a set of servers. Assuming $\mathsf{term}_S^j = \mathsf{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of S^j (i.e., the message sent by the instance) is given to the adversary, who receives the updated values of $\mathsf{sid}_S^j, \mathsf{acc}_S^j$, and term_S^j. If S is corrupted, the adversary also receives the entire internal state of S. This oracle call also models an active attack to the protocol.

- $\mathsf{Execute}(C, i, \mathbb{S})$ – If the client instance C^i and t server instances, denoted as $\mathbb{S}$, have not yet been used, this oracle executes the protocol between these instances and outputs the transcript of this execution. This oracle call represents passive eavesdropping of a protocol execution. In addition to the transcript, the adversary receives the values of sid, acc, and term for client and server instances, at each step of protocol execution. In addition, if any server in $\mathbb{S}$ is corrupted, the adversary is given the entire internal state of the server.

- $\mathsf{Corrupt}(S)$ – This sends the password and secret shares of all clients stored in the server S to the adversary. This oracle models possible compromising of a server due to, for example, hacking into the server.

- $\mathsf{Corrupt}(C)$ – This query allows the adversary to learn the password of the client C and then the secret of the client, which models the possibility of subverting a client by, for example, witnessing a user typing in his password, or installing a "Trojan horse" on his machine.

- $\mathsf{Test}(C, i)$ – This oracle does not model any real-world capability of the adversary, but is instead used to define security. If $\mathsf{acc}_C^i = \mathsf{TRUE}$, a random bit b is generated. If $b = 0$, the adversary is given s_C^i, and if $b = 1$ the adversary is given a random number. The adversary is allowed only a single Test query, at any time during its execution.

Correctness. To be viable, a TPASS protocol must satisfy the following notion of correctness: If a client instance C^i and t server instances $\mathbb{S}$ runs an honest execution of the protocol with no interference from the adversary, then $\mathsf{acc}_C^i = \mathsf{acc}_S^j = \mathsf{TRUE}$ for any server instance S^j in $\mathbb{S}$.

Freshness. To formally define the adversary's success we need to define a notion of freshness for a client, where freshness of the client is meant to indicate that the adversary does not trivially know the value of the secret of the client. We say a client instance C^i is fresh if (1) C has not been corrupted; (2) $\mathsf{Test}(C)$ has not been queried; and (3) at least $n-t+1$ out of n servers are not corrupted.

Advantage of the Adversary. We consider passive and active attacks, respectively. In a passive attack, the adversary is allowed to call $\mathsf{Execute}$, $\mathsf{Corrupt}$ and Test oracles. Informally, a passive adversary succeeds if it can guess the bit b used by the Test oracle. We say a passive adversary $\mathcal{A}$ succeeds if it makes a query $\mathsf{Test}(C, i)$ to a fresh client instance C^i, with $\mathsf{acc}_C^i = \mathsf{TRUE}$ at the time of this query, and outputs a bit b' with $b' = b$ (recall that b is the bit chosen by the Test oracle). We denote this event by $\mathsf{Succ_P}$. The advantage of a passive adversary $\mathcal{A}$ in attacking protocol P is then given by

$$\mathsf{Adv}_{\mathcal{P}\mathcal{A}}^{P}(k) = 2 \cdot \mathsf{Pr}[\mathsf{Succ_P}] - 1$$

where the probability is taken over the random coins used by the adversary and the random coins used during the course of the experiment (including the initialization phase).

Definition 1. Protocol P is a secure TPASS protocol against the passive attack, if, for all passive PPT adversaries $\mathcal{A}$, there exists a negligible function $\varepsilon(\cdot)$ such that for a security parameter k,

$$\mathsf{Adv}^{P}_{\mathcal{PA}}(k) \leq \varepsilon(k)$$

In an active attack, the adversary is allowed to call Send and $\mathsf{Corrupt}$ oracles. Informally, an active adversary succeeds if it can convince a client to accept a wrong secret key. We say an active adversary $\mathcal{A}$ succeeds if it makes an query $\mathsf{Send}(C, i)$ to a fresh client instance C^i, resulting in $\mathsf{acc}^i_C = \mathsf{TRUE}$. We denote this event by $\mathsf{Succ_A}$. The advantage of an active adversary $\mathcal{A}$ in attacking protocol P is then given by

$$\mathsf{Adv}^{P}_{\mathcal{AA}}(k) = \mathsf{Pr}[\mathsf{Succ_A}]$$

where the probability is taken over the random coins used by the adversary and the random coins used during the course of the experiment (including the initialization phase).

The active adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack. A protocol is secure against the active attack if this is the best an adversary can do. The on-line attacks correspond to Send queries. Formally, each instance for which the adversary has made a Send query counts as one on-line attack. The number of on-line attacks represents a bound on the number of passwords the adversary could have tested in an on-line fashion.

Definition 2. Protocol P is a secure TPASS protocol against the active attack if, for all dictionary size N and for all active PPT adversaries $\mathcal{A}$ making at most $Q(k)$ on-line attacks, there exists a negligible function $\varepsilon(\cdot)$ such that for a security parameter k,

$$\mathsf{Adv}^{P}_{\mathcal{AA}}(k) \leq Q(k)/N + \varepsilon(k)$$

3. The Proposed TPASS Protocols

In this section, we describe two TPASS protocols based on two-phase commitment protocol and zero-knowledge proof, respectively.

Initialization. Given a security parameter $k \in \mathbb{Z}^*$, the initialization includes:

Parameter Generation: On input k, the n servers agree on a cyclic group $\mathbb{G}$ of large prime order q with a generator g_1 and a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$. Then the n servers cooperate to generate g_2, like [18], such that no one knows the discrete logarithm of g_2 based on g_1 if one out of the n server is honest. The public parameters for the protocol is $\mathsf{params} = \{\mathbb{G}, q, g_1, g_2, H\}$.

Password Generation: On input params, each client $C \in \mathsf{Client}$ with identity ID_C uniformly draws a string pw_C, the password, from the dictionary $\mathsf{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$. The client then randomly chooses a polynomial $f_1(x)$ of degree $t-1$ over $\mathbb{Z}_q$ such that $\mathsf{pw}_C = f_1(0)$, and distributes $\{ID_C, i, f_1(i)\}$ to the server S_i via a secure channel, where $i = 1, 2, \cdots, n$.

Secret Sharing: On input params, each client $C \in \mathsf{Client}$ randomly chooses s from $\mathbb{Z}_q^*$. The client then randomly chooses two polynomials $f_2(x)$ and $f_3(x)$ of degree $t-1$ over $\mathbb{Z}_q$ such that $s = f_2(0)$ and $H(g_2^s) = f_3(0)$, and distributes $\{ID_C, i, f_2(i), f_3(i)\}$ to the server S_i via a secure channel, where $i = 1, 2, \cdots, n$. We define the secret s_C as g_2^s.

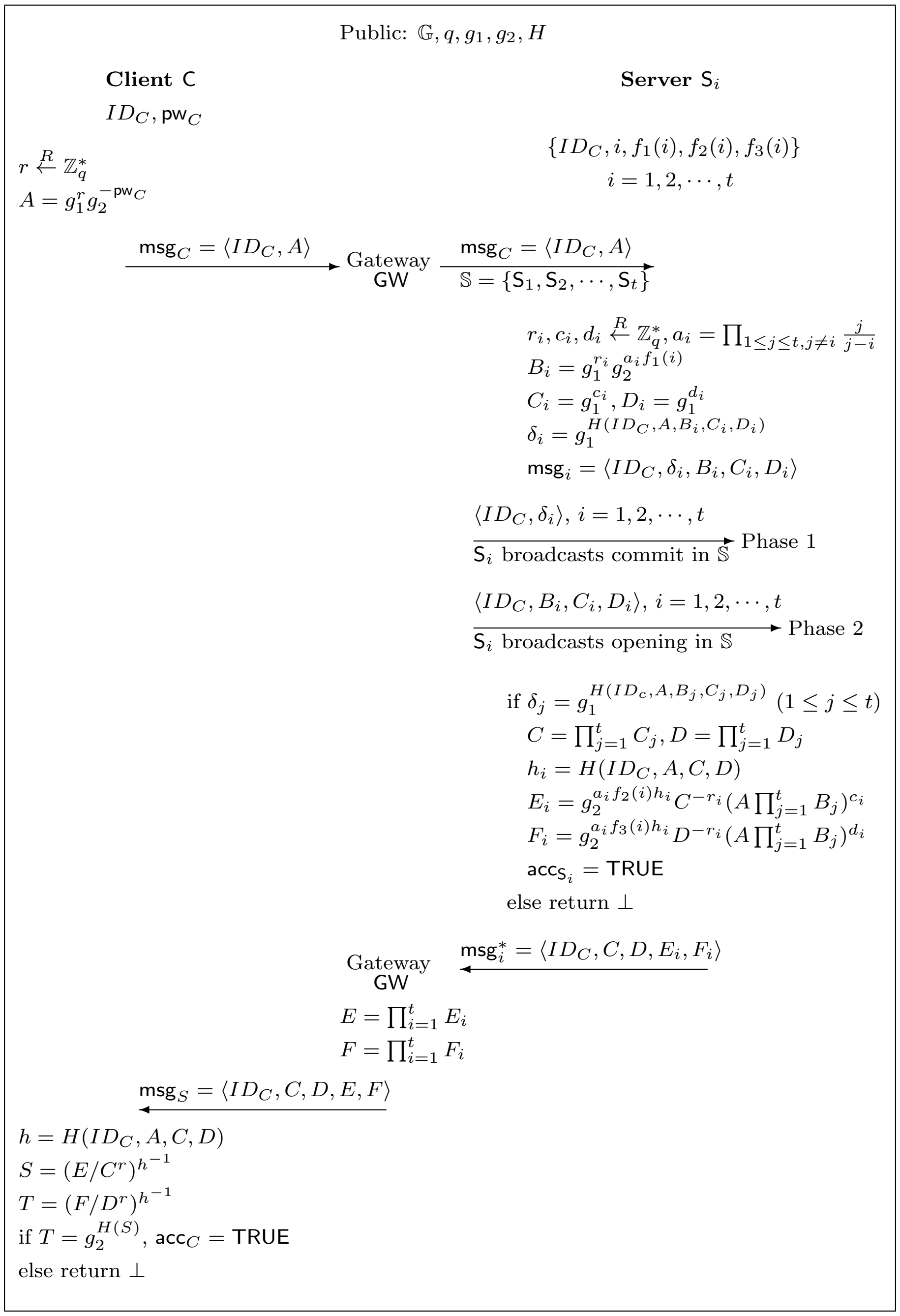
Protocol Execution. Given the public $\mathsf{params} = \{\mathbb{G}, q, g_1, g_2, H\}$, the client C (knowing its identity ID_C and password pw_C) runs TPASS protocol P with t servers (each server knowing $\{ID_C, i, f_1(i), f_2(i), f_3(i)\}$) to retrieve the secret s_C as shown in Fig. 2.

