ID-Based Two-Server Password-Authenticated Key Exchange

Xun Yi[1], Feng Hao[2], and Elisa Bertino[3]

[1] School of CS and IT, RMIT University, Australia
[2] School of Computing Science, Newcastle University, UK
[3] Department of Computer Science, Purdue University, USA

Abstract. In two-server password-authenticated key exchange (PAKE) protocol, a client splits its password and stores two shares of its password in the two servers, respectively, and the two servers then cooperate to authenticate the client without knowing the password of the client. In case one server is compromised by an adversary, the password of the client is required to remain secure. In this paper, we present a compiler that transforms any two-party PAKE protocol to a two-server PAKE protocol. This compiler is mainly built on two-party PAKE and identity-based encryption (IBE), where the identities of the two servers are used as their public keys. By our compiler, we can construct a two-server PAKE protocol which achieves implicit authentication with only two communications between the client and the servers. As long as the underlying two-party PAKE protocol and IBE scheme have provable security without random oracles, the two-server PAKE protocol constructed by our compiler can be proven to be secure without random oracles.

Keywords: Password-authenticated key exchange, identity-based encryption, Diffie-Hellman key exchange, Decisional Diffie-Hellman problem.

1 Introduction

Bellovin and Merritt [4] were the first to introduce password-based authenticated key exchange (PAKE), where two parties, based only on their knowledge of a password, establish a cryptographic key by exchange of messages. A PAKE protocol has to be immune to on-line and off-line dictionary attacks. In an off-line dictionary attack, an adversary exhaustively tries all possible passwords in a dictionary in order to determine the password of the client on the basis of the exchanged messages. In on-line dictionary attack, an adversary simply attempts to login repeatedly, trying each possible password. By cryptographic means only, none of PAKE protocols can prevent on-line dictionary attacks. But on-line attacks can be stopped simply by setting a threshold to the number of login failures.

Since Bellovin and Merritt [4] introduced the idea of PAKE, numerous PAKE protocols have been proposed. In general, there exist two kinds of PAKE settings, one assumes that the password of the client is stored in a single server and another assumes that the password of the client is distributed in multiple servers.

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part II, LNCS 8713, pp. 257–276, 2014.

PAKE protocols in the single-server setting can be classified into three categories as follows.

- *Password-only PAKE*: Typical examples are the "encrypted key exchange" (EKE) protocols given by Bellovin and Merritt [4], where two parties, who share a password, exchange messages encrypted by the password, and establish a common secret key. The formal model of security for PAKE was firstly given in [3, 7]. Based on the security model, PAKE protocols [1, 2, 9, 10, 17, 21, 23] have been proposed and proved to be secure.
- *PKI-based PAKE*: PKI-based PAKE protocol was first given by Gong et al. [18], where the client stores the server's public key in addition to share a password with the server. Halevi and Krawczyk [19] were the first to provide formal definitions and rigorous proofs of security for PKI-based PAKE.
- *ID-based PAKE*: ID-based PAKE protocols were proposed by Yi et al. [33, 34], where the client needs to remember a password in addition to the identity of the server, whereas the server keeps the password in addition to a private key related to its identity. ID-based PAKE can be thought as a trade-off between password-only and PKI-based PAKE.

In the single-server setting, all the passwords necessary to authenticate clients are stored in a single server. If the server is compromised, due to, for example, hacking or even insider attacks, passwords stored in the server are all disclosed. To address this problem, the multi-server setting for PAKE was first suggested in [15, 20], where the password of the client is distributed in n servers.

PAKE protocols in the multi-server setting can be classified into two categories as follows.

- *Threshold PAKE*: The first PKI-based threshold PAKE protocol was given by Ford and Kaliski [15], where n severs, sharing the password of the client, cooperate to authenticate the client and establish independent session keys with the client. As long as $n - 1$ or fewer servers are compromised, their protocol remains secure. Jablon [20] gave a protocol with similar functionality in the password-only setting. MacKenzie et al. proposed a PKI-based threshold PAKE protocol which requires only t out of n servers to cooperate in order to authenticate the client. Their protocol remains secure as long as $t - 1$ or fewer servers are compromised. Di Raimondo and Gennaro [27] suggested a password-only threshold PAKE protocol which requires fewer than $1/3$ of the servers to be compromised.
- *Two-server PAKE*: Two-server PKI-based PAKE was first given by Brainard [8], where two servers cooperate to authenticate the client and the password remains secure if one server is compromised. A variant of the protocol was later proved to be secure in [28]. The first two-server password-only PAKE protocol was given by Katz et al. [24], in which two servers symmetrically contribute to the authentication of the client. The protocol in the server side can run in parallel. Efficient protocols [30–32, 22] were later proposed, where the front-end server authenticates the client with the help of the back-end server and only the front-end server establishes a session key with the

client. These protocols are asymmetric in the server side and have to run in sequence. Recently, Yi et al. gave a symmetric solution [35] which is even more efficient than asymmetric protocols [22, 30–32].

In this paper, we will consider the two-server setting for PAKE only. A typical example is the two-server PAKE protocol given by Katz et al. [24], which is built upon the two-party PAKE protocol (i.e., the KOY protocol [23]), where two parties, who share a password, exchange messages to establish a common secret key. Their basic two-server protocol is secure against a passive (i.e., "honest-but-curious") adversary who has access to one of the servers throughout the protocol execution, but cannot cause this server to deviate from its prescribed behavior. In [24], Katz et al. also showed how to modify their basic protocol so as to achieve security against an active adversary who may cause a corrupted server to deviate arbitrarily from the protocol. The core of their protocol is the KOY protocol. The client looks like running two KOY protocols with two servers in parallel. However, each server must perform a total of roughly 70 exponentiations (i.e., each server's work is increased by a factor of roughly 6 as compared to the basic protocol [24]). In general, this construction implies a compiler which transfers a two-party PAKE protocol to a two-server PAKE protocol. The compiler employs a two-party PAKE protocol between the client and the servers.

Our Contribution. In this paper, we propose a new compiler to construct a two-server PAKE protocol with any two-party PAKE protocol. Our compiler employs the two-party PAKE protocol between two servers when they authenticate the client.

To achieve the goal, our compiler needs an identity-based encryption (IBE) scheme to protect the messages (containing the password information) from the client to the two servers. The basic idea is: first of all, the client splits its password into two shares and each server keeps one share of the password in addition to a private key related to its identity. In key exchange, the client sends to each server one share of the password encrypted according to the identity of the server. From the client messages, both servers can derive the same one-time password, by which the two servers can run a two-party PAKE protocol to authenticate the client. Our compiler also needs a public key encryption scheme for the servers to protect the messages (containing the password information) from the servers to the client. The one-time public key is generated by the client and sent to the servers along with the password information in the first phase.

In an IBE scheme, the decryption key of a server is usually generated by a Private Key Generator (PKG). Therefore the PKG can decrypt any messages encrypted with the identity of the server. As mentioned in [5], using standard techniques from threshold cryptography, the PKG can be distributed so that the master-key is never available in a single location. In order to prevent a malicious PKG from decrypting the password information encrypted with the identity of a server, a strategy is to employ multiple PKGs which cooperate to generate the decryption key for the server. As long as one of the PKGs is honest to follow the protocol, the decryption key for the server is known only to the server. Since we

can assume that the two servers in two-server PAKE never collude, we can also assume that at least one of the PKGs do not collude with other PKGs.

We define an ID-based security model for two-server PAKE. Unlike the ID-based security model for single-server PAKE defined in [33, 34], an adversary can compromise one of the two servers, each having one share of the password. Based on our security model, we provide a rigorous proof of security for our compiler. Our compiler does not rely on the random oracle model as long as the underlying primitives themselves do not rely on it. For example, by using the KOY protocol [23] and the Waters IBE scheme [29] and the Cramer-Shoup public key encryption scheme [12], our compiler can construct a two-server PAKE with provable security in the standard model.

