Practical Threshold Password-Authenticated Secret Sharing Protocol

Xun Yi[1(✉)], Feng Hao[2], Liqun Chen[3], and Joseph K. Liu[4]

[1] School of CS and IT, RMIT University, Melbourne, Australia
Xun.yi@rmit.edu.au
[2] School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
[3] Hewlett-Packard Laboratories, Bristol, UK
[4] Faculty of Information Technology, Monash University, Melbourne, Australia

Abstract. Threshold password-authenticated secret sharing (TPASS) protocols allow a client to secret-share a secret s among n servers and protect it with a password pw, so that the client can later recover s from any subset of t of the servers using the password pw, but so that no coalition smaller than t learns anything about s or can mount an offline dictionary attack on the password pw. Some TPASS protocols have appeared in the literature recently. The protocol by Bagherzandi et al. (CCS 2011) leaks the password if a client mistakenly executes the protocol with malicious servers. The first t-out-of-n TPASS protocol for any $n > t$ that does not suffer from this shortcoming was given by Camenisch et al. (CRYPTO 2014). This protocol, 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. In this paper, we present a practical TPASS protocol which is in particular efficient for the client, who only needs to send a request and receive a response. In addition, we have provided a rigorous proof of security for our protocol in the standard model.

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 [5], 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. Practical applications include secure password managers (where the shared secret is a list of strongly random website passwords) and encrypting

© Springer International Publishing Switzerland 2015
G. Pernul et al. (Eds.): ESORICS 2015, Part I, LNCS 9326, pp. 347–365, 2015.
DOI: 10.1007/978-3-319-24174-6_18

data in the cloud (where the shared secret is the encryption key) based on a single master password.

The first TPASS protocol was given by Bagherzandi et al. [1]. It is built on the PKI model, secure under the decisional Diffie-Hellman assumption, using non-interactive zero-knowledge proofs. The basic idea is: The client initially generates an ElGamal public and private key pairs $(sk, pk = g^{sk})$ [7] and secret-shares sk among servers using an t-out-of-n secret sharing [15] 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 sends the encryption $E(g^{\mathsf{pw}'}, pk)$ to the servers, each of which computes and returns $A_i = [E(g^{\mathsf{pw}}, pk)/E(g^{\mathsf{pw}'}, pk)]^{t_i} = E(g^{t_i(\mathsf{pw}-\mathsf{pw}')}, pk)$. 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 $B = E(s, pk)A = E(sg^{\sum t_i(\mathsf{pw}-\mathsf{pw}')}, pk)$ 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. [1] 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 sends out an encryption of his password attempt pw' he 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 encrypt $g^{\mathsf{pw}'}$, the malicious servers can perform an offline dictionary attack on $g^{\mathsf{pw}'}$ to obtain the password pw'.

Authenticating to the wrong servers is a common scenario when users are tricked in phishing attacks. To overcome this shortcoming, Camenisch et al. [5] proposed the first t-out-of-n TPASS protocol for any $n > t$ that does not require trusted, user-specific state information to be carried over from the setup phase. The protocol requires the client to only remember a username and a password, assuming that a PKI is available. If the client misremember his list of servers and tries to retrieve his 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 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 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. protocol, 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.

Our Contribution. We provide a practical t-out-of-n TPASS protocol for any $n > t$. The basic idea is: The 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. In the end, the client then decrypts the two ciphertexts and accepts the secret if one decrypted value is another's digest.

Our protocol is significantly more efficient than Camenisch et al. protocol [5] in terms of communication rounds for the client and computation and communication complexities as well. In our protocol, 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 $t + 10$ 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 a rigorous proof of security for our protocol in the standard model. Like Camenisch et al. protocol [5], our protocol can protect the password of the client even if he communicates with all-malicious servers by mistake. In addition, it prevents the servers from planting a different secret into the user's mind than the secret that he stored earlier.

Related Work. 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 [1].