In Fig. 2, TPASS protocol is executed with three algorithms as follows.

Retrieval Request. Given the public parameters $\{\mathbb{G}, g_1, g_2, q, H\}$, the client C with the identity ID_C validates if q is a large prime and $g_1^q = g_2^q = 1$. If so, the client, who remembers the password pw_C, randomly chooses r from $\mathbb{Z}_q^*$ and computes

$$A = g_1^r g_2^{-\mathsf{pw}_C}.$$

Then the client submits $\mathsf{msg}_C = \langle ID_C, A\rangle$ to the gateway GW for the n servers.



Public: $\mathbb{G}, q, g_1, g_2, H$

Client C

ID_C, pw_C

Server S_i

$\{ID_C, i, f_1(i), f_2(i), f_3(i)\}$

$i = 1, 2, \cdots, t$

$r \overset{R}{\leftarrow} \mathbb{Z}_q^*$

$A = g_1^r g_2^{-\mathsf{pw}_C}$

$\xrightarrow{\mathsf{msg}_C = \langle ID_C, A\rangle}$ Gateway GW $\xrightarrow[\mathbb{S} = \{\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_t\}]{\mathsf{msg}_C = \langle ID_C, A\rangle}$

$r_i, c_i, d_i \overset{R}{\leftarrow} \mathbb{Z}_q^*, a_i = \prod_{1\le j\le t, j\ne i} \frac{j}{j-i}$

$B_i = g_1^{r_i} g_2^{a_i f_1(i)}$

$C_i = g_1^{c_i}, D_i = g_1^{d_i}$

$\delta_i = g_1^{H(ID_C, A, B_i, C_i, D_i)}$

$\mathsf{msg}_i = \langle ID_C, \delta_i, B_i, C_i, D_i\rangle$

$\xrightarrow[\mathsf{S}_i \text{ broadcasts commit in } \mathbb{S}]{\langle ID_C, \delta_i\rangle,\ i = 1, 2, \cdots, t}$ Phase 1

$\xrightarrow[\mathsf{S}_i \text{ broadcasts opening in } \mathbb{S}]{\langle ID_C, B_i, C_i, D_i\rangle,\ i = 1, 2, \cdots, t}$ Phase 2

if $\delta_j = g_1^{H(ID_c, A, B_j, C_j, D_j)}$ $(1 \le j \le t)$

$C = \prod_{j=1}^t C_j, D = \prod_{j=1}^t D_j$

$h_i = H(ID_C, A, C, D)$

$E_i = g_2^{a_i f_2(i) h_i} C^{-r_i} (A \prod_{j=1}^t B_j)^{c_i}$

$F_i = g_2^{a_i f_3(i) h_i} D^{-r_i} (A \prod_{j=1}^t B_j)^{d_i}$

$\mathsf{acc}_{\mathsf{S}_i} = \mathsf{TRUE}$

else return $\perp$

Gateway GW $\xleftarrow{\mathsf{msg}_i^* = \langle ID_C, C, D, E_i, F_i\rangle}$

$E = \prod_{i=1}^t E_i$

$F = \prod_{i=1}^t F_i$

$\xleftarrow{\mathsf{msg}_S = \langle ID_C, C, D, E, F\rangle}$

$h = H(ID_C, A, C, D)$

$S = (E/C^r)^{h^{-1}}$

$T = (F/D^r)^{h^{-1}}$

if $T = g_2^{H(S)}$, $\mathsf{acc}_C = \mathsf{TRUE}$

else return $\perp$

Figure 2: The Proposed TPASS Protocol P Based on Two-Phase Commitment

Remark. The purpose for the client to validate the public parameters is to ensure that the discrete logarithm over $\{\mathbb{G}, q, g_1, g_2\}$ is hard in case that the adversary can change the public parameters.

Retrieval Response. After receiving the request msg_C from the client C, the gateway GW forwards it to t available servers to response the request. Without loss of generality, we assume that the first t servers, denoted as $\mathbb{S} = \{\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_t\}$, cooperate to generate a response. There are two phases for the t servers to generate retrieval response.

Commitment Phase. Based on the identity ID_C of the client, each server S_i $(i = 1, 2, \cdots, t)$ randomly chooses r_i, c_i, d_i from $\mathbb{Z}_q^*$ and computes

$$B_i = g_1^{r_i} g_2^{a_i f_1(i)}, C_i = g_1^{c_i}, D_i = g_1^{d_i}, \delta_i = g_1^{H(ID_C, A, B_i, C_i, D_i)}$$

where $a_i = \prod_{1 \le j \le t, j \ne i} \frac{j}{j-i}$.

In the commitment phase, S_i broadcasts its commitment $\langle ID_C, \delta_i \rangle$.

Opening Phase. After receiving all commitments $\langle ID_C, \delta_j \rangle$ $(1 \le j \le t)$, S_i broadcasts its opening $\langle ID_C, B_i, C_i, D_i \rangle$.

Each server S_i verifies if $\delta_j = g_1^{H(ID_C, A, B_j, C_j, D_j)}$ for all $j \ne i$. If so, based on the identity ID_C of the client, S_i computes

$$C = \prod_{j=1}^{t} C_j, D = \prod_{j=1}^{t} D_j, h_i = H(ID_C, A, C, D)$$

$$E_i = g_2^{a_i f_2(i) h_i} C^{-r_i} (A \prod_{j=1}^{t} B_j)^{c_i}, F_i = g_2^{a_i f_3(i) h_i} D^{-r_i} (A \prod_{j=1}^{t} B_j)^{d_i}$$

and sets $\mathsf{acc}_{\mathsf{S}_i} = \mathsf{TRUE}$.

Then S_i sends $\mathsf{msg}_i^* = \{ID_C, C, D, E_i, F_i\}$ to the gateway GW.

The gateway GW computes

$$E = \prod_{i=1}^{t} E_i, F = \prod_{i=1}^{t} F_i$$

and returns to the client with $\mathsf{msg}_S = \{ID_C, C, D, E, F\}$.

Secret Retrieval. After receiving the response $\mathsf{msg}_S = \{ID_C, C, D, E, F\}$ from the gateway, the client computes

$$h = H(ID_C, A, C, D), S = (E/C^r)^{h^{-1}}, T = (F/D^r)^{h^{-1}}$$

and verifies if $T = g_2^{H(S)}$. If so, the client sets $\mathsf{acc}_C = \mathsf{TRUE}$, the secret $s_C = S$ and $\perp$ otherwise.

Remark. In the end of the protocol, the servers can check if the password attempt is correct or not by one more step, where the servers provide the client with a ciphertext for decryption, assuming that the client initially stores a pair of plaintext and ciphertext in the servers. If the client can return a decrypted result same as the plaintext stored in the servers previously, the password attempt is correct. Otherwise, the servers should block the account after several failed attempts. In this way, the proposed protocol can prevent the Distributed Denial-of-Service (DDoS) attack efficiently and effectively.

3.2. The Proposed Protocol Based on Zero-Knowledge Proof

Initialization. The initialization is the same as described in Section 3.1.

Protocol Execution. Given the public $\mathsf{params} = \{\mathbb{G}, q, g_1, g_2, H\}$, the client C (knowing its identity ID_C and password pw_C) runs TPASS protocol P with t servers (each server knowing $\{ID, i, f_1(i), f_2(i), f_3(i)\}$) to retrieve the secret s_C as shown in Fig. 3.