We also compare our ID-based two-server PAKE protocol with the Katz et al.'s two-server PAKE protocol [24] with provable security in the standard model. The Katz et al.'s protocol is password-only, where the client needs to remember the password only and refer to common public parameters, and each server, having a public and private key pair, and keeps a share of the password. Our protocol is identity-based, where the client needs to remember the password in addition to the meaningful identities of the two servers, and refer to common public parameters, including the master public key, and each server, having a private key related to his identity, keeps a share of the password.

In terms of setting, the Katz et al.s protocol is superior to our protocol. However, in the Katz et al.'s protocol, each server performs approximately six times the amount of the work as the KOY protocol, whereas in our protocol, each server performs the same amount of work as the KOY protocol in addition to one IBE decryption and one public key encryption. In addition, the Katz et al.'s protocol needs three communications between the client and the servers to achieve implicit authentication, whereas our protocol achieves implicit authentication with only two communications between the client and the servers.

Organization. In Section 2, we introduce our security model for ID-based two-server PAKE. In Section 3, we present our ID-based two-server PAKE compiler. After that, in Section 4, a sketch of security proof for our protocol is provided. We conclude this paper in Section 5.

2 Definitions

A formal model of security for two-server PAKE was given by Katz et al. [24] (based on the MacKenzie et al.'s model for PKI-based PAKE [26]). Boneh and Franklin [5] defined chosen ciphertext security for IBE under chosen identity attack. Combining the two models, we give an ID-based model for two-server PAKE.

Participants, Initialization and Passwords. An ID-based PAKE protocol involves three kinds of protocol participants: (1) A set of clients (denoted as Client), each of which requests services from servers on the network; (2) A set of servers (denoted as Server), each of which provides services to clients on the network; (3) A group of Private Key Generators (PKGs), which generate public parameters and corresponding private keys for servers.

We assume that $\mathsf{ClientServerTriple}$ is the set of triples of the client and two servers, where the client is authorized to use services provided by the two servers, $\mathsf{Client} \bigcap \mathsf{Server} = \emptyset$, $\mathsf{User} = \mathsf{Client} \bigcup \mathsf{Server}$, and any $\mathsf{PKG} \notin \mathsf{User}$. It is obvious that $\mathsf{ClientServerTriple} \subseteq \mathsf{Client} \times \mathsf{Server} \times \mathsf{Server}$.

Prior to any execution of the protocol, we assume that an initialization phase occurs. During initialization, the PKGs cooperate to generate public parameters for the protocol, which are available to all participants, and private keys for servers, which are given to the appropriate servers. The user may keep the public parameter in a personal device, such as a smart card or a USB flash drive. When the PKGs generate the private key for a server, each PKG generates and sends a private key component to the server via a secure channel. The server then derives its private key by combining all private key components from all PKGs. **We assume that at least one of PKGs is honest to follow the protocol**. Therefore, the private key of the server is known to the server only.

For any triple $(C, A, B) \in \mathsf{ClientServerTriple}$, we assume that the client C chooses its password pw_C independently and uniformly at random from a "dictionary" $\mathcal{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ of size N, where $\mathcal{D} \subset \mathbb{Z}_p$, N is a fixed constant which is independent of any security parameter, and p is a large prime. The password is then split into two shares $\mathsf{pw}_{C,A}$ and $\mathsf{pw}_{C,B}$ and stored at the two servers A and B, respectively, for authentication. **We assume that the two servers never collude to determine the password of the client**. The client C needs to remember pw_C to log into the servers A and B.

For simplicity, we assume that each client C shares its password pw_C with exactly two servers A and B. In this case, we say that servers A and B are associated with C. A server may be associated with multiple clients.

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 (please refer to [3]) 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, pid_U^i and sk_U^i are variables containing the session identity, partner identity, and session key for an instance U^i, respectively. Computation of the session key is, of course, the ultimate goal of the protocol. 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. The partner identity denotes the identity of the user with whom U^i believes it is interacting. For a client C, sk_C^i consists of a pair $(\mathsf{sk}_{C,A}^i, \mathsf{sk}_{C,B}^i)$, which are the two keys shared with servers A and B, respectively.

- 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. In our case, acceptance means that the instance is sure that it has established a session key with its intended partner; thus, when an instance U^i has been accepted, sid_U^i, pid_U^i and sk_U^i are 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:

- $\mathsf{Send}(C, i, A, B, M)$ – This sends message M to a client instance C^i, supposedly from two servers A and B. 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 $\mathsf{sid}_C^i, \mathsf{pid}_C^i, \mathsf{acc}_C^i$, and term_C^i. This oracle call models the active attack to a protocol. If M is empty, this query represents a prompt for C to initiate the protocol.
- $\mathsf{Send}(S, i, U, M)$ – This sends message M to a server instance S^i, supposedly from a user U (either a client or a server). Assuming $\mathsf{term}_S^i = \mathsf{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of S^i (i.e., the message sent by the instance) is given to the adversary, who receives the updated values of $\mathsf{sid}_S^i, \mathsf{pid}_S^i, \mathsf{acc}_S^i$, and term_S^i. If S is corrupted, the adversary also receives the entire internal state of S. This oracle call also models the active attack to a protocol.
- $\mathsf{Execute}(C, i, A, j, B, k)$ – If the client instance C^i and the server instances A^j and B^k have not yet been used (where $(C, A, B) \in \mathsf{ClientServerTriple}$), 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, pid, acc, and term for client and server instances, at each step of protocol execution. In addition, if $S \in \{A, B\}$ is corrupted, the adversary is given the entire internal state of S.
- $\mathsf{Corrupt}(S)$ – This sends the private key of the server S in addition to all password information 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, 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.
- $\mathsf{Reveal}(U, U', i)$ – This outputs the current value of session key $\mathsf{sk}^i_{U,U'}$ held by instance U^i if $\mathsf{acc}^i_U = \mathsf{TRUE}$, where $U' \in \mathsf{pid}^i_U$. This oracle call models possible leakages of session keys due to, for example, improper erasure of session keys after use, compromise of a host computer, or cryptanalysis.
- $\mathsf{Test}(U, U', i)$ – This oracle does not model any real-world capability of the adversary, but is instead used to define security. Assume $U' \in \mathsf{pid}^i_U$, if $\mathsf{acc}^i_U = \mathsf{TRUE}$, a random bit b is generated. If $b = 0$, the adversary is given $\mathsf{sk}^i_{U,U'}$, and if $b = 1$ the adversary is given a random session key. The adversary is allowed only a single Test query, at any time during its execution.

Partnering. Let $(C, A, B) \in \mathsf{ClientServerTriple}$. For the client instance C^i, let $\mathsf{sid}^i_C = (\mathsf{sid}^i_{C,A}, \mathsf{sid}^i_{C,B})$, where $\mathsf{sid}^i_{C,A}$ (resp., $\mathsf{sid}^i_{C,B}$) denotes the ordered sequence of messages sent to / from the client C and the server A (resp., server B). For the server instance A^j, let $\mathsf{sid}^j_A = (\mathsf{sid}^j_{A,C}, \mathsf{sid}^j_{A,B})$, where $\mathsf{sid}^i_{A,C}$ denotes the ordered sequence of messages sent to / from the server A and the client C, and $\mathsf{sid}^j_{A,B}$ denote the ordered sequence of message sent to / from the server A and the server B. We say that instances C^i and A^j are partnered if (1) $\mathsf{sid}^i_{C,A} = \mathsf{sid}^j_{A,C} \neq \mathsf{NULL}$ and (2) $A \in \mathsf{pid}^i_C$ and $C \in \mathsf{pid}^j_A$. We say that instances A^j and B^k are partnered if (1) $\mathsf{sid}^j_{A,B} = \mathsf{sid}^k_{B,A} \neq \mathsf{NULL}$ and (2) $A \in \mathsf{pid}^k_B$ and $B \in \mathsf{pid}^j_A$.