The first TPAKE protocols, due to Ford and Kaliski [8] and Jablon [9], 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. [12]. The 1-out-of-2 protocol of Brainard et al. [3] is implemented in EMC's RSA Distributed Credential Protection [14]. 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 [13] and the 1-out-of-2 protocol by Katz et al. [11] 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 password, was proposed by Yi et al. [17]. In case that the public parameters for IBE can be tampered and changed by the adversary, the protocol leaks the password.

In addition, the 1-out-of-2 TPASS by Camenisch et al. [4] leaks the password when the client tries to retrieve his secret from a set of all-malicious servers.

2 Definition of Security

In this section, we define the security for TPASS protocol on the basis of the security models for PAKE [2,10].

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. In the real world, 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 secret of a client, where freshness of the secret is meant to indicate that the adversary does not trivially know the value of the secret. We say a secret s_C^i is fresh if (1) C is not corrupted and (2) at least $n - t + 1$ out of n servers are not corrupted.

Advantage of the Adversary. Informally, the adversary succeeds if it can guess the bit b used by the Test oracle. We say an adversary $\mathcal{A}$ succeeds if it makes a single 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 single bit b' with $b' = b$ (recall that b is the bit chosen by the Test oracle). We denote this event by Succ. The advantage of adversary $\mathcal{A}$ in attacking protocol P is then given by

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

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).

An adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack. A protocol is secure 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. Instances with which the adversary interacts via $\mathsf{Execute}$ are not counted as on-line attacks. 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 1. Protocol P is a secure TPASS protocol if, for all dictionary size N and for all 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,

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

3 Our TPASS Protocol

3.1 Description of Our Protocol

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 generators g_1 and a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$. Then the n servers cooperate to generate g_2, like [16], such that none 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 $\{ID, i, f_1(i), f_2(i), f_3(i)\}$) to retrieve the secret s_C as shown in Fig. 1.

In Fig. 1, TPASS protocol is executed in three phases 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.

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 as follows.