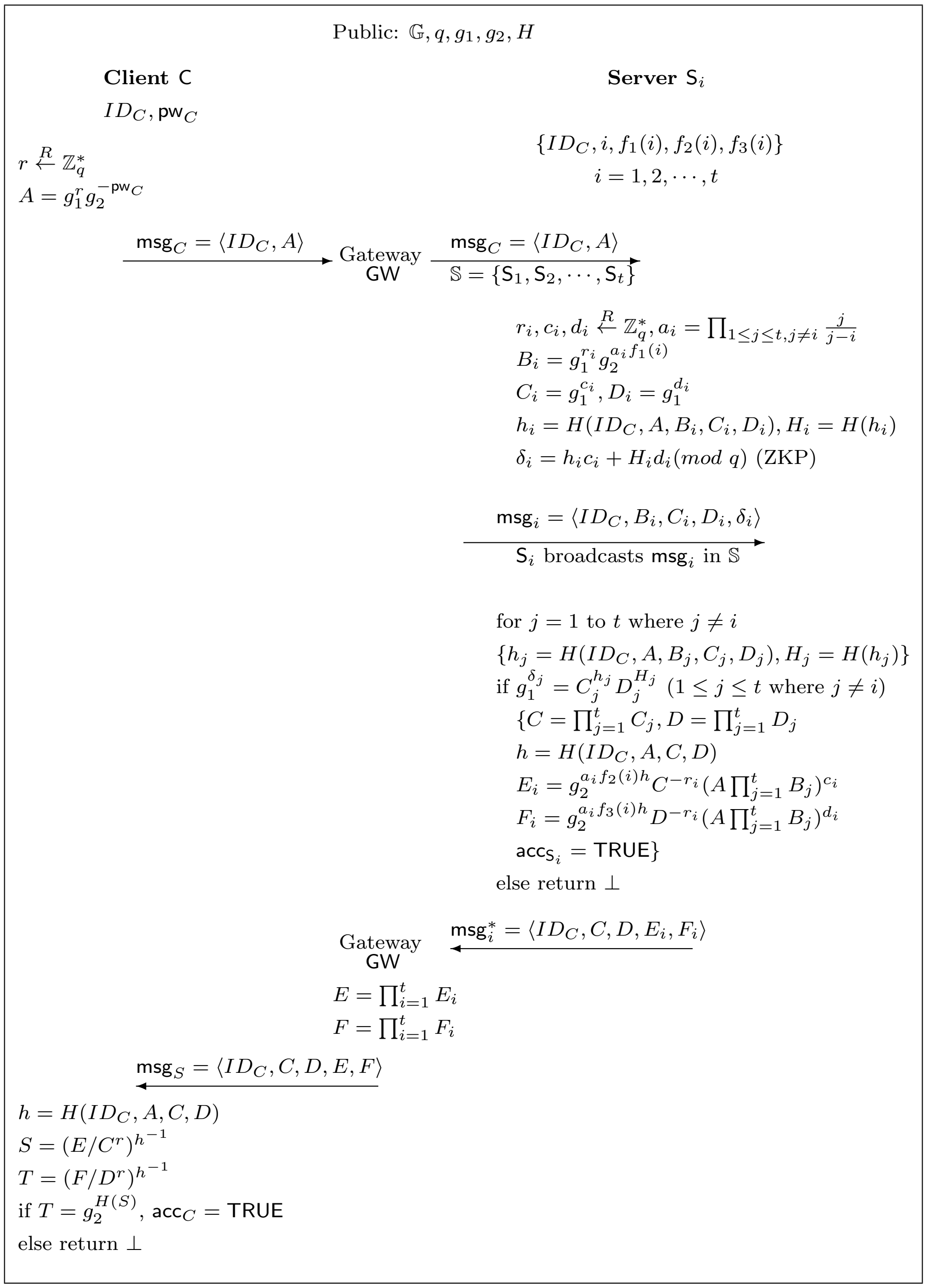
In Fig. 3, TPASS protocol is executed with three algorithms as follows.

Retrieval Request. Given the public parameters $\{\mathbb{G}, g_1, g_2, q, H\}$, the client C with the identity ID_C validates if q is a large prime and $g_1^q = g_2^q = 1$. If so, the client, who remembers the password pw_C, randomly chooses r from $\mathbb{Z}_q^*$ and computes

$$A = g_1^r g_2^{-\mathsf{pw}_C}.$$

Then the client submits $\mathsf{msg}_C = \langle ID_C, A\rangle$ to the gateway GW for the n servers.

Retrieval Response. After receiving the request msg_C from the client C, the gateway GW forwards it to t available servers to respond the request. With-



Public: $\mathbb{G}, q, g_1, g_2, H$

Client C

ID_C, pw_C

$r \overset{R}{\leftarrow} \mathbb{Z}_q^*$

$A = g_1^r g_2^{-\mathsf{pw}_C}$

Server S_i

$\{ID_C, i, f_1(i), f_2(i), f_3(i)\}$

$i = 1, 2, \cdots, t$

$\xrightarrow{\mathsf{msg}_C = \langle ID_C, A \rangle}$ Gateway GW $\xrightarrow[\mathbb{S} = \{\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_t\}]{\mathsf{msg}_C = \langle ID_C, A \rangle}$

$r_i, c_i, d_i \overset{R}{\leftarrow} \mathbb{Z}_q^*, a_i = \prod_{1 \le j \le t, j \ne i} \frac{j}{j-i}$

$B_i = g_1^{r_i} g_2^{a_i f_1(i)}$

$C_i = g_1^{c_i}, D_i = g_1^{d_i}$

$h_i = H(ID_C, A, B_i, C_i, D_i), H_i = H(h_i)$

$\delta_i = h_i c_i + H_i d_i (mod\ q)$ (ZKP)

$\xrightarrow[\mathsf{S}_i \text{ broadcasts } \mathsf{msg}_i \text{ in } \mathbb{S}]{\mathsf{msg}_i = \langle ID_C, B_i, C_i, D_i, \delta_i \rangle}$

for $j = 1$ to t where $j \ne i$

$\{h_j = H(ID_C, A, B_j, C_j, D_j), H_j = H(h_j)\}$

if $g_1^{\delta_j} = C_j^{h_j} D_j^{H_j}$ $(1 \le j \le t$ where $j \ne i)$

  $\{C = \prod_{j=1}^t C_j, D = \prod_{j=1}^t D_j$

  $h = H(ID_C, A, C, D)$

  $E_i = g_2^{a_i f_2(i) h} C^{-r_i} (A \prod_{j=1}^t B_j)^{c_i}$

  $F_i = g_2^{a_i f_3(i) h} D^{-r_i} (A \prod_{j=1}^t B_j)^{d_i}$

  $\mathsf{acc}_{\mathsf{S}_i} = \mathsf{TRUE}\}$

else return $\perp$

Gateway GW $\xleftarrow{\mathsf{msg}_i^* = \langle ID_C, C, D, E_i, F_i \rangle}$

$E = \prod_{i=1}^t E_i$

$F = \prod_{i=1}^t F_i$

$\xleftarrow{\mathsf{msg}_S = \langle ID_C, C, D, E, F \rangle}$

$h = H(ID_C, A, C, D)$

$S = (E/C^r)^{h^{-1}}$

$T = (F/D^r)^{h^{-1}}$

if $T = g_2^{H(S)}$, $\mathsf{acc}_C = \mathsf{TRUE}$

else return $\perp$

Figure 3: The Proposed TPASS Protocol P Based on Zero-Knowledge Proof

out loss of generality, we assume that the first t servers, denoted as $\mathbb{S} = \{\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_t\}$, cooperate to generate a response. Unlike the proposed protocol based on two-phase commitment, the t servers can generate the response in one phase.

Based on the identity ID_C of the client, each server S_i $(i = 1, 2, \cdots, t)$ randomly chooses r_i, c_i, d_i from $\mathbb{Z}_q^*$ and computes

$$B_i = g_1^{r_i} g_2^{a_i f_1(i)}, C_i = g_1^{c_i}, D_i = g_1^{d_i},$$

$$h_i = H(ID_C, A, B_i, C_i, D_i), H_i = H(h_i), \delta_i = h_i c_i + H_i d_i (mod\ q),$$

where $a_i = \prod_{1 \leq j \leq t, j \neq i} \frac{j}{j-i}$, (C_i, D_i, δ_i) is a zero-knowledge proof of knowledge of (c_i, d_i).

Then S_i broadcasts $\mathsf{msg}_i = \langle ID_C, B_i, C_i, D_i, \delta_i \rangle$ in $\mathbb{S}$.

Each server S_i computes

$$h_j = H(ID_C, A, B_j, C_j, D_j), H_j = H(h_j)$$

for all $j \neq i$ and verifies the zero-knowledge proof of knowledge of (c_j, d_j) by checking if

$$g_1^{\delta_j} = C_j^{h_j} D_j^{H_j}$$

for all $j \neq i$. If so, based on the identity ID_C of the client, S_i computes

$$C = \prod_{j=1}^{t} C_j, D = \prod_{j=1}^{t} D_j, h = H(ID_C, A, C, D)$$

$$E_i = g_2^{a_i f_2(i) h} C^{-r_i} (A \prod_{j=1}^{t} B_j)^{c_i}, F_i = g_2^{a_i f_3(i) h} D^{-r_i} (A \prod_{j=1}^{t} B_j)^{d_i}$$

and sets $\mathsf{acc}_{\mathsf{S}_i} = \mathsf{TRUE}$.

Then S_i sends $\mathsf{msg}_i^* = \{ID_C, C, D, E_i, F_i\}$ to the gateway GW.

The gateway GW computes

$$E = \prod_{i=1}^{t} E_i, F = \prod_{i=1}^{t} F_i$$

and returns to the client with $\mathsf{msg}_S = \{ID_C, C, D, E, F\}$.

Secret Retrieval. After receiving the response $\mathsf{msg}_S = \{ID_C, C, D, E, F\}$ from the gateway, the client computes

$$h = H(ID_C, A, C, D), S = (E/C^r)^{h^{-1}}, T = (F/D^r)^{h^{-1}}$$

and verifies if $T = g_2^{H(S)}$. If so, the client sets $\mathsf{acc}_C = \mathsf{TRUE}$, the secret $s_C = S$ and $\perp$ otherwise.