Correctness. To be viable, a key exchange protocol must satisfy the following notion of correctness: If a client instance C^i and server instances A^j and B^k runs an honest execution of the protocol with no interference from the adversary, then $\mathsf{acc}^i_C = \mathsf{acc}^j_A = \mathsf{acc}^k_B = \mathsf{TRUE}$, and $\mathsf{sk}^i_{C,A} = \mathsf{sk}^j_{A,C}$, $\mathsf{sk}^i_{C,B} = \mathsf{sk}^k_{B,C}$ and $\mathsf{sk}^i_{C,A} \neq \mathsf{sk}^i_{C,B}$.

Freshness. To formally define the adversary's success we need to define a notion of freshness for a session key, where freshness of a key is meant to indicate that the adversary does not trivially know the value of the session key. We say a session key $\mathsf{sk}^i_{U,U'}$ is fresh if (1) both U and U' are not corrupted; (2) the adversary never queried $\mathsf{Reveal}(U, U', i)$; (3) the adversary never queried $\mathsf{Reveal}(U', U, j)$ where U^i and U'^j are partnered.

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 $\mathsf{Test}(U, U', i)$ to a fresh instance U^i, with $\mathsf{acc}^i_U = \mathsf{TRUE}$ 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 $\mathsf{Adv}^P_{\mathcal{A}}(k) = 2 \cdot \mathsf{Pr}[\mathsf{Succ}] - 1$, 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 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 two-server protocol for PAKE 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 $\mathsf{Adv}_{\mathcal{A}}^{P}(k) \leq Q(k)/N + \varepsilon(k)$.

3 Our Compiler for Two-Server PAKE Protocol

3.1 Description of Our Compiler

In this section, we present our compiler transforming any two-party PAKE protocol P to a two-server PAKE protocol P'. Besides P, we need an identity-based encryption scheme (IBE) as our cryptographic building block. If we remove authentication elements from our compiler, our key exchange protocol is essentially the Diffie-Hellman key exchange protocol [13]. A high-level description of our compiler is given in Figure 1, in which the client C and two servers A and B establish two authenticated keys, respectively.

We present the protocol by describing initialization and execution.

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

Parameter Generation: On input k, (1) m PKGs cooperate to run Setup^P of the two-party PAKE protocol P to generate system parameters, denoted as params^P. (2) m PKGs cooperate to run Setup^E of the IBE scheme to generate public system parameters for the IBE scheme, denoted as params^E, and the secret $\mathsf{master\text{-}key}^E$. Assume that $\mathcal{G}$ is a generator of IBE plaintext group $\mathbb{E}$ with an order n. (3) m PKGs cooperate to choose a large cyclic group $\mathbb{G}$ with a prime order q and two generators g_1, g_2, and two hash functions, $H_1 : \{0,1\}^* \rightarrow Z_n^*$ and $H_2 : \{0,1\}^* \rightarrow Z_q^*$, from a collision-resistant hash family. The public system parameters for the protocol P' is $\mathsf{params} = \mathsf{params}^{P,E} \bigcup \{\mathbb{E}, \mathcal{G}, n, \mathbb{G}, q, g_1, g_2, H_1, H_2\}$ and the secret $\mathsf{master\text{-}key}^E$ is secretly shared by the PKGs in a manner that any coalition of PKGs cannot determine $\mathsf{master\text{-}key}^E$ as long as one of the PKGs is honest to follow the protocol.

Remark. Taking the Waters' IBE scheme [29] for example, m PKG agree on randomly chosen $\mathcal{G}, \mathcal{G}_2 \in \mathbb{G}$ and each PKG randomly chooses $\alpha_i \in \mathbb{Z}_p$ and broadcast $\mathcal{G}^{\alpha_i}$ with a zero-knowledge proof of knowing α_i and a signature. Then we can set $\mathcal{G}_1 = \mathcal{G}^{\sum_i \alpha_i}$ as the public master key and the secret $\mathsf{master\text{-}key}^E = \mathcal{G}_2^{\sum_i \alpha_i}$. The secret master key is privately shared among m PKGs and unknown to anyone even if $m-1$ PKGs maliciously collude.

Key Generation: On input the identity S of a server $S \in \mathsf{Server}$, params^E, and the secret sharing $\mathsf{master\text{-}key}^E$, PKGs cooperate to run $\mathsf{Extract}^E$ of the IBE scheme and generate a private (decryption) key for S, denoted as d_S, in a manner that any coalition of PKGs cannot determine d_S as long as one of the PKGs is honest to follow the protocol.

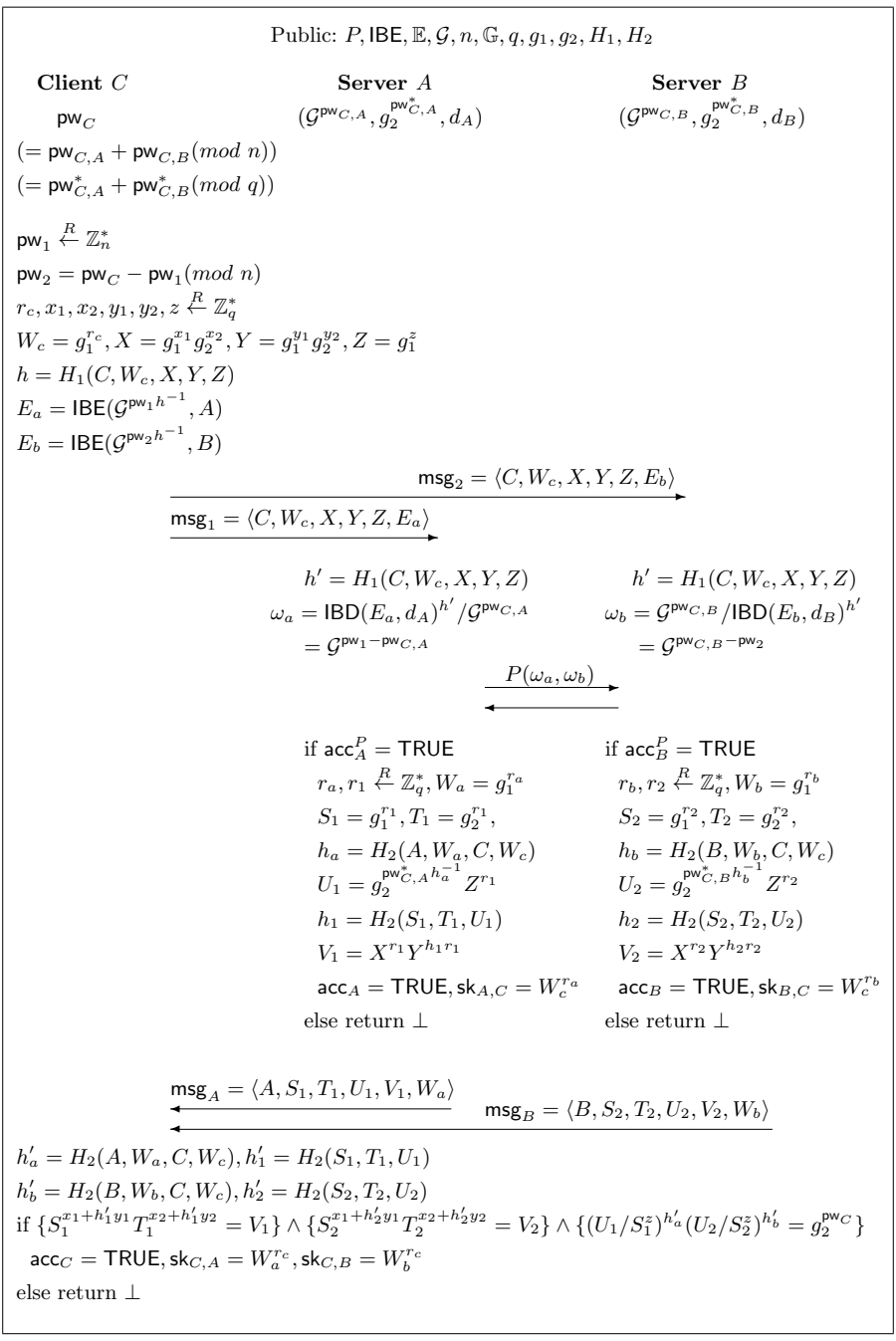

Fig. 1. Two-Server PAKE Protocol P'