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

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

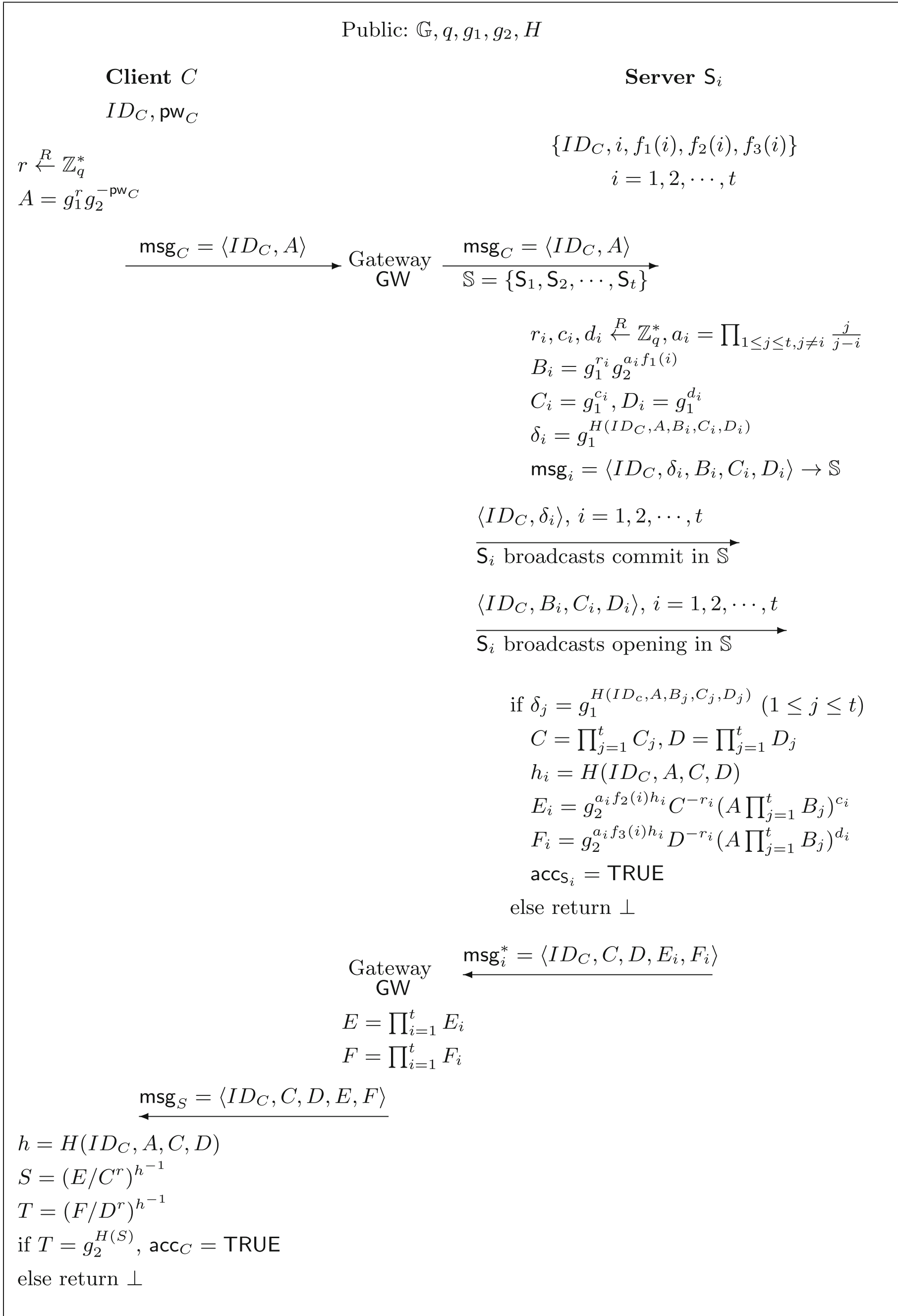

Fig. 1. Our TPASS protocol P

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

Then S_i broadcasts $\mathsf{msg}_i = \langle ID_C, \delta_i, B_i, C_i, D_i \rangle$ in $\mathbb{S}$ in two phases. In the commit phase, S_i broadcasts its commitment $\langle ID_C, \delta_i \rangle$. After receiving all commitments $\langle ID_C, \delta_j \rangle$ $(1 \leq j \leq t)$, S_i broadcasts its opening $\langle ID_C, B_i, C_i, D_i \rangle$ in the reveal phase.

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

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

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

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 $\mathsf{acc}_C = \mathsf{TRUE}$ and $\perp$ otherwise.

3.2 Correctness and Efficiency

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

$$C = \prod_{j=1}^{t} C_j = g_1^{\sum_{j=1}^{t} c_j}$$

$$D = \prod_{j=1}^{t} D_j = g_1^{\sum_{j=1}^{t} d_j}$$

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

$$
\begin{aligned}
&= g_2^{a_i f_2(i) h_i} g_1^{-r_i \sum_{j=1}^t c_j} (g_1^r g_2^{-\mathsf{pw}_C} g_1^{\sum_{j=1}^t r_j} g_2^{\mathsf{pw}_C})^{c_i} \\
&= g_2^{a_i f_2(i) h_i} g_1^{-r_i \sum_{j=1}^t c_j} g_1^{c_i \sum_{j=1}^t r_j} g_1^{c_i r} \\
F_i &= g_2^{a_i f_3(i) h_i} D^{-r_i} (A \prod_{j=1}^t B_j)^{d_i} \\
&= g_2^{a_i f_3(i) h_i} g_1^{-r_i \sum_{j=1}^t d_j} (g_1^r g_2^{-\mathsf{pw}_C} g_1^{\sum_{j=1}^t r_j} g_2^{\mathsf{pw}_C})^{d_i} \\
&= g_2^{a_i f_3(i) h_i} g_1^{-r_i \sum_{j=1}^t d_j} g_1^{d_i \sum_{j=1}^t r_j} g_1^{d_i r} \\
h &= h_1 = h_2 = \cdots = h_t \\
&= H(ID_C, A, C, D) \\
E &= \prod_{i=1}^t E_i = \prod_{i=1}^t g_2^{a_i f_3(i) h} g_1^{-r_i \sum_{j=1}^t d_j} g_1^{d_i \sum_{j=1}^t r_j} g_1^{d_i r} \\
&= g_2^{sh} g_1^{-\sum_{i=1}^t r_i \sum_{j=1}^t c_j} g_1^{\sum_{i=1}^t c_i \sum_{j=1}^t r_j} g_1^{r \sum_{i=1}^t c_i} \\
&= g_2^{sh} C^r \\
F &= \prod_{i=1}^t F_i = \prod_{i=1}^t g_2^{a_i f_3(i) h} g_1^{-r_i \sum_{j=1}^t d_j} g_1^{d_i \sum_{j=1}^t r_j} g_1^{d_i r} \\
&= g_2^{H(g_2^s) h} g_1^{-\sum_{i=1}^t r_i \sum_{j=1}^t d_j} g_1^{\sum_{i=1}^t d_i \sum_{j=1}^t r_j} g_1^{r \sum_{i=1}^t d_i} \\
&= g_2^{H(g_2^s) h} D^r
\end{aligned}
$$