3.3. Correctness

330 **Correctness of the Proposed TPASS protocol based on two-phase commitment.** Assume that a client instance C^i and t server instances $\mathbb{S}$ run an honest execution of the proposed TPASS protocol P with no interference from the adversary. With reference to Fig. 2, it is obvious that $\mathsf{acc}_{\mathsf{S}_j} = \mathsf{TRUE}$ for $1 \leq j \leq t$. In addition, we have

$$\begin{aligned}
C &= \prod_{j=1}^{t} C_j = g_1^{\sum_{j=1}^{t} c_j}, D = \prod_{j=1}^{t} D_j = g_1^{\sum_{j=1}^{t} d_j} \\
E_i &= g_2^{a_i f_2(i) h_i} C^{-r_i} (A \prod_{j=1}^{t} B_j)^{c_i} \\
&= g_2^{a_i f_2(i) h_i} g_1^{-r_i \sum_{j=1}^{t} c_j} (g_1^r g_2^{-\mathsf{pw}_C} g_1^{\sum_{j=1}^{t} r_j} g_2^{\mathsf{pw}_C})^{c_i} \\
&= g_2^{a_i f_2(i) h_i} g_1^{-r_i \sum_{j=1}^{t} c_j} g_1^{c_i \sum_{j=1}^{t} r_j} g_1^{c_i r} \\
F_i &= g_2^{a_i f_3(i) h_i} D^{-r_i} (A \prod_{j=1}^{t} B_j)^{d_i} \\
&= g_2^{a_i f_3(i) h_i} g_1^{-r_i \sum_{j=1}^{t} d_j} (g_1^r g_2^{-\mathsf{pw}_C} g_1^{\sum_{j=1}^{t} r_j} g_2^{\mathsf{pw}_C})^{d_i} \\
&= g_2^{a_i f_3(i) h_i} g_1^{-r_i \sum_{j=1}^{t} d_j} g_1^{d_i \sum_{j=1}^{t} r_j} g_1^{d_i r} \\
h &= H(ID_C, A, C, D) \\
E &= \prod_{i=1}^{t} E_i = \prod_{i=1}^{t} g_2^{a_i f_2(i) h} g_1^{-r_i \sum_{j=1}^{t} c_j} g_1^{c_i \sum_{j=1}^{t} r_j} g_1^{c_i r} \\
&= g_2^{sh} g_1^{-\sum_{i=1}^{t} r_i \sum_{j=1}^{t} c_j} g_1^{\sum_{i=1}^{t} c_i \sum_{j=1}^{t} r_j} g_1^{r \sum_{i=1}^{t} c_i} = g_2^{sh} C^r \\
F &= \prod_{i=1}^{t} F_i = \prod_{i=1}^{t} g_2^{a_i f_3(i) h} g_1^{-r_i \sum_{j=1}^{t} d_j} g_1^{d_i \sum_{j=1}^{t} r_j} g_1^{d_i r} \\
&= g_2^{H(g_2^s) h} g_1^{-\sum_{i=1}^{t} r_i \sum_{j=1}^{t} d_j} g_1^{\sum_{i=1}^{t} d_i \sum_{j=1}^{t} r_j} g_1^{r \sum_{i=1}^{t} d_i} = g_2^{H(g_2^s) h} D^r
\end{aligned}$$

We can see that (C, E) and (D, F) are in fact the EGamal encryptions of g_2^{sh} and $g_2^{H(g^s)h}$ under the public key g_1^r, respectively. Therefore, we have $\mathsf{acc}_C = \mathsf{TRUE}$ because

$$\begin{aligned} h &= H(ID_C, A, C, D) \\ S &= (E/C^r)^{h^{-1}} = (g_2^{sh})^{h^{-1}} = g_2^s \\ T &= (F/D^r)^{h^{-1}} = (g_2^{H(g_2^s)h})^{h^{-1}} = g_2^{H(g_2^s)} \\ T &= g_2^{H(S)}. \end{aligned}$$

In summary, the proposed TPASS protocol based on two-phase commitment has correctness.

Correctness of the proposed TPASS protocol based on zero-knowledge proof. Comparing the proposed two protocols shown in Fig. 2 and Fig. 3, we can see that there are two differences: (1) δ_i is computed differently; (2) δ_i is verified differently. Therefore, we only need to show that $g_1^{\delta_j} = C_j^{h_j} D_j^{H_j}$ for $j = 1, 2, \cdots, t$ for the second proposed protocol if each server S_i follows the protocol to compute B_i, C_i, D_i and δ_i.

Because $C_j = g_1^{c_j}, D_j = g_1^{d_j}, \delta_j = h_j c_j + H_j d_j$ for $j = 1, 2, \cdots, t$, we have

$$g_1^{\delta_j} = g_1^{h_j c_j + H_j d_j} = (g_1^{c_j})^{h_j} (g_1^{d_j})^{H_j} = C_j^{h_j} D_j^{H_j}$$

for $j = 1, 2, \cdots, t$. Therefore, the proposed TPASS protocol based zero-knowledge proof has correctness.

3.4. Efficiency

Efficiency of the proposed TPASS protocol based on two-phase commitment. In the first proposed protocol, the client needs to compute 7 exponentiations in $\mathbb{G}$ and send or receive 5 group elements in $\mathbb{G}$. Each server needs to compute $t+10$ exponentiations in $\mathbb{G}$ and send or receive $4t+5$ group elements in $\mathbb{G}$.

The client involves only two communication rounds with the gateway, i.e., sending msg_C to the gateway and receiving msg_S from the gateway. Each server

S_i participates in six communication rounds with other servers and the gateway, i.e., receiving msg_C from the gateway, broadcasting the commitment $\langle ID_C, \delta_i \rangle$ to other servers, receiving $\langle ID_C, \delta_j \rangle$ for all $j \neq i$ from other servers, broadcasting $\langle ID_C, B_i, C_i, D_i \rangle$, receiving $\langle ID_C, B_j, C_j, D_j \rangle$ for all $j \neq i$, and finally sending msg_i^* to the gateway.

The performance comparison of Camenisch et al. protocol [6] (with provable security in the UC framework) and the proposed protocol (with provable security in the standard model) can be shown in Tab. 1.

In Table 1, exp. represent the computation complexity of a modular exponentiation, $|g|$ is the size of a group element in $\mathbb{G}$ and $|q|$ is the size of a group element in $\mathbb{Z}_q$, C, S and G stand for Client, Server and Gateway, respectively. In addition, $pk = \prod epk_i, tpk = \prod tpk_i$. In Camenisch et al. protocol [6], a hash value is counted as half a group element.

In our initialization, the client secret-shares the password, secret and the digest of the secret with the n servers via n secure channels which may be established with PKI. In the setup protocol of Camenisch et al., the client setups the shares with the n servers based on PKI. The proposed retrieval protocol does not rely on PKI, but the retrieval protocol of Camenisch et al. still requires PKI. In view of this, the proposed retrieval protocol can be implemented easier than Camenisch et al. retrieval protocol.

From Table 1, we can see that the proposed retrieval protocol is significantly more efficient than the retrieval protocol of Camenisch et al. not only in communication rounds for client but also in computation and communication complexities. In particular, the performance of the client in the proposed retrieval protocol is independent of the number of the servers and the threshold.

Efficiency of the proposed TPASS protocol based on zero-knowledge proof. The second proposed protocol has the same initialization, client retrieval request and client secret retrieval as the first proposed protocol.

In the retrieval response, each server S_i participates in four communication rounds with other servers and the gateway, i.e., receiving msg_C from the gateway,

Table 1: Performance Comparison of the Camenisch et al. Protocol and the Proposed Protocols

	Camenisch et al. [6]	Protocol 1	Protocol 2
Public Keys	C: username S: S_i: epk_i, spk_i, tpk_i	C: username S S_i: none	
Private Keys	C: pw_C S: S_i: esk_i, ssk_i, tsk_i $E(\mathsf{pw}_C, pk), E(s, pk)$ $E(\mathsf{pw}_C, tpk), E(s, tpk)$	C: pw_C S: S_i: $f_1(i), f_2(i), f_3(i)$ where $\sum a_i f_1(i) = \mathsf{pw}_C, \sum a_i f_2(i) = s$ and $\sum a_i f_3(i) = H(g_2^s)$	
Setup Comp. Complexity	C: $5n + 15$ (exp.) S: $n + 18$ (exp.)	C: $3n$ polynomial evaluations S: none	
Setup Comm. Complexity	$n(2.5n + 18.5)\|g\|$	C: $3n\|q\|$ S: $3\|q\|$	
Setup Comm. Round	4	C: 1 S: 1	
Retrieve Comp. Complexity	C: $14t + 24$ (exp.) S: $7t + 28$ (exp.)	C: 7 (exp.) S: $t + 10$ (exp.) G: 0 (exp.)	C: 7 (exp.) S: $3t + 10$ (exp.) G: 0 (exp.)
Retrieve Comm. Complexity	$(t + 1)(36.5 + 2.5n$ $+10.5(t + 1))\|g\|$	C: $5\|g\|$ S: $(4t + 5)\|g\|$ G: $(4t + 5)\|g\|$	C: $5\|g\|$ S: $(3t + 5)\|g\| + t\|q\|$ G: $(4t + 5)\|g\|$
Retrieve Comm. Rounds	10	C: 2 / S: 6 G: 4	C: 2 / S: 4 G: 4

broadcasting $\langle ID_C, B_i, C_i, D_i, \delta_i \rangle$ to other servers, receiving $\langle ID_C, B_j, C_j, D_j, \delta_j \rangle$ for all $j \neq i$ from other servers, and finally sending msg_i^* to the gateway.

The performance of the second proposed protocol is also shown in Table 1, comparing with the performance of Camenisch et al. protocol [6] and the first proposed protocol.

From Table 1, we can see that the retrieval computation complexity of the second proposed protocol in a server is more than that in the first proposed protocol, but is much less than that in Camenisch et al. protocol. The retrieval communication complexity of the second proposed protocol in a server is less than that in the first proposed protocol because $|q|$ is less than $|g|$. In addition, the second proposed protocol has reduce the communications among serves from two phases to one phase. The total communication overhead for each proposed protocol is also much less than that in Camenisch et al. protocol.