Remark. In the Waters' IBE scheme with m PKG , each PKG computes one component of the private key for a server S, i.e., $(\mathcal{G}_2^{\alpha_i} H(S)^{r_i}, \mathcal{G}^{r_i})$, where H is the Waters' hash function, and sends it to the server via a secure channel. Combining all components, the server can construct its private key $d_S = (\mathcal{G}_2^{\sum_i \alpha_i} H(S)^{\sum_i r_i}, \mathcal{G}^{\sum_i r_i})$, which is known to the server only even if $m-1$ PKGs maliciously collude. In addition, the identity of a server is public, meaningful, like an e-mail address, and easy to remember or keep. Anyone can write down the identity of a server on a note.

Password Generation: On input a triple $(C, A, B) \in \mathsf{Client\ ServerTriple}$, a string pw_C, the password, is uniformly drawn from the dictionary $\mathcal{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ by the client C, and randomly split into $\mathsf{pw}_{C,A}$ and $\mathsf{pw}_{C,B}$ such that $\mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} = \mathsf{pw}_C (mod\ n)$, and $\mathsf{pw}^*_{C,A}$ and $\mathsf{pw}^*_{C,B}$ such that $\mathsf{pw}^*_{C,A} + \mathsf{pw}^*_{C,B} = \mathsf{pw}_C (mod\ q)$, and stored in the servers A and B, respectively. We implicitly assume that $N < min(n, q)$, which will certainly be true in practice.

Protocol Execution. Given a triple $(C, A, B) \in \mathsf{Client\ ServerTriple}$, the client C (knowing its password pw_C) runs the protocol P' with the two servers A (knowing $\mathcal{G}^{\mathsf{pw}_{C,A}}$, $g_2^{\mathsf{pw}^*_{C,A}}$ and its private key d_A) and B (knowing $\mathcal{G}^{\mathsf{pw}_{C,B}}$, $g_2^{\mathsf{pw}^*_{C,B}}$ and its private key d_B) to establish two session keys, respectively, as shown in Figure 1.

At first, the client randomly chooses pw_1 from Z_n^* and computes $\mathsf{pw}_2 = \mathsf{pw}_C - \mathsf{pw}_1 (mod\ n)$. Next the client C randomly chooses $r_c, x_1, x_2, y_1, y_2, z$ from Z_q^* and computes $W_c = g_1^{r_c}, X = g_1^{x_1} g_2^{x_2}, Y = g_1^{y_1} g_2^{y_2}, Z = g_1^z$, $h = H_1(C, W, X, Y, Z)$, where (X, Y, Z) is one-time public encryption key and (x_1, x_2, y_1, y_2, z) is one-time private decryption key of the Cramer-Shoup public key encryption scheme [12].

Next, according to the identities of the two servers A and B, the client C performs the identity-based encryptions $E_a = \mathsf{IBE}(\mathcal{G}^{\mathsf{pw}_1 h^{-1}}, A), E_b = \mathsf{IBE}(\mathcal{G}^{\mathsf{pw}_2 h^{-1}}, B)$.

Then, the client sends $\mathsf{msg}_1 = \langle C, W, X, Y, Z, E_a \rangle$ and $\mathsf{msg}_2 = \langle C, W, X, Y, Z, E_b \rangle$ to the two servers A and B, respectively.

After receiving $\mathsf{msg}_1 = \langle C, W, X, Y, Z, E_a \rangle$ from C, the server A computes $h' = H_1(C, W, X.Y, Z)$, $\omega_a = \mathsf{IBD}(E_a, d_A)^{h'} / \mathcal{G}^{\mathsf{pw}_{C,A}} = \mathcal{G}^{\mathsf{pw}_1 - \mathsf{pw}_{C,A}}$, where IBD denotes identity-based decryption.

After receiving $\mathsf{msg}_2 = \langle C, W, X.Y, Z, E_b \rangle$ from C, the server B computes $\omega_b = \mathcal{G}^{\mathsf{pw}_{C,B}} / \mathsf{IBD}(E_a, d_B)^{h'} = \mathcal{G}^{\mathsf{pw}_{C,B} - \mathsf{pw}_2}$, where $h' = H_1(C, W, X, Y, Z)$.

Because $\mathsf{pw}_C = \mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} (mod\ n)$ and $\mathsf{pw}_C = \mathsf{pw}_1 + \mathsf{pw}_2 (mod\ n)$, we have $\mathsf{pw}_1 - \mathsf{pw}_{C,A} = \mathsf{pw}_{C,B} - \mathsf{pw}_2 (mod\ n)$ and thus $\omega_a = \omega_b$.

Using ω_a and ω_b as one-time password, the servers A and B run a two-party PAKE protocol P to establish a session key. If the server A accepts the session key as an authenticated key according to P (i.e., $\mathsf{acc}_A^P = \mathsf{TRUE}$), it randomly chooses two integers r_a, r_1 from $\mathbb{Z}_q^*$ and computes $W_a = g_1^{r_a}, S_1 = g_1^{r_1}, T_1 = g_2^{r_1}$, $h_a = H_2(A, W_a, C, W_c), U_1 = g_2^{\mathsf{pw}^*_{C,A} h_a^{-1}} Z^{r_1}$, $h_1 = H_2(S_1, T_1, U_1), V_1 = X^{r_1} Y^{h_1 r_1}, \mathsf{sk}_{A,C} = W_c^{r_a}$, where $\mathsf{sk}_{A,C}$ is the session key between A and C and

(S_1, T_1, U_1, V_1) is the Cramer-Shoup encryption of $g_2^{\mathsf{pw}^*_{C,A} h_a^{-1}}$. Then the server A sets $\mathsf{acc}_A = \mathsf{TRUE}$ and replies to the client C with $\mathsf{msg}_A = \langle A, S_1, T_1, U_1, V_1, W_a \rangle$.

If the server B accepts the session key as an authenticated key according to P (i.e., $\mathsf{acc}_B^P = \mathsf{TRUE}$), it randomly chooses two integers r_b, r_2 from $\mathbb{Z}_q^*$ and computes $W_b = g_1^{r_b}, S_2 = g_1^{r_2}, T_2 = g_2^{r_2}$, $h_b = H_2(A, W_b, C, W_c), U_2 = g_2^{\mathsf{pw}^*_{C,B} h_b^{-1}} Z^{r_2}$, $h_2 = H_2(S_2, T_2, U_2), V_2 = X^{r_2} Y^{h_2 r_2}, \mathsf{sk}_{B,C} = W_c^{r_b}$, where $\mathsf{sk}_{B,C}$ is the session key between B and C and (S_2, T_2, U_2, V_2) is the Cramer-Shoup encryption of $g_2^{\mathsf{pw}^*_{C,B} h_b^{-1}}$. Then the server B sets $\mathsf{acc}_B = \mathsf{TRUE}$ and replies to the client C with $\mathsf{msg}_B = \langle B, S_2, T_2, U_2, V_2, W_b \rangle$.