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, our TPASS protocol has correctness.

Efficiency. In our TPASS 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 [5] and our protocol can be shown in Table 1.

Table 1. Performance comparison of Camenisch et al. protocol and our protocol

	Camenisch et al. protocol [5]	Our protocol
Public keys	Client: username Server S_i: epk_i, spk_i, tpk_i	Client: username Server S_i: none
Private keys	Client: pw_C Server S_i: esk_i, ssk_i, tsk_i $E(\mathsf{pw}_C, pk), E(s, pk), pk = \prod epk_i$ $E(\mathsf{pw}_C, tpk), E(s, tpk), tpk = \prod tpk_i$	Client: pw_C Server 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	Client: $5n + 15$ (exp.) Server: $n + 18$ (exp.)	Client: $3n$ polynomial evaluations Server: none
Setup Comm. Complexity	$n(2.5n + 18.5)\|g\|$	Client: $3n\|q\|$ Server: $3\|q\|$
Setup Comm. Round	4	Client: 1 Server: 1
Retrieve Comp. Complexity	Client: $14t + 24$ (exp.) Server: $7t + 28$ (exp.)	Client: 7 (exp.) Server: $t + 10$ (exp.) Gateway: 0 (exp.)
Retrieve Comm. Complexity	$(t + 1)(36.5 + 2.5n$ $+10.5t(t + 1))\|g\|$	Client: $5\|g\|$ Server: $(4t + 5)\|g\|$ Gateway: $(4t + 5)\|g\|$
Retrieve Comm. Rounds	10	Client: 2/Server: 6 Gateway: 4

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 $\mathbb{Z}_q$. In Camenisch et al. protocol [5], 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. Our retrieval protocol does not rely on PKI, but the retrieval protocol of Camenisch et al. still requires PKI. In view of this, our retrieval protocol can be implemented easier than Camenisch et al. retrieval protocol.

From Table 1, we can see that our 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 our retrieval protocol is independent of the number of the servers and the threshold.

4 Security Analysis

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

Theorem 1. Assuming that the decisional Diffie-Hellman (DDH) problem [6] is hard over $\{\mathbb{G}, q, g_1\}$ and H is a collision-resistant hash function, then our TPASS protocol P illustrated in Fig. 1 is secure according to Definition 1.

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

Given an 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 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 $\perp$.

When $\mathcal{A}$ queries the Test oracle, the simulator $\mathcal{S}$ chooses a random bit b. When the adversary completes its execution and output a bit b', the simulator can tell whether the adversary succeeds by checking if (1) a single Test query was made regarding some fresh client session, and (2) $b' = b$. Success of the adversary is denoted by event Succ. For any experiment P, we denote $\mathsf{Adv}_{\mathcal{A}}^P = 2 \cdot \mathsf{Pr}[\mathsf{Succ}] - 1$, where $\mathsf{Pr}[\cdot]$ denotes the probability of an event when the simulator interacts with the adversary in accordance with experiment P.

We will use some terminology 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 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.