4. Security Analysis

4.1. Security Analysis of the Proposed TPASS Protocol Based on Two-Phase Commitment

Based on the security model defined in Section 2, we have the following theorem:

Theorem 1. Assuming that the decisional Diffie-Hellman (DDH) problem [7] is hard over $\{\mathbb{G}, q, g_1\}$, then the proposed TPASS protocol P based on two-phase commitment illustrated in Fig. 2 is secure against the passive attack according to Definition 1.

Proof. In the security analysis, we consider the worst case where $t-1$ servers have been corrupted. Without loss of generality, we assume that the first server S_1 is honest and the rest have been corrupted.

Given a passive adversary $\mathcal{A}$ attacking the protocol, we imagine a simulator $\mathcal{S}$ that runs the protocol for $\mathcal{A}$.

First of all, the simulator $\mathcal{S}$ initializes the system by generating public parameters $\mathsf{params} = \{\mathbb{G}, q, g_1, g_2, H\}$. Next, $\mathsf{Server} = \{\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_n\}$ and Client

sets are determined. For each $C \in \mathsf{Client}$, a password pw_C and a secret s_C are chosen at random and then secret-shared with the n servers. In addition, the digest of the secret $H(s_C)$ is also secret-shared with the n servers.

The public parameters params and the shares $\{ID_C, i, f_1(i), f_2(i), f_3(i)\}$ for $i = 2, 3, \cdots, t$ are provided to the adversary. When answering to any oracle query, the simulator $\mathcal{S}$ provides the adversary $\mathcal{A}$ with the internal state of the corrupted servers S_i $(i = 2, 3, \cdots, t)$.

We refer to the real execution of the experiment, as described above, as P_0. We introduce a sequence of transformations to the experiment P_0 and bound the effect of each transformation on the adversary's advantage. We then bound the adversary's advantage in the final experiment. This immediately yields a bound on the adversary's advantage in the original experiment.

Experiment P_1: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_0 except that the adversary's queries to $\mathsf{Execute}$ oracles are handled differently: in any $\mathsf{Execute}(C, i, \mathbb{S})$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$, the password pw_C in $\mathsf{msg}_C = \langle ID_C, A\rangle$ where $A = g_1^r g_2^{\mathsf{pw}_C}$ are replaced with a random pw in $\mathbb{Z}_q^*$.

Because r in $A = g_1^r g_2^{\mathsf{pw}_C}$ is randomly chosen from $\mathbb{Z}_q^*$ by the simulator, the adversary cannot distinguish $g_1^r g_2^{\mathsf{pw}_C}$ with $g_1^r g_2^{\mathsf{pw}}$. Otherwise, we can break the semantic security of the ElGamal encryption, i.e., given an ElGamal encryption (g^r, my^r) where m is either $g_2^{\mathsf{pw}_C}$ or g_2^{pw}, g is a generator of $\mathbb{G}$, and $y = g_1$, determine if it is an encryption of $g_2^{\mathsf{pw}_C}$. The semantic security of the ElGamal encryption is built on the DDH assumption. Therefore, we have

Claim 1. If the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}^{P_0}_{\mathcal{PA}}(k) - \mathsf{Adv}^{P_1}_{\mathcal{PA}}(k)|$ is negligible.

Experiment P_2: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_1 except that: for any $\mathsf{Execute}(C, i, \mathbb{S})$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$ and $\mathsf{corrupt}(\mathsf{S}_1)$, $a_1 f_1(1)$ in $\mathsf{msg}_1 = \langle ID_C, \delta_1, B_1, C_1, D_1\rangle$ where $B_1 = g_1^{r_1} g_2^{a_1 f_1(1)}$ is replaced by a random number in $\mathbb{Z}_q^*$.

Because r_1 in $B_1 = g_1^{r_1} g_2^{a_1 f_1(1)}$ is randomly chosen from $\mathbb{Z}_q^*$ by the simulator, the adversary cannot distinguish $g_1^{r_1} g_2^{a_1 f_1(1)}$ with $g_1^{r_1} g_2^{\alpha}$, where α is a random number in $\mathbb{Z}_q^*$.

Claim 2. If the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{PA}}^{P_1}(k) - \mathsf{Adv}_{\mathcal{PA}}^{P_2}(k)|$ is negligible.

Experiment P_3: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_2 except that: for any $\mathsf{Execute}(C, i, \mathbb{S})$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$ and $\mathsf{corrupt}(\mathsf{S}_1)$, E_1 in $\mathsf{msg}_1^* = \langle ID_C, C, D, E_1, F_1 \rangle$ is replaced with a random element in the group $\mathbb{G}$.

The difference between the current experiment and the previous one is bounded by the probability to solve the decisional Diffie-Hellman (DDH) problem over $\{\mathbb{G}, q, g_1\}$. More precisely, we have

Claim 3. If the DDH problem over $\{\mathbb{G}, q, g_1\}$ is hard, $|\mathsf{Adv}_{\mathcal{PA}}^{P_2}(k) - \mathsf{Adv}_{\mathcal{PA}}^{P_3}(k)|$ is negligible.

If $|\mathsf{Adv}_{\mathcal{PA}}^{P_2}(k) - \mathsf{Adv}_{\mathcal{PA}}^{P_3}(k)|$ is non-negligible, we show that the simulator can use $\mathcal{A}$ as a subroutine to solve the DDH problem with non-negligible probability as follows.

Given a DDH problem (g_1^x, g_1^y, Z), where x, y are randomly chosen from $\mathbb{Z}_q^*$ and Z is either g_1^{xy} or a random element z from $\mathbb{G}$, the simulator replaces g_1^r in $A = g_1^r g_2^{\mathsf{pw}_C}$ with g_1^x, $C_1 = g_1^{c_1}$ with g_1^y, and $(g_1^{c_1}, g_1^{c_1 r})$ in

$$E_1 = g_2^{a_1 f_2(1) h_1} g_1^{-r_1 \sum_{j=1}^{t} c_j} g_1^{c_1 \sum_{j=1}^{t} r_j} g_1^{c_1 r}$$

with g_1^y, Z, respectively, where r_j $(j = 1, 2, \cdots, t)$ and c_j $(j = 2, 3, \cdots, t)$ are randomly chosen by the simulator. When $Z = g^{xy}$, the experiment is the same as the experiment P_2. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_3. If the adversary can distinguish the experiments P_2 and P_3 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability.

Experiment P_4: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_3 except that: for any $\mathsf{Execute}(C, i, \mathbb{S})$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$ and $\mathsf{corrupt}(\mathsf{S}_1)$, F_1 in $\mathsf{msg}_1^* = \langle ID_C, C, D, E_1, F_1\rangle$ is replaced with a random element in the group $\mathbb{G}$.

Like the experiment P_3, we have

Claim 4. If the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{P}\mathcal{A}}^{P_3}(k) - \mathsf{Adv}_{\mathcal{P}\mathcal{A}}^{P_4}(k)|$ is negligible.

In experiment P_4, msg_C, msg_1, msg_1^* in $\mathsf{Execute}$ oracles have become independent of the password pw_C used by the client C and the secret s_C and $g_2^{H(s_C)}$ in the view of the adversary $\mathcal{A}$, if $\mathcal{A}$ has not require $\mathsf{Corrupt}(C)$ and $\mathsf{Corrupt}(\mathsf{S}_1)$. In view of this, any off-line dictionary attack cannot succeed.

Experiment P_5: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_4 except that: for any $\mathsf{Execute}(C, i, \mathbb{S})$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$ and $\mathsf{corrupt}(\mathsf{S}_1)$, the secret s_C of the client is replaced with a random element in the group $\mathbb{G}$.

Given a DDH problem (g_1^x, g_1^y, Z), where x, y are randomly chosen from $\mathbb{Z}_q^*$ and Z is either g_1^{xy} or a random element z from $\mathbb{G}$, the simulator replaces g_1^r in $A = g_1^r g_2^{\mathsf{pw}_C}$ with g_1^x, $C_1 = g_1^{c_1}$ with g_1^y, and $(g_1^r, g_1^{c_1 r})$ in

$$s_C = (E/C^r)^{h^{-1}} = (E/(g_1^{r\sum_{i=2}^{t} c_i} g_1^{rc_1}))^{h^{-1}}$$

with g_1^x, Z, respectively, where $h = H(ID_C, A, C, D)$, c_j $(j = 2, 3, \cdots, t)$ are randomly chosen by the simulator. When $Z = g^{xy}$, the experiment is the same as the experiment P_4. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_5. If the adversary can distinguish the experiments P_4 and P_5 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability. Therefore, we have

Claim 5. If the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{P}\mathcal{A}}^{P_4}(k) - \mathsf{Adv}_{\mathcal{P}\mathcal{A}}^{P_5}(k)|$ is negligible.

In experiment P_5, when the passive adversary $\mathcal{A}$ queries the $\mathsf{Test}(C, i)$ oracle, the simulator $\mathcal{S}$ chooses a random bit b. When the adversary completes its execution and outputs a bit b', the simulator can tell whether the adversary succeeds by checking if (1) $\mathsf{Test}(C, i)$ query was made regarding some fresh client C, and (2) $b' = b$.