Finally, after the client C receives msg_A and msg_B, it computes $h'_a = H_2(A, W_a, C, W_c), h'_1 = H_2(A, S_1, T_1, U_1)$, $h'_b = H_2(B, W_b, C, W_c), h'_2 = H_2(B, S_2, T_2, U_2)$, and check if

$$S_1^{x_1 + h'_1 y_1} T_1^{x_2 + h'_1 y_2} = V_1, S_2^{x_1 + h'_2 y_1} T_2^{x_2 + h'_2 y_2} = V_2, (U_1 / S_1^z)^{h'_a} (U_2 / S_2^z)^{h'_b} = g_2^{\mathsf{pw}_C}.$$

If so, the client C sets $\mathsf{acc}_C = \mathsf{TRUE}$ and computes two session keys $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$.

3.2 Correctness, Explicit Authentication, and Efficiency

Correctness. Assume that a client instance C^i and server instances A^j and B^k runs an honest execution of the protocol P' with no interference from the adversary and the two-party PAKE P has the correctness property.

With reference to Figure 1, the server instances A^j and B^k are able to derive the same one-time password ω_a $(= \omega_b)$. Because P has the correctness property, after running P based on ω_a and ω_b, the server instances A^j and B^k accept the established session key as an authenticated key. This indicates that the client C has provided a correct password pw_C. Next, the server instances A^j and B^k compute the session keys with the client C, i.e., $\mathsf{sk}_{A,C} = W_c^{r_a}$ and $\mathsf{sk}_{B,C} = W_c^{r_b}$, and let $\mathsf{acc}_A^j = \mathsf{TRUE}$ and $\mathsf{acc}_B^k = \mathsf{TRUE}$.

With reference to Figure 1, we have $h'_a = h_a, h'_1 = h_1, h'_b = h_b, h'_2 = h_2$, and

$$\begin{aligned} S_1^{x_1 + h'_1 y_1} T_1^{x_2 + h'_1 y_2} &= g_1^{r_1 (x_1 + h_1 y_1)} g_2^{r_1 (x_2 + h_1 y_2)} \\ &= (g_1^{x_1} g_2^{x_2})^{r_1} (g_1^{y_1} g_2^{y_2})^{r_1 h_1} = X^{r_1} Y^{h_1 r_1} = V_1, \end{aligned}$$

$$U_1 / S_1^z = g_2^{\mathsf{pw}^*_{C,A} h_a^{-1}} (g_1^z)^{r_1} / (g_1^{r_1})^z = g_2^{\mathsf{pw}^*_{C,A} h'^{-1}_a},$$

$$\begin{aligned} S_2^{x_1 + h'_2 y_1} T_2^{x_2 + h'_2 y_2} &= g_1^{r_2 (x_1 + h_2 y_1)} g_2^{r_2 (x_2 + h_2 y_2)} \\ &= (g_1^{x_1} g_2^{x_2})^{r_2} (g_1^{y_1} g_2^{y_2})^{r_2 h_2} = X^{r_2} Y^{h_2 r_2} = V_2, \end{aligned}$$

$$U_2 / S_2^z = g_2^{\mathsf{pw}^*_{C,B} h_b^{-1}} (g_1^z)^{r_2} / (g_1^{r_2})^z = g_2^{\mathsf{pw}^*_{C,B} h'^{-1}_b},$$

$$(U_1/S_1^z)^{h'_a}(U_2/S_2^z)^{h'_b} = g_2^{\mathsf{pw}^*_{C,A}} g_2^{\mathsf{pw}^*_{C,B}} = g_2^{\mathsf{pw}_C}.$$

If the three checks succeed, the client C computes two session keys, i.e., $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$, and lets $\mathsf{acc}_C^i = \mathsf{TRUE}$. Since $W_c = g_1^{r_c}, W_a = g_1^{r_a}, W_b = g_1^{r_b}$, we have $\mathsf{sk}_{C,A} = W_a^{r_c} = g_1^{r_a r_c} = W_c^{r_a} = \mathsf{sk}_{A,C}$ and $\mathsf{sk}_{C,B} = W_b^{r_c} = g_1^{r_b r_c} = W_c^{r_b} = \mathsf{sk}_{B,C}$. In addition, because r_a, r_b are chosen randomly, the probability of $\mathsf{sk}_{C,A} = \mathsf{sk}_{C,B}$ is negligible. Therefore, our protocol has the correctness property.

Explicit Authentication. By running the two-party PAKE protocol P based on w_a (derived by $\mathsf{pw}_{C,A}$) and w_b (derived by $\mathsf{pw}_{C,B}$), the two servers A and B can verify if the client C provides a password pw_C such that $\mathsf{pw}_C = \mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} (mod\ n)$. In addition, by checking that $(U_1/S_1^z)^{h'_a}\ (U_2/S_2^z)^{h'_b} = g_2^{\mathsf{pw}_C}$ (involving pw_C), the client C can verify if the two servers provide two shares of the password, $\mathsf{pw}^*_{C,A}$ and $\mathsf{pw}^*_{C,B}$, such that $\mathsf{pw}_C = \mathsf{pw}^*_{C,A} + \mathsf{pw}^*_{C,B} (mod\ q)$. This shows that when $\mathsf{acc}_A^j = \mathsf{TRUE}$, the server A knows that its intended client C and server B are authentic, and when $\mathsf{acc}_C^i = \mathsf{TRUE}$, the client C knows that its intended servers A and B are authentic. Our protocol achieves the implicit authentication by only two communications between the client and the servers. Using standard techniques, however, it is easy to add explicit authentication to any protocol achieving implicit authentication.

Efficiency. The efficiency of our protocol depends on performance of the underlying two-party PAKE protocol and IBE scheme. Suppose that our compiler uses the KOY PAKE protocol [23] and the Waters IBE scheme [29] as cryptographic building blocks, the performance comparison of the Katz et al. two-server PAKE protocol [24] (secure against active adversary) and our protocol can be shown in Table 1.

In Table 1, Exp., Sign. and Pairing for computation (comp.) represent the computation complexities of a modular exponentiation, a signature generation and a pairing, respectively. Exp. and Sign. in communication (comm.) denote

Table 1. Performance Comparison of Katz et al. Protocol and Our Protocol

	Katz et al. Protocol [24]	Our Protocol
Public Keys	Client: None Sever A: Public Key pk_A Sever B: Public Key pk_B	Client: None Server A: A Server B: B
Private Keys	Client: pw_C Sever A: $\mathsf{pw}_{C,A}$, Private Key sk_A Sever B: $\mathsf{pw}_{C,B}$, Private Key sk_B	Client: pw_C Server A: $\mathsf{pw}_{C,A}, \mathsf{pw}^*_{C,A}, d_A$ Server B: $\mathsf{pw}_{C,B}, \mathsf{pw}^*_{C,B}, d_B$
Computation Complexity	Client: 21(Exp.)+1(Sign) Server: about 6(KOY)	Client: 23(Exp.) Server: about 1(KOY)+ 2(Pairing)+9(Exp.)
Communication Complexity	Client/Server: 27(Exp.)+1(Sign) Server/Server: about 2(KOY)	Client/Server: 24(Exp.) Server/Server: about 1(KOY)

the size of the modulus and the size of the signature. KOY stands for the computation or communication complexity of the KOY protocol. From Table 1, we can see that the client has almost the same computation and communication complexities in both protocols, but the server in our protocol has about 1/3 of the computation complexity, 1/2 of the communication complexity of the Katz et al. two-server PAKE protocol if the computation of a pairing approximates to the computation of 4 exponentiations. Furthermore, our protocol achieves implicit authentication with only two communications between the client and the servers, whereas the Katz et al. two-server PAKE protocol needs three communications between the client and the servers to achieve implicit authentication.