As shown in the appendix, we have $\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) \leq Q(k)/N + \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$. This completes the proof of the theorem. △

In our retrieval protocol, the client sends out one message $A = g_1^r g_2^{\mathsf{pw}_C}$ only after validating the public parameters $\{\mathbb{G}, q, g_1, g_2\}$. Even if the client communicates with all malicious servers by mistake and the adversary can change the public parameters, our retrieval protocol does not leak the password because r in A is randomly chosen from $\mathbb{Z}_q^*$ by the client.

In addition, in the appendix, we have modified the definition of the security in order to take into account the attack where the adversary attempts to plant a different secret into the user's mind than the secret that he stored earlier. This attack is restricted to online dictionary attack.

5 Conclusion

In this paper, we have presented a practical t-out-of-n TPASS protocol 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. Our protocol is significantly more efficient than existing TPASS protocols. Furthermore, we have provide a rigorous proof of security for our protocol in the standard model.

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

Appendix: Security Proof

Experiment P_1: 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_i, msg_i^*, or msg_S) 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 1. If H is a collision-resistant hash function, $|\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_1}(k)|$ is negligible.

Experiment P_2: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_1 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 $A = g_1^r g_2^{\mathsf{pw}_C}$ is replaced with a random number 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 $g_1^r g_2^{\mathsf{pw}}$. Therefore, we have

Claim 2. $|\mathsf{Adv}_{\mathcal{A}}^{P_1}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_2}(k)|$ 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 $\mathsf{corrupt}(\mathsf{S}_1)$, $a_1 f_1(1)$ 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 $\mathbb{Z}_q^*$.

Although the adversary who has corrupted $\mathsf{S}_2, \mathsf{S}_3, \cdots, \mathsf{S}_t$ can obtain $B_1' = B_1 g_2^{a_2 f_1(2) + \cdots + a_t f_1(t)} = g_1^{r_1} g_2^{\mathsf{pw}_C}$ for B_1, he cannot distinguish $g_1^{r_1} g_2^{\mathsf{pw}_C}$ with $g_1^{r_1} g_2^{\mathsf{pw}}$ for a randomly chosen pw because r_1 is randomly chosen by the simulator. This leads that he cannot distinguish $g_1^{r_1} g_2^{a_1 f_1} = g_1^{r_1} g_2^{\mathsf{pw}_C - a_2 f_1(2) - \cdots - a_t f_1(t)}$ with $g_1^{r_1} g_2^{\mathsf{pw} - a_2 f_1(2) - \cdots - a_t f_1(t)}$ for a randomly chosen pw. Therefore, we have

Claim 3. $|\mathsf{Adv}_{\mathcal{A}}^{P_2}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_3}(k)|$ is negligible.

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 $\mathsf{corrupt}(\mathsf{S}_1)$, 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 4. If the decisional Diffie-Hellman (DDH) problem over $\{\mathbb{G}, q, g_1\}$ is hard, $|\mathsf{Adv}_{\mathcal{A}}^{P_3}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_4}(k)|$ is negligible.

If $|\mathsf{Adv}_{\mathcal{A}}^{P_3}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_4}(k)|$ 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_3. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_4. If the adversary can distinguish the experiments P_3 and P_4 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability.

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 $\mathsf{corrupt}(\mathsf{S}_1)$, 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_4, we have

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

Experiment P_6: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_5 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

$$s_C = (E/C^r)^{h^{-1}} = (E/g_1^{r \sum_{i=1}^{t} c_i})^{h^{-1}} = (E/(g_1^{r \sum_{i=2}^{t} c_i} g_1^{r c_1}))^{h^{-1}}$$

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_5. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_6. If the adversary can distinguish the experiments P_5 and P_6 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability. Therefore, we have

Claim 6. If the decisional Diffie-Hellman (DDH) problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{A}}^{P_5}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_6}(k)|$ is negligible.

In experiment P_6, the 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, all passive adversaries cannot win the game.

The rest of the proof concentrates on the instances invoked by Send oracles.