The passive adversary's probability of correctly guessing the bit b used by the Test oracle is exactly 1/2 when the Test query is made to a fresh client instance C^i invoked by an $\mathsf{Execute}(C, i, \mathbb{S})$ oracle. This is so because the secret s_C is chosen at random from $\mathbb{G}$, and hence there is no way to distinguish whether the Test oracle outputs a random secret or the "actual" secret (which is a random element, anyway). Therefore, $\mathsf{Adv}^{P_5}_{\mathcal{PA}}(k) = 2 \cdot 1/2 - 1 = 0$. Based on Claims 1-5, we know $|\mathsf{Adv}^{P_0}_{\mathcal{PA}}(k) - \mathsf{Adv}^{P_5}_{\mathcal{PA}}(k)|$ is negligible. According to Definition 1, the protocol is secure against the passive attack and the theorem is proved. $\triangle$

Theorem 2. Assuming that the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$ and H is a collision-resistant hash function, then the proposed TPASS protocol P based on two-phase commitment illustrated in Fig. 2 is secure against the active attack according to Definition 2.

Proof. Like in the proof of Theorem 1, we consider the worst case where $t-1$ servers have been corrupted. The worst case can be divided into two subcases: (i) in the protocol, there is one honest server which has not been compromised by the adversary; (ii) all servers in the protocol are dishonest and controlled by the adversary.

For the first subcase, without loss of generality, we assume that the first server S_1 is honest and the rest have been corrupted. Because of the inspection of the honest server, the published public parameters $\mathbb{G}, g_1, g_2, q$ cannot be changed and no one knows the discrete logarithm of g_2 based on g_1. Otherwise, it turns to the second subcase. This proof concentrates on the instances invoked by Send oracles. We view the adversary's queries to its Send oracles as queries to four different oracles as follows:

- $\mathsf{Send}(C, i)$ represents a request for instance C^i of client C to initiate the protocol. The output of this query is $\mathsf{msg}_C = \langle ID_C, A\rangle$.

- $\mathsf{Send}(\mathsf{S}_1, j, C, \mathsf{msg}_C)$ represents sending message msg_C to instance S_1^j of the server S_1, supposedly from the client C. The input of this query is $\mathsf{msg}_C = \langle ID_C, A\rangle$ and the output of this query is $\mathsf{msg}_1 = \langle ID_C, \delta_1, B_1, C_1, D_1\rangle$.
- $\mathsf{Send}(\mathsf{S}_1, j, \mathsf{S}_2, \mathsf{S}_3, \cdots, \mathsf{S}_t, M)$ represents sending message M to instance S_1^j of the server S_1, supposedly from the servers $\mathsf{S}_2, \mathsf{S}_3, \cdots, \mathsf{S}_t$. The input of this query is $M = \mathsf{msg}_2 \| \mathsf{msg}_3 \| \cdots \| \mathsf{msg}_t$ and the output of this query is $\mathsf{msg}_1^* = \langle ID_C, C, D, E_1, F_1\rangle$ or $\perp$.
- $\mathsf{Send}(C, i, \mathsf{msg}_S)$ represents sending the message msg_S to instance C^i of the client C. The input of this query is $\mathsf{msg}_S = \langle ID_C, C, D, E, F\rangle$ and the output of this query is either $\mathsf{acc}_C^i = \mathsf{TRUE}$ or $\perp$.

We will use some terminologies throughout the proof. A given message is called oracle-generated if it was output by the simulator in response to some oracle query. The message is said to be adversarially-generated otherwise. An adversarially-generated message must not be the same as any oracle-generated message.

We say an active adversary $\mathcal{A}$ succeeds if it makes an query $\mathsf{Send}(C, i, \mathsf{msg}_S)$ to a fresh client instance C^i with **an adversarially-generated message** msg_S, resulting in $\mathsf{acc}_C^i = \mathsf{TRUE}$. We denote this event by Succ_A.

Experiment P_6: In this experiment, the simulator interacts with the adversary as P_0 except that the adversary does not succeed, and the experiment is aborted, if any of the following occurs:

1. At any point during the experiment, an oracle-generated message (e.g., msg_C, msg_1, or msg_1^*) is repeated.
2. At any point during the experiment, a collision occurs in the hash function H (regardless of whether this is due to a direct action of the adversary, or whether this occurs during the course of the simulator's response to an oracle query).

It is immediate that events 1 occurs with only negligible probability, event 2 occurs with negligible probability assuming H as collision-resistant hash functions. Put everything together, we are able to see that

Claim 6. If H is a collision-resistant hash function, $|\mathsf{Adv}_{\mathcal{AA}}^{P_0}(k) - \mathsf{Adv}_{\mathcal{AA}}^{P_6}(k)|$ is negligible.

Experiment P_7: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_6 except that (1) the adversary's queries to $\mathsf{Send}(C, i)$ oracles are handled differently: in any $\mathsf{Send}(C, i)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$, the password pw_C in $\mathsf{msg}_C = \langle ID_C, A\rangle$ where $A = g_1^r g_2^{\mathsf{pw}_C}$ is replaced with a random number pw in $\mathbb{Z}_q^*$; (2) the adversary's queries to $\mathsf{Send}(\mathsf{S}_1, j, C, \mathsf{msg}_C)$ oracles are handled differently: in any $\mathsf{Send}(\mathsf{S}_1, j, C, \mathsf{msg}_C)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$ and $\mathsf{corrupt}(\mathsf{S}_1)$, $a_1 f_1(1)$ in $\mathsf{msg}_1 = \langle ID_C, B_1, C_1, D_1\rangle$ where $B_1 = g_1^{r_1} g_2^{a_1 f_1(1)}$ is replaced by a random number in $\mathbb{Z}_q^*$; (3) the adversary's queries to $\mathsf{Send}(\mathsf{S}_1, j, \mathsf{S}_2, \cdots, \mathsf{S}_t,\ \mathsf{msg}_2\| \cdots \|\mathsf{msg}_t)$ oracles are handled differently: in any $\mathsf{Send}(\mathsf{S}_1, j, \mathsf{S}_2, \cdots, \mathsf{S}_t,\ \mathsf{msg}_2\| \cdots \|\mathsf{msg}_t)$, where $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$ and $\mathsf{corrupt}(\mathsf{S}_1)$, E_1 and F_1 in $\mathsf{msg}_1^* = \langle ID_C, C, D, E_1, F_1\rangle$ are replaced with random elements in the group $\mathbb{G}$.

Like in experiments P_1 and P_2, the changes (1) and (2) will only bring a negligible change to the advantage of the active adversary.

For the change (3), if $\mathsf{msg}_C, \mathsf{msg}_2, \mathsf{msg}_3, \cdots, \mathsf{msg}_t$ are all oracle-generated, we can replace E_1 with a random element in $\mathbb{G}$ as in the experiment P_3.

If some of $\mathsf{msg}_C, \mathsf{msg}_2, \mathsf{msg}_3, \cdots, \mathsf{msg}_t$ are adversarially-generated, the adversary $\mathcal{A}$ cannot produce $A, (B_j, C_j, D_j)$ $(j = 2, 3, \cdots, t)$, such as $A\prod_{j=1}^t B_j$ excludes B_1 and $\delta_j = g_1^{H(ID_C, A, B_j, C_j, D_j)}$ for $j = 2, 3, \cdots, t$ still hold because A and the commitments δ_j $(j = 2, 3, \cdots, t)$ must be broadcast and received by the server S_1 at first and H is a collision-resistant hash function.

Because $B_1 = g_1^{r_1} g_2^{a_1 f_1(1)}$, we have

$$\begin{aligned} E_1 &= g_2^{a_1 f_2(1) h_1} C^{-r_1} (A \prod_{j=1}^{t} B_j)^{c_i} \\ &= g_2^{a_1 f_2(1) h_1} C^{-r_1} (A \prod_{j=2}^{t} B_j)^{c_i} g_1^{c_1 r_1} (g_2^{c_1})^{a_1 f_1(1)}. \end{aligned}$$

Given a DDH problem (g_1^x, g_1^y, Z), where x, y are randomly chosen from $\mathbb{Z}_q^*$ and Z is either g_1^{xy} or a random element z from $\mathbb{G}$, the simulator replaces g_2 with g_1^x, $C_1 = g_1^{c_1}$ with g_1^y, and $(g_1^{c_1}, g_2^{c_1})$ in the above E_1 with g_1^y, Z, respectively, where r_1 is randomly chosen by the simulator. When $Z = g^{xy}$, the experiment is the same as the experiment P_6. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_7.

In the same way, we can make the change (4).

Because no one knows the discrete logarithm x of g_2 based on g_1, if the adversary can distinguish the experiments P_6 and P_7 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability. Therefore, we have

Claim 7. If the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{AA}}^{P_6}(k) - \mathsf{Adv}_{\mathcal{AA}}^{P_7}(k)|$ is negligible.

In experiment P_7, msg_C, msg_1, msg_1^* in Send oracles have become independent of the password pw_C used by the client C and the secret s_C and $g_2^{H(s_C)}$ in the view of the adversary $\mathcal{A}$, if $\mathcal{A}$ has not require $\mathsf{Corrupt}(C)$ and $\mathsf{Corrupt}(\mathsf{S}_1)$. In view of this, any off-line dictionary attack cannot succeed.

To evaluate $\mathsf{Pr}[\mathsf{Succ}_\mathsf{A}]$, we consider three cases as follows.

Case 1. The adversary $\mathcal{A}$ forges $\mathsf{msg}'_C = \langle ID_C, A' \rangle$ where $A' = g_1^{r'} g_2^{\mathsf{pw}'_C}$ by choosing his own r' from $\mathbb{Z}_q^*$ and pw'_C from the dictionary D. In this case, if Succ_A occurs, the adversary can conclude that the password used by the client is pw'_C. Therefore, the probability $\mathsf{Pr}[\mathsf{Succ}_A] = Q_1(k)/N$, where $Q_1(k)$ denotes the number of queries to $\mathsf{Send}(\mathsf{S}_1, j, C, \mathsf{msg}_C)$ oracle.