The purpose of the users personal device is to keep public parameters. To efficiently compute 23 modular exponentiations, the client may load the public parameters from the personal device into a computing device.

4 Proof of Security

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

Theorem 1. Assuming that (1) the identity-based encryption (IBE) scheme is secure against the chosen-ciphertext attack; (2) the Cramer-Shoup public key encryption scheme is secure against the chosen-ciphertext attack; (3) the decisional Diffie-Hellman problem is hard; (4) the protocol P is a secure two-party PAKE protocol with explicit authentication; (5) H_1, H_2 are collision-resistant hash functions, then the protocol P' illustrated in Figure 1 is a secure two-server PAKE protocol according to Definition 1.

Proof. 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 $\mathsf{params} = \mathsf{params}^{P,E} \bigcup \{\mathbb{E}, \mathcal{G}, n, \mathbb{G}, q, g_1, g_2, H_1, H_2\}$ and the secret $\mathsf{master\text{-}key}^E$. Next, Client, Server, and Client ServerTriple sets are determined. Passwords for clients are chosen at random and split, and then stored at corresponding servers. Private keys for servers are computed using $\mathsf{master\text{-}key}^E$.

The public information is provided to the adversary. Considering $(C, A, B) \in \mathsf{ClinetServerTriple}$, we assume that the adversary $\mathcal{A}$ chooses the server B to corrupt and the simulator $\mathcal{S}$ gives the adversary $\mathcal{A}$ the information held by the corrupted server B, including the private key of the server B, i.e., d_B, and one share of the password of the client C, $\mathcal{G}^{\mathsf{pw}_{B,C}}$ and $g_2^{\mathsf{pw}^*_{B,C}}$. After computing the appropriate answer to any oracle query, the simulator $\mathcal{S}$ provides the adversary $\mathcal{A}$ with the internal state of the corrupted server B involved in the query.

We view the adversary's queries to its Send oracles as queries to four different oracles as follows:

- $\mathsf{Send}(C, i, A, B)$ represents a request for instance C^i of client C to initiate the protocol. The output of this query is $\mathsf{msg}_1 = \langle C, W_c, X, Y, Z, E_a \rangle$ and $\mathsf{msg}_2 = \langle C, W_c, X, Y, Z, E_b \rangle$.

- $\mathsf{Send}(A, j, \mathsf{msg}_1)$ represents sending message msg_1 to instance A^j of the server A. The output of this query is either $\mathsf{msg}_A = \langle A, S_1, T_1, U_1, V_1, W_a \rangle$ or $\perp$.
- $\mathsf{Send}(C, i, \mathsf{msg}_A | \mathsf{msg}_B)$ represents sending the message $\mathsf{msg}_A | \mathsf{msg}_B$ to instance C^i of the client C. The output of this query is either $\mathsf{acc}_C^i = \mathsf{TRUE}$ or $\perp$.
- $\mathsf{Send}^P(A, j, B, M)$ represents sending message M to instance A^j of the server A, supposedly by the server B, in the two-party PAKE protocol P. The input and output of this query depends on the protocol P.

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 session key $\mathsf{sk}_{U,U'}^i$, 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.

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_1, msg_2, msg_A, or msg_B) is repeated.
2. At any point during the experiment, a collision occurs in the hash function H_1 or H_2 (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_1, H_2 as collision-resistant hash functions. Put everything together, we are able to see that

Claim 1. If H_1 and H_2 are collision-resistant hash functions, $|\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_1}(k)|$ 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 $\mathsf{Execute}(C, i, A, j, B, k)$, where the adversary $\mathcal{A}$ has

not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $\mathcal{G}^{\mathsf{pw}_1 h^{-1}}$ in the ID-based encryption E_a is replaced with a random element in $\mathbb{E}$.

The difference between the current experiment and the previous one is bounded by the probability that an adversary breaks the semantic security of the identity-based encryption (IBE) scheme. More precisely, we have

Claim 2. If the identity-based encryption (IBE) scheme is semantically secure, $|\mathsf{Adv}^{P_1}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_2}_{\mathcal{A}}(k)|$ 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 $\mathsf{Execute}(C, i, A, j, B, k)$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $g_2^{\mathsf{pw}^*_{C,A} h_a^{-1}}$ in the Cramer-Shoup encryption (S_1, T_1, U_1, V_1) is replaced by a random element in the group $\mathbb{G}$.

The difference between the current experiment and the previous one is bounded by the probability that an adversary breaks the semantic security of the Cramer-Shoup encryption scheme. More precisely, we have

Claim 3. If the Cramer-Shoup encryption scheme is semantically secure, $|\mathsf{Adv}^{P_2}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_3}_{\mathcal{A}}(k)|$ 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 $\mathsf{Execute}(C, i, A, j, B, k)$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the session keys $\mathsf{sk}_{C,A}$ and $\mathsf{sk}_{A,C}$ are replaced with a same 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 $(\mathbb{G}, g, q)$. More precisely, we have

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

If $|\mathsf{Adv}^{P_3}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_4}_{\mathcal{A}}(k)|$ is non-negligible, we show that the simulator can use $\mathcal{A}$ as a subroutine to solve the DDH problem with non-negligible probability in a similar way as follows.

Given a DDH problem (g^α, g^β, Z), where α, β are randomly chosen from $\mathbb{Z}_q^*$ and Z is either $g^{\alpha\beta}$ or a random element z from $\mathbb{G}$, the simulator replaces W_c with g^α, and W_a with g^β, and the session keys $\mathsf{sk}_{C,A}$, $\mathsf{sk}_{A,C}$ with Z. When $Z = g^{\alpha\beta}$, 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. Assuming that the DDH problem is hard, Claim 4 is true.

In experiment P_4, 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 or a fresh server instance A^j invoked by an $\mathsf{Execute}(C, i, A, j, B, k)$ oracle, even if the adversary queried $\mathsf{corrupt}(B)$ (i.e., the adversary corrupted the

server B). This is so because the session keys $\mathsf{sk}_{C,A}$ and $\mathsf{sk}_{A,C}$ for such instances in P_4 are chosen at random from $\mathbb{G}$, and hence there is no way to distinguish whether the Test oracle outputs a random session key or the "actual" session key (which is just a random element, anyway). Therefore, all passive adversaries cannot win the game, even if they can query $\mathsf{Corrupt}(B)$ oracles.

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

Experiment P_5: In this experiment, we modify the simulator's responses to $\mathsf{Send}(A, j, \mathsf{msg}_1)$ and $\mathsf{Send}(C, i, \mathsf{msg}_A|\mathsf{msg}_B)$ queries.

Before describing this change we introduce some terminology. For a query $\mathsf{Send}(A, j, \mathsf{msg}_1)$, where msg_1 is adversarially - generated, if $\mathsf{acc}_A^j = \mathsf{TRUE}$, then msg_1 is said to be valid. Otherwise, msg_1 is said to be invalid. Similarly, for a query $\mathsf{Send}(C, i, \mathsf{msg}_A|\mathsf{msg}_B)$, where $\mathsf{msg}_A|\mathsf{msg}_B$ is adversarially-generated, if $\mathsf{acc}_C^i = \mathsf{TRUE}$, then $\mathsf{msg}_A|\mathsf{msg}_B$ is said to be valid. Otherwise, it is said to be invalid. Informally, valid messages use correct passwords while invalid messages do not. Given this terminology, we continue with our description of experiment P_5. When the adversary makes oracle query $\mathsf{Send}(A, j, \mathsf{msg}_1)$, the simulator examines msg_1. If it is adversarially-generated and valid, the simulator halts and acc_A^j is assigned the special value ∇. In any other case, (i.e., msg_1 is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment P_4. When the adversary makes oracle query $\mathsf{Send}(C, i, \mathsf{msg}_A|\mathsf{msg}_B)$, the simulator examines $\mathsf{msg}_A|\mathsf{msg}_B$. If the message is adversarially-generated and valid, the simulator halts and acc_C^i is assigned the special value ∇. In any other case, (i.e., $\mathsf{msg}_A|\mathsf{msg}_B$ is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment P_4.