Experiment P_7: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_6 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 the experiment P_2, we have

Claim 7. $|\mathsf{Adv}_{\mathcal{A}}^{P_6}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_7}(k)|$ is negligible.

Experiment P_8: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_7 except that 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 $\mathbb{Z}_q^*$.

Like the experiment P_3, we have

Claim 8. $|\mathsf{Adv}_{\mathcal{A}}^{P_7}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_8}(k)|$ is negligible.

Experiment P_9: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_8 except that 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 in $\mathsf{msg}_1^* = \langle ID_C, C, D, E_1, F_1\rangle$ is replaced with a random element in the group $\mathbb{G}$.

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_4.

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_8. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_9. If the adversary can distinguish the experiments P_8 and P_9 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability. Therefore, we have

Claim 9. If the decisional Diffie-Hellman (DDH) problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{A}}^{P_8}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_9}(k)|$ is negligible.

Experiment P_{10}: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_9 except that 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 $\mathsf{corrupt}(\mathsf{S}_1)$, 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_9, we have

Claim 10. If the decisional Diffie-Hellman (DDH) problem is hard over $\{\mathbb{G}, q, g_1\}$, $|\mathsf{Adv}_{\mathcal{A}}^{P_9}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_{10}}(k)|$ is negligible.

Experiment P_{11}: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_{10} except that we change the definition of the adversary's success as follows: (1) If the adversary queries $\mathsf{Send}(\mathsf{S}_1, j, C, \mathsf{msg}_C)$ oracle to a fresh client instance C^i for adversarially-generated $\mathsf{msg}_C = \langle ID_C, A' \rangle$ where $A' = g_1^{r'} g_2^{\mathsf{pw}'_C}$, which results in $T' = g_2^{H(S')}$ where $h' = H(ID_C, A', C', D')$, $S' = (E'/C'^{r'})^{h'^{-1}}$, $T' = (F'/D'^{r'})^{h'^{-1}}$ and C', D', E', F' are generated by t honest servers and the honest gateway according the protocol, the simulator halts and the adversary succeeds (let Succ_1 denote this event); (2) If the adversary ever queries $\mathsf{Send}(C, i, \mathsf{msg}_S)$ oracle to a fresh client instance C^i for adversarially-generated $\mathsf{msg}_S = \langle ID_C, C', D', E', F' \rangle$, which results in $\mathsf{acc}_C^i = \mathsf{TRUE}$, the simulator halts and the adversary succeeds (let Succ_2 denote this event); Otherwise the adversary's success is determined as in experiment P_{10}.

The distribution on the adversary's view in experiments P_{10} and P_{11} are identical up to the point when either Succ_1 or Succ_2 occurs. If both Succ_1 and Succ_2 never occur, the distributions on the view are identical. Therefore, we have

Claim 11. $\mathsf{Adv}_{\mathcal{A}}^{P_{10}}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_{11}}(k)$.

Remark. The modified definition of security takes into account the attack where the adversary attempts to plant a different secret into the user's mind than the secret that he stored earlier.

In experiment P_{11}, msg_C, msg_1, msg_1^* in $\mathsf{Execute}$ and 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.

The adversary $\mathcal{A}$ succeeds only if one of the following occurs: (1) Succ_1 occurs; (2) Succ_2 occurs; (3) neither Succ_1 nor Succ_2 occurs, the adversary wins the game by a Test query to a fresh instance C^i.

To evaluate $\mathsf{Pr}[\mathsf{Succ}_1]$ and $\mathsf{Pr}[\mathsf{Succ}_2]$, 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_1 occurs, the adversary can conclude that the password used by the client is pw'_C. Therefore, the probability $\mathsf{Pr}[\mathsf{Succ}_1] = Q_1(k)/N$, 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 $\mathsf{Pr}[\mathsf{Succ}_2] = Q_2(k)/N$, 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 t servers twice to get two responses $\mathsf{msg}_S = \langle ID_C, C, D, E, F \rangle$ and $\mathsf{msg}'_S = \langle ID_C, C', D', E', F' \rangle$. Then the adversary $\mathcal{A}$ sends to the client a forged message $\mathsf{msg}_S = \langle ID_C, C'/C, D'/D, g_2^{s^*h^*}E'/E, g_2^{H(g_2^{s^*}h^*)}F'/F \rangle$, where $E'/E = g_1^{s(h'-h)}(C'/C)^r$, $F'/F = g_1^{H(g_1^s)(h'-h)}(D'/D)^r$, $h = H(ID_C, A, C, D)$, $h' = H(ID_C, A, C', D')$, $h^* = H(ID_C, A, C'/C, D'/D)$ 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 $\mathsf{Pr}[\mathsf{Succ}_2]$ is negligible in this case.