Case 2. Given $\mathsf{msg}_C = \langle ID_C, A \rangle$, the adversary $\mathcal{A}$ forges $\mathsf{msg}_S = \langle ID_C, C', D', E', F' \rangle$ by choosing his own s', c', d' from $\mathbb{Z}_q^*$ and pw'_C from the dictionary D and computing $C' = g_1^{c'}, D' = g_1^{d'}, E' = g_2^{s'h'}(Ag_2^{\mathsf{pw}'_C})^{c'}, F' = g_2^{H(g_2^{s'})h'}(Ag_2^{\mathsf{pw}'_C})^{d'}$ where $h' = H(ID_C, A, C', D')$. When $\mathsf{pw}_C = \mathsf{pw}'_C$, we have $\mathsf{acc}_C^i = \mathsf{TRUE}$. Therefore, in this case, the probability $\mathsf{Pr}[\mathsf{Succ}_A] = Q_2(k)/N$, where $Q_2(k)$ denotes the number of queries to $\mathsf{Send}(C, i, \mathsf{msg}_S)$ oracle.

Case 3. Given $\mathsf{msg}_C = \langle ID_C, A \rangle$, the adversary $\mathcal{A}$ forwards msg_C to the server S_1 twice to get two responses $\langle ID_C, C_1, D_1, E_1, F_1\rangle$ and $\langle ID_C, C_1', D_1', E_1', F_1'\rangle$. Then the adversary $\mathcal{A}$ sends to the client a forged message $\mathsf{msg}_S = \langle ID_C, C_1'/C_1,\ D_1'/D_1, g_2^{s^*h^*}E_1'/E_1, g_2^{H(g_2^{s^*})h^*}F_1'/F_1\rangle$, where $E_1'/E_1 = g_1^{s(h'-h)}(C_1'/C_1)^r$, $F_1'/F_1 = g_1^{H(g_1^s)(h'-h)}(D_1'/D_1)^r$, $h = H(ID_C, A, C, D)$, $h' = H(ID_C, A, C', D')$, $h^* = H(ID_C, A, C_1'/C_1, D_1'/D_1)$ and s^* is chosen from $\mathbb{Z}_q^*$ by the adversary. The client accepts msg_S if and only if $h' = h$. Because H is a collision-resistant hash function, the probability $\mathsf{Pr}[\mathsf{Succ}_A]$ is negligible in this case.

In summary, in experiment P_7, $\mathsf{Pr}[\mathsf{Succ}_A] = Q(k)/N$, where $Q(k)$ denotes the number of on-line attacks. Based on Claims 6-7, we know $|\mathsf{Adv}^{P_0}_{\mathcal{PA}}(k) - \mathsf{Adv}^{P_7}_{\mathcal{PA}}(k)|$ is negligible. According to Definition 2, the protocol is secure against the active attack in the first subcase. $\triangle$

For the second subcase, all servers in the protocol are dishonest and controlled by the adversary, the active adversary can cheat the client with forged public parameters $\mathbb{G}, q, g_1, g_2$. However, the client can check if q is a large prime and $g_1^q = g_2^q = 1$ so that the discrete logarithm over $\mathbb{G}$ is hard although the adversary may know the discrete of g_2 based on g_1.

In this case, the active adversary can only query two Send oracles: $\mathsf{Send}(C, i)$ and $\mathsf{Send}(C, i, \mathsf{msg}_S)$.

Experiment P_8: In this experiment, the simulator interacts with $\mathcal{A}$ as in experiment P_0 except that the adversary's queries to $\mathsf{Send}(C, i)$ oracles are handled differently: in any $\mathsf{Send}(C, i)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(C)$, the password pw_C in $\mathsf{msg}_C = \langle ID_C, A\rangle$ where $A = g_1^r g_2^{\mathsf{pw}_C}$ is replaced with a random number pw in $\mathbb{Z}_q^*$.

Like in experiment P_1, if the adversary can distinguish $g_1^r g_2^{\mathsf{pw}_C}$ with $g_1^r g_2^{\mathsf{pw}}$, we can break the semantic security of the ElGamal encryption. Therefore, we have

Claim 8. If the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}^{P_0}_{\mathcal{AA}}(k) - \mathsf{Adv}^{P_8}_{\mathcal{AA}}(k)|$ is negligible.

In experiment P_8, the adversary can only perform the active attack as described in Case 2 of experiment P_7. In this case, the probability $\mathsf{Pr}[\mathsf{Succ}_A] = Q_2(k)/N$, where $Q_2(k)$ denotes the number of queries to $\mathsf{Send}(C, i, \mathsf{msg}_S)$ oracle.

Based on Claim 8, we know $|\mathsf{Adv}^{P_0}_{\mathcal{PA}}(k) - \mathsf{Adv}^{P_8}_{\mathcal{PA}}(k)|$ is negligible. According to Definition 2, the protocol is secure against the active attack in the second subcase.

This completes the proof of the theorem. $\triangle$

4.2. Security Analysis of the Proposed TPASS Protocol Based on Zero-Knowledge Proof

Before analyzing the security of the second proposed protocol, we introduce an assumption about non-interactive zero-knowledge proof of knowledge as follows.

Non-Interactive Zero-Knowledge Proof of Knowledge Assumption. Considering a protocol where Prover, wishing to prove to Verifier that he knows x such that $y = g^x$, sends $(R = g^r, \alpha = H(g, y, R)x + r)$ (where r is randomly chosen by Prover) to Verifier, who accepts the proof if $g^\alpha = y^{H(g,r,R)}R$ and otherwise rejects the proof. We assume that in this protocol, Prover has to know x to generate the non-interactive zero-knowledge proof of knowledge (R, α) such that $g^\alpha = y^{H(g,r,R)}R$ and Verifier gains no knowledge of x after the proof.

Theorem 3. Assuming that DDH problem is hard over $\{\mathbb{G}, q, g_1\}$ and the non-interactive zero-knowledge proof of knowledge assumption holds, the proposed TPASS protocol P based on zero-knowledge proof illustrated in Fig. 3 is secure against the passive attack according to Definition 1.

Proof. The proof is the same as that of Theorem 1, except from Claims 3 and 4.

Claim 3'. If the DDH problem over $\{\mathbb{G}, q, g_1\}$ is hard and the non-interactive zero-knowledge proof of knowledge assumption holds, $|\mathsf{Adv}^{P_2}_{\mathcal{PA}}(k) - \mathsf{Adv}^{P_3}_{\mathcal{PA}}(k)|$ is negligible.

If $E_1 = g_2^{a_1 f_2(1) h_1} C^{-r_1} (A \prod_{j=1}^{t} B_j)^{c_i}$ contains $B_1 = g_1^{r_1} g_2^{a_1 f_1(1)}$, Claim 3' can be proved to be true in the same way as the proof of Theorem 1. If E_1

does not contain B_1, E_1 must contain $g_1^{r_1 c_1}$ because $C = g_1^{\sum c_j}$ and the zero-knowledge proof of knowledge assumption ensures the server S_j knowing c_j such that $C_j = g_1^{c_j}$. Given $C_1 = g_1^{c_1}$ and $g_1^{r_1} = B_1 g_2^{\sum_{j \neq 1} a_j f_1(j)} / g_2^{\mathsf{pw}}$ (where pw is chosen for offline dictionary attack), the $t-1$ servers $S_2, \cdots, S_t$ cannot distinguish $g_1^{r_1 c_1}$ with a random group element due to the DDH assumption. Note that the zero-knowledge proof of knowledge assumption ensures the server S_j $(j \neq 1)$ having no knowledge of c_1. Therefore, Claim 3' is true.

Claim 4'. If the DDH problem over $\{\mathbb{G}, q, g_1\}$ is hard and the non-interactive zero-knowledge proof of knowledge assumption holds, $|\mathsf{Adv}_{\mathcal{PA}}^{P_3}(k) - \mathsf{Adv}_{\mathcal{PA}}^{P_4}(k)|$ is negligible.

The proof can be obtained from the proof of Claim 3' by replacing E_1, C, C_j, c_j with F_1, D, D_j, d_j. Please note that (C_j, D_j, δ_j) is also a non-interactive zero-knowledge proof of knowledge of d_j. $\triangle$

Theorem 4. Assuming that the DDH problem is hard over $\{\mathbb{G}, q, g_1\}$, the non-interactive zero-knowledge proof of knowledge assumption holds, and H is a collision-resistant hash function, then the proposed TPASS protocol P based on zero-knowledge proof illustrated in Fig. 3 is secure against the active attack according to Definition 2.

Theorem 4 can be proved by combining the proofs of Theorems 1-3.

5. Experiments

Experiments have been carried out to validate the performance of the proposed two protocols. The experiments are performed with the following hardware specifications: CPU: 2.2 GHz Intel Core i7, Memory: 16 GB 1600 MHz DDR3.

We implemented the proposed two protocols with JRE System Library [JavaSE - 1.7] in the settings where there are 3, 5, 10, 15, 20 servers, respectively. In our experiments, the size of a group element (i.e, $|g|$) has 1024 bits and the order of the group (i.e, $|q|$) has 160 bits. One modular exponentiation for 160-bit

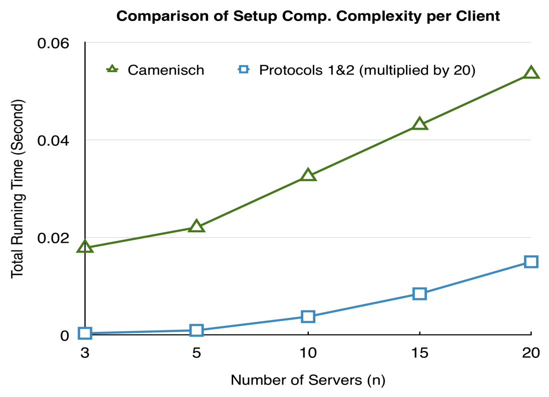


Figure 4: Comparison of time spent (in seconds) for setting up

exponent takes approximately 0.0028 seconds and one modular multiplication takes approximate 0.000005 seconds.