Now, we change the definition of the adversary's success in P_5. At first, we define that a server instance A^j is fresh if the adversary has not queried $\mathsf{Corrupt}(A)$ and a client instance C^i is fresh if the adversary has not queried $\mathsf{Corrupt}(C)$. If the adversary ever queries $\mathsf{Send}(A, j, \mathsf{msg}_1)$ oracle to a fresh server instance A^j with $\mathsf{acc}_A^j = \nabla$ or $\mathsf{Send}(C, i, \mathsf{msg}_A|\mathsf{msg}_B)$ oracle to a fresh client instance C^i with $\mathsf{acc}_C^i = \nabla$, the simulator halts and the adversary succeeds. Otherwise the adversary's success is determined as in experiment P_4.

The distribution on the adversary's view in experiments P_4 and P_5 are identical up to the point when the adversary queries $\mathsf{Send}(A, j, \mathsf{msg}_1)$ oracle to a fresh server instance with $\mathsf{acc}_A^j = \nabla$ or $\mathsf{Send}(C, i, \mathsf{msg}_A|\mathsf{msg}_B)$ oracle to a fresh client instance with $\mathsf{acc}_C^i = \nabla$. If such a query is never made, the distributions on the view are identical. Therefore, we have

Claim 5. $\mathsf{Adv}_{\mathcal{A}}^{P_4}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_5}(k)$.

Experiment P_6: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_5 except that the adversary's queries to $\mathsf{Send}(C, i, A, B)$ and $\mathsf{Send}(A, j, \mathsf{msg}_1)$ oracles are handled differently: in any $\mathsf{Send}(C, i, A, B)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $\mathcal{G}^{\mathsf{pw}_1 h^{-1}}$ in E_a is replaced with a random element in the group $\mathbb{E}$; in any $\mathsf{Send}(A, j, \mathsf{msg}_1)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $g_2^{\mathsf{pw}^*_{C,A} h_a^{-1}}$ in the Cramer - Shoup

encryption (S_1, T_1, U_1, V_1) (if any) is replaced with a random element in the group $\mathbb{G}$.

As we prove Claims 2 and 3, we can prove

Claim 6. If both the IBE scheme and the Cramer-Shoup scheme are semantically secure, $|\mathsf{Adv}_{\mathcal{A}}^{P_5}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_6}(k)|$ is negligible.

In experiment P_6, msg_1 and msg_A from $\mathsf{Execute}$ and Send oracles become independent of the password pw_C used by the client C in the view of the adversary $\mathcal{A}$, even if $\mathcal{A}$ may require $\mathsf{Corrupt}(B)$. In addition, although the adversary who has corrupted the server B is able to obtain $\mathcal{G}^{\mathsf{pw}_2}$, $\mathcal{G}^{\mathsf{pw}_{C,B}}$ and $g_2^{\mathsf{pw}^*_{C,B}}$, they are independent of the password pw_C in the view of the adversary because the references msg_1 and msg_A are independent of the password in the view of the adversary. In view of this, any off-line dictionary attack cannot succeed.

The adversary $\mathcal{A}$ succeeds only if one of the following occurs: (1) the adversary queries $\mathsf{Send}(A, j, \mathsf{msg}_1)$ oracle to a fresh server instance A^j for adversarially-generated and valid msg_1, that is, $\mathsf{acc}_A^j = \nabla$ (let Succ_1 denote this event); (2) the adversary queries $\mathsf{Send}(C, i, \mathsf{msg}_A|\mathsf{msg}_B)$ oracle to a fresh client instance C^i for adversarially-generated and valid $\mathsf{msg}_A|\mathsf{msg}_B$, that is, $\mathsf{acc}_C^i = \nabla$ (let Succ_2 denote this event); (3) neither Succ_1 nor Succ_2 happens, the adversary wins the game by a Test query to a fresh instance C^i or a server instance A^j.

To evaluate $\mathsf{Pr}[\mathsf{Succ}_1]$ and $\mathsf{Pr}[\mathsf{Succ}_2]$, we assume that the adversary $\mathcal{A}$ has corrupted the server B and consider four cases as follows.

Case 1. The adversary $\mathcal{A}$ modifies $\mathsf{msg}_1 = \langle C, W_c, X, Y, Z, E_a\rangle$ from the client by changing W_c, X, Y, Z and then the plaintext $\mathcal{G}^{\mathsf{pw}_1} h^{-1}$ in E_a. Changing the plaintext in a ciphertext works with a public key cryptosystem with homomorphic property, such as the ElGamal scheme [14]. It does not work with the IBE scheme which is secure against the chosen-ciphertext attack. The best the adversary can do is choosing W_c', X', Y', Z' such that $H_1(C, W_c', X', Y', Z') = H_1(C, W_c, X, Y, Z)$. But H_1 is a collision-resistant hash function. Therefore, the probability of Succ_1 in this case is negligible.

Case 2. The adversary $\mathcal{A}$ forges $\mathsf{msg}_1' = \langle C, W_c', X', Y', Z'\ E_a'\rangle$ by choosing his own W_c', X', Y', Z', E_a'. In this case, the best the adversary can do is to choose $r_c', x_1', x_2', y_1', y_2'$ and computes $W_c' = g_1^{r_c'}, X' = g_1^{x_1'} g_2^{x_2'}, Y' = g_1^{y_1'} g_2^{y_2'}, Z' = g_1^{z'}, h^* = H_1(C, W_c', X', Y', Z')$ and performs identity-based encryption $E_a' = \mathsf{IBE}(\mathcal{G}^{-\mathsf{pw}_{C,B}} h^{*-1}, A)$, and sends msg_1' to the server A. Note that the adversary $\mathcal{A}$ has corrupted B and know $\mathcal{G}^{\mathsf{pw}_{C,B}}$. After receiving msg_1', the server A derives $\omega_a = \mathcal{G}^{-\mathsf{pw}_{C,B} - \mathsf{pw}_{C,A}} = \mathcal{G}^{-\mathsf{pw}_C}$ and then runs two-party PAKE protocol P with the server B (controlled by the adversary) on the basis of ω_a. Without knowing ω_a, the probability of Succ_1 depends on the protocol P. If P is a secure two-party PAKE protocol with explicit authentication, $\mathsf{Pr}[\mathsf{Succ}_1] \leq Q^P(k)/N + \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$, where $Q^P(k)$ denotes the number of on-line attacks in the protocol P.

Case 3. The adversary $\mathcal{A}$ modifies $\mathsf{msg}_A = \langle A, S_1, T_1, U_1,\ V_1, W_a\rangle$ from the server A. In msg_A, (S_1, T_1, U_1, V_1) is a Cramer-Shoup encryption of $g_2^{\mathsf{pw}^*_{C,A}} h_a^{-1}$,

where $h_a = H_2(A,\ W_a, C, W_c)$. If the adversary $\mathcal{A}$ changes the plaintext in (S_1, T_1, U_1, V_1), the message becomes invalid because the Cramer-Shoup encryption scheme is secure against the chosen ciphertext attack. The best the the adversary can do is choosing his own W'_a such that $H_2(A, W'_a, C, W_c) = H_2(A, W_a, C, W_c)$. But H_2 is a collision-resistant hash function. Therefore, the probability of Succ_2 in this case is negligible.