In summary, $\mathsf{Pr}[\mathsf{Succ}_1 \vee \mathsf{Succ}_2] = Q(k)/N$, where $Q(k)$ denotes the number of on-line attacks.

In experiment P_{11}, the adversary's probability of success when neither Succ_1 nor Succ_2 occurs is 1/2. The preceding discussion implies that

$$Pr_{\mathcal{A}}^{P_{11}}[\mathsf{Succ}] \leq Q(k)/N + 1/2 \cdot (1 - Q(k)/N)$$

and thus the adversary's advantage in experiment P_{11}

$$\begin{aligned} \mathsf{Adv}_{\mathcal{A}}^{P_{11}}(k) &= 2Pr_{\mathcal{A}}^{P_{11}}[\mathsf{Succ}] - 1 \\ &\leq 2Q(k)/N + 1 - Q(k)/N - 1 \\ &= Q(k)/N \end{aligned}$$

The sequence of claims proved above show that

$$\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_{11}}(k) + \varepsilon(k) \leq Q(k)/N + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$. This completes the proof of the theorem.

References

1. Bagherzandi, A., Jarecki, S., Saxena, N., Lu, Y.: Password-protected secret sharing. In: ACM CCS 2011, pp. 433–444 (2011)

2. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Eurocrypt 2000, pp. 139–155 (2000)
3. Brainard, J., Juels, A., Kaliski, B., Szydlo, M.: Nightingale: a new two-server approach for authentication with short secrets. In: 12th USENIX Security Symposium, pp. 201–213 (2003)
4. Camenisch, J., Lysyanskaya, A., Neven, G.: Practical yet universally composable two-server password-authenticated secret sharing. In: ACM CCS 2012, pp. 525–536 (2012)
5. Camenisch, J., Lysyanskaya, A., Lysyanskaya, A., Neven. G.: Memento: How to reconstruct your secrets from a single password in a hostile environment. In: Crypto 2014, pp. 256–275 (2014)
6. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **32**(2), 644–654 (1976)
7. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985)
8. Ford, W., Kaliski, B.S.: Server-assisted generation of a strong secret from a password. In: 5th IEEE International Workshop on Enterprise Security (2000)
9. Jablon, D.: Password authentication using multiple servers. In: CT-RSA 2001, pp. 344–360 (2001)
10. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Eurocrypt 2001, pp. 457–494 (2001)
11. Katz, J., MacKenzie, P., Taban, G., Gligor, V.: Two-server password-only authenticated key exchange. In: ACNS 2005, pp. 1–16 (2005)
12. MacKenzie, P., Shrimpton, T., Jakobsson, M.: Threshold password-authenticated key exchange. J. Cryptol. **19**(1), 27–66 (2006)
13. Di Raimondo, M., Gennaro, R.: Provably secure threshold password-authenticated key exchange. J. Comput. Syst. Sci. **72**(6), 978–1001 (2006)
14. RSA, The Security Division of EMC: New RSA innovation helps thwart "smash-and-grab" credential theft. Press release (2012)
15. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
16. Yi, X., Ling, S., Wang, H.: Efficient two-server password-only authenticated key exchange. IEEE Trans. Parallel Distrib. Syst. **24**(9), 1773–1782 (2013)
17. Yi, X., Hao, F., Bertino, E.: ID-based two-server password-authenticated key exchange. In: ESORICS 2014, pp. 257–276 (2014)