In the following discussion, we mainly compare the performance of our protocols with Camenisch et al. protocol [6].

5.1. Performance of Initialization

For $n = 3, 5, 10, 15, 20$, the performance of the proposed two protocols per client in the initialisation, comparing with Camenisch et al. protocol [6], is illustrated in Fig. 4-5.

Because the proposed two protocols have the same initialization, their performance are the same.

In order to make the performance of the proposed protocols visible in the comparison, the computational and communication complexities of the proposed protocols for a client have been enlarged by 20 times.

Form Fig. 4 and Fig. 5, we can see that the proposed protocols are significant more efficient than Camenisch et al.'s protocol and the difference increases with the increase of the number of servers (n).

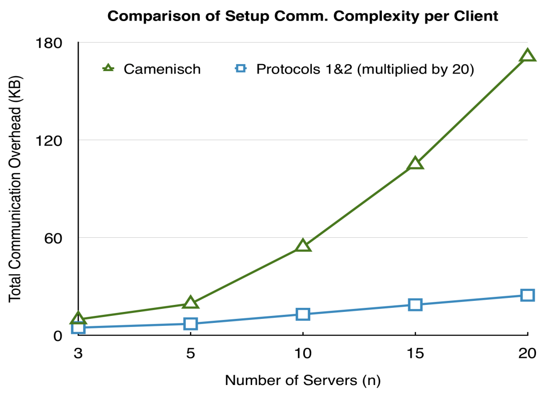


Figure 5: Comparison of communication size (in KB) for setting up

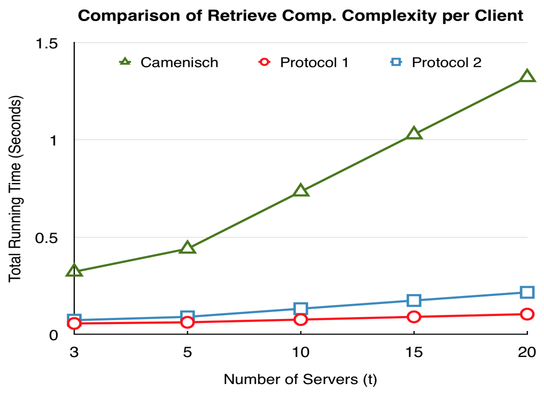


Figure 6: Comparison of time spent for retrieving

5.2. Performance of Retrieve

For $t = 3, 5, 10, 15, 20$, the performance of the proposed two protocols per client in the retrieval, comparing with Camenisch et al. protocol [6], is illustrated in Fig. 6-7.

From Fig. 6, we can see that the total running time of the proposed two protocols per client is less than Camenisch et al.'s protocol. We save up to 95% in the first proposed protocol and 85% in the second proposed protocol. Although the difference is just a couple of seconds for a client, it will become significantly large when the servers provide services to a large number of clients

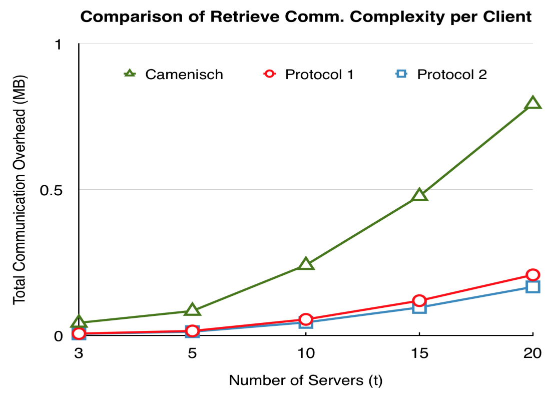


Figure 7: Comparison of communication size for retrieving

concurrently. If we ignore the communication time, the verification of the second proposed protocol is a little bit slower that the first proposed protocol. However, if the communication time cannot be ignored, the second proposed protocol may be more efficient than the first proposed protocol because it reduces the communications of the servers from two phases to one phase.

The communication overhead of Camenisch et al.'s protocol depends on the total number n of the servers. In Fig. 7, we assume $n = t + 1$. In addition, we assume that the broadcast channel in the proposed two protocols is implemented by point to point communication. In this case, the total communication overheads of the proposed two protocols are $(4t^2 + 5t)|g|$ bits and $(3t^2 + 5t)|g| + t^2|q|$ bits, respectively.

From Fig. 7, we can see that the proposed two protocols have almost the same communication overhead, which is significant less than Camenisch et al.'s protocol. We save up to 65% in the first proposed protocol and 75% in the second proposed protocol.

In addition, for $t = 10$, the performance of the proposed two protocols per server in the retrieval, comparing with Camenisch et al. protocol [6], is illustrated in Fig. 8-9, which also shows that the proposed protocols are more efficient than Camenisch et al.'s protocol.

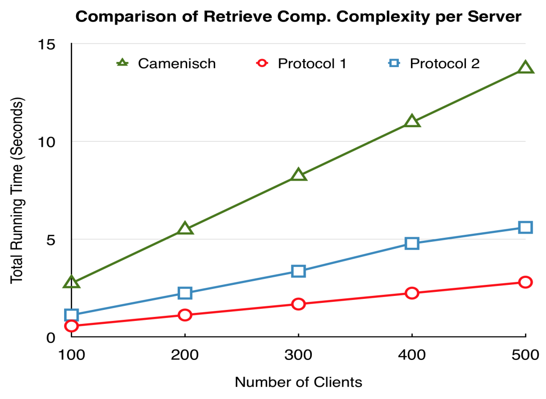


Figure 8: Comparison of average time spent by a server for retrieving

6. Conclusion

In this paper, we have presented two efficient t-out-of-n TPASS protocols for any $n > t$ that protects the password of the client when he tries to retrieve his secret from all corrupt servers as well as prevents the adversary from planting a different secret into the user's mind than the secret that he stored earlier. The proposed protocols are significantly more efficient than existing TPASS protocols. Furthermore, we have provided a rigorous proof of security for the proposed protocols in the standard model and performed some experiments.

Our future work will study how efficiently to detect the corrupted servers and implement the proposed protocol in light-weight mobile devices to support cloud-based services.

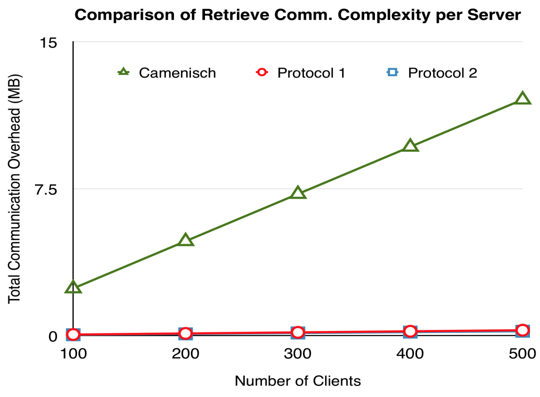


Figure 9: Comparison of average communication size for a server in retrieving

References

[1] M. Abdalla, M. Cornejo, A. Nitulescu, and D. Pointcheval. Robust password-protected secret sharing. ESORICS'16, pages 61-79, 2016.

[2] A. Bagherzandi, S. Jarecki, N. Saxena, Y. Lu. Password-protected secret sharing. ACM CCS'11, pages 433-444, 2011.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. Eurocrypt'00, pages 139-155, 2000.

[4] J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. Nightingale: A new two-server approach for authentication with short secrets. 12th USENIX Security Symp., pages 201-213, 2003.

[5] J. Camenisch, A. Lysyanskaya, and G. Neven. Practical yet universally composable two-server password-authenticated secret sharing. ACM CCS'12, pages 525-536, 2012.

[6] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. Crypto'14, pages 256-275, 2014.

[7] W. Diffie and M. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, 32(2): 644-654, 1976.

[8] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 31(4): 469-472, 1985.

[9] W. Ford and B. S. Kaliski, Server-assisted generation of a strong secret from a password, 9th IEEE Intl. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000), Pages 176-180.

[10] D. Jablon. Password authentication using multiple servers. CT-RSA'01, pages 344-360, 2001.

[11] S. aw Jarecki, A. Kiayias, H. Krawczyk, J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. ACNS'17, pages 39-58, 2017.

[12] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. Eurocrypt'01, pages 457-494, 2001.

[13] J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. ACNS'05, pages 1-16, 2005.

[14] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. J. Cryptology, 19(1): 27-66, 2006.

[15] M. Di Raimondo and R. Gennaro. Provably Secure Threshold Password-Authenticated Key Exchange. J. Computer and System Sciences, 72(6): 978-1001, 2006.

[16] A. Shamir. How to share a secret. Communications of the ACM, 22(11): 612-613, 1979.

[17] M. Szydlo and B. Kaliski. Proofs for two-server password authentication. CT-RSA'05, pages 227-244, 2005.

[18] X. Yi, S. Ling, and H. Wang. Efficient two-server password-only authenticated key exchange. IEEE Trans. Parallel Distrib. Syst., 24(9): 1773-1782, 2013.

[19] X. Yi, F. Hao, E. Bertino. ID-based two-server password-authenticated key exchange. ESORICS'14, pages 257-276, 2014.

[20] X. Yi, F. Hao, L. Chen, J. K. Liu. Practical threshold password-authenticated secret sharing protocol. ESORICS'15, pages 347-365, 2015.