Case 4. The adversary $\mathcal{A}$ forges $\mathsf{msg}'_A = \langle A, S'_1, T'_1, U'_1,\ V'_1, W'_a\rangle$ by choosing a password pw'_C from the dictionary $\mathcal{D}$ and $r'_a, r'_1 \in \mathbb{Z}_q^*$ and computing $W'_a = g_1^{r'_a}, S'_1 = g_1^{r'_1}, T'_1 = g_2^{r'_1}, h^*_a = H_2(A, W'_a, C, W_c), U_1 = g_2^{(\mathsf{pw}'_C - \mathsf{pw}^*_{C,B})h^{*-1}_a} Z^{r'_1}, h^*_1 = H_2(S'_1, T'_1, U'_1), V_1 = X'^{r'_1} Y'^{h^*_1 r'_1}$. Then $\mathcal{A}$ sends $\mathsf{msg}'_A|\mathsf{msg}_B$ to the client, suppose that msg_B is constructed as defined by the protocol. In this case, the event Succ_2 occurs if and only if $\mathsf{pw}'_C = \mathsf{pw}_C$. Therefore, $\mathsf{Pr}[\mathsf{Succ}_2] \leq Q^C(k)/N$, where $Q^C(k)$ denotes the number of on-line attacks to the client instance C^i.

The above discussion shows that

Claim 7. If (1) P is a secure two-party PAKE protocol with explicit authentication; (2) the IBE scheme and the Cramer-Shoup scheme are secure against the chosen-ciphertext attack; (4) H_1 and H_2 are collision-resistant hash functions, then $\mathsf{Pr}[\mathsf{Succ}_1 \vee \mathsf{Succ}_2] \leq Q(k)/N + \varepsilon(k)$, where $Q(k)$ denotes the number of on-line attacks and $\varepsilon(k)$ is a negligible function.

Remark. If a public key encryption scheme is secure against the chosen-ciphertext attack (CCA), it is secure against the chosen-plaintext attack (CPA) (i.e., it is semantically secure).

In experiment P_6, the adversary's probability of success when neither Succ_1 nor Succ_2 occurs is 1/2. The preceding discussion implies that

$$Pr^{P_6}_{\mathcal{A}}[\mathsf{Succ}] \leq Q(k)/N + \varepsilon(k) + 1/2 \cdot (1 - Q(k)/N - \varepsilon(k))$$

and thus the adversary's advantage in experiment P_6

$$\begin{aligned}\mathsf{Adv}^{P_6}_{\mathcal{A}}(k) &= 2Pr^{P_6}_{\mathcal{A}}[\mathsf{Succ}] - 1 \\ &\leq 2Q(k)/N + 2\varepsilon(k) + 1 - Q(k)/N - \varepsilon(k) - 1 \\ &= Q(k)/N + \varepsilon(k)\end{aligned}$$

for some negligible function $\varepsilon(\cdot)$. The sequence of claims proved above show that

$$\mathsf{Adv}^{P_0}_{\mathcal{A}}(k) \leq \mathsf{Adv}^{P_6}_{\mathcal{A}}(k) + \varepsilon(k) \leq Q(k)/N + \varepsilon(k)$$

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

5 Conclusion

In this paper, we present an efficient compiler to transform any two-party PAKE protocol to a two-server PAKE protocol from identity-based encryption. In addition, we have provided a rigorous proof of security for our compiler without random oracle. Our compiler is in particular suitable for the applications of password-based authentication where an identity-based system has already established.

References

1. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005)
2. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005)
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocol secure against dictionary attack. In: Proc. 1992 IEEE Symposium on Research in Security and Privacy, pp. 72–84 (1992)
5. Boneh, D., Franklin, M.: Identity based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Boneh, D., Katz, J.: Improved efficiency for CCA-secure cryptosystems built using identity based encryption. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 87–103. Springer, Heidelberg (2005)
7. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
8. Brainard, J., Juels, A., Kaliski, B., Szydlo, M.: Nightingale: A new two-server approach for authentication with short secrets. In: Proc. 12th USENIX Security Symp., pp. 201–213 (2003)
9. Bresson, E., Chevassut, O., Pointcheval, D.: Security proofs for an efficient password-based key exchange. In: Proc. CCS 2003, pp. 241–250 (2003)
10. Bresson, E., Chevassut, O., Pointcheval, D.: New security results on encrypted key exchange. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 145–158. Springer, Heidelberg (2004)
11. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
12. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
13. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory 32(2), 644–654 (1976)
14. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
15. Ford, W., Kaliski, B.S.: Server-assisted generation of a strong secret from a password. In: Proc. 5th IEEE Intl. Workshop on Enterprise Security (2000)
16. Gentry, C.: Practical identity-based encryption without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006)
17. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 408–432. Springer, Heidelberg (2001)

18. Gong, L., Lomas, T.M.A., Needham, R.M., Saltzer, J.H.: Protecting poorly-chosen secret from guessing attacks. IEEE J. on Selected Areas in Communications 11(5), 648–656 (1993)
19. Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. ACM Transactions on Information and System Security 2(3), 230–268 (1999)
20. Jablon, D.: Password authentication using multiple servers. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 344–360. Springer, Heidelberg (2001)
21. Jiang, S., Gong, G.: Password based key exchange with mutual authentication. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 267–279. Springer, Heidelberg (2004)
22. Jin, H., Wong, D.S., Xu, Y.: An efficient password-only two-server authenticated key exchange system. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 44–56. Springer, Heidelberg (2007)
23. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer, Heidelberg (2001)
24. Katz, J., MacKenzie, P., Taban, G., Gligor, V.: Two-server password-only authenticated key exchange. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 1–16. Springer, Heidelberg (2005)
25. MacKenzie, P., Patel, S., Swaminathan, R.: Password-authenticated key exchange based on RSA. Intl. J. Information Security 9(6), 387–410 (2010)
26. MacKenzie, P., Shrimpton, T., Jakobsson, M.: Threshold password-authenticated key exchange. J. Cryptology 19(1), 27–66 (2006)
27. Di Raimondo, M., Gennaro, R.: Provably Secure Threshold Password-Authenticated Key Exchange. J. Computer and System Sciences 72(6), 978–1001 (2006)
28. Szydlo, M., Kaliski, B.: Proofs for two-server password authentication. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 227–244. Springer, Heidelberg (2005)
29. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
30. Yang, Y., Bao, F., Deng, R.H.: A new architecture for authentication and key exchange using password for federated enterprise. In: Sasaki, R., Qing, S., Okamoto, E., Yoshiura, H. (eds.) SEC 2005. IFIP AICT, vol. 181, pp. 95–111. Springer, Heidelberg (2005)
31. Yang, Y., Deng, R.H., Bao, F.: A practical password-based two-server authentication and key exchange system. IEEE Trans. Dependable and Secure Computing 3(2), 105–114 (2006)
32. Yang, Y., Deng, R.H., Bao, F.: Fortifying password authentication in integrated healthcare delivery systems. In: Proc. ASIACCS 2006, pp. 255–265 (2006)
33. Yi, X., Tso, R., Okamoto, E.: ID-based group password-authenticated key exchange. In: Takagi, T., Mambo, M. (eds.) IWSEC 2009. LNCS, vol. 5824, pp. 192–211. Springer, Heidelberg (2009)
34. Yi, X., Tso, R., Okamoto, E.: Identity-based password-authenticated key exchange for client/server model. In: SECRYPT 2012, pp. 45–54 (2012)
35. Yi, X., Ling, S., Wang, H.: Efficient two-server password-only authenticated key exchange. IEEE Trans. Parallel Distrib. Syst. 24(9), 1773–1782 (2013)