Identity-based Password-Authenticated Key Exchange for Client/Server Model

Xun Yi[1], Raylin Tso[2] and Eiji Okamoto[3]
[1]*School of Engineering and Science, Victoria University, Melbourne, Victoria 8001, Australia*
[2]*Department of Computer Science, National Chengchi University, Taipei 11605, Taiwan*
[3]*Department of Risk Engineering, University of Tsukuba, Tsukuba, Ibaraki, 305-8573, Japan*

Keywords: PAKE, Client/Server Model, Identity-based Encryption, Decisional Diffie-Hellman Problem.

Abstract: Password-Authenticated Key Exchange for Client/Server model (PAKE-CS) is where a client and a server, based only on their knowledge of a password, establish a cryptographic key for secure communication. In this paper, we propose a PAKE-CS protocol on the basis of identity-based encryption, where the client needs to remember a password only while the server keeps the password in addition to a private key related to his identity, where the private key is generated by multiple private key generators. Our protocol takes advantage of the features of client/server model and is more efficient than other PAKE-CS protocols in terms that it achieves explicit authentication with two-round communications only. In order to analyze the security of our protocol, we construct an ID-based formal model of security for PAKE-CS by embedding ID-based model into PAKE model. If the underlying identity-based encryption scheme has provable security without random oracle, we can provide a rigorous proof of security for our protocol without random oracles.

1 INTRODUCTION

Nowadays, the client/server model has become one of the core ideas of network computing. The Internet's main application protocols, such as HTTP, SMTP, Telnet, DNS, etc, are all built on the client/server model. Most business applications use this model as well.

In the client/server model, a client and a server communicate to exchange information. It is essential for the server to identify the client before providing the client with services, such as access to resources. In order to authenticate the client, it is common for the client to choose a password from a known small space, such as a dictionary of English words, in order to remember it, and shares it with the server in advance. After that, each time when the client sends the password to the server, the server can identify the client. Such a password authentication protocol can go back to 1981 (Lamport, 1981) and even earlier.

After password authentication, the client and the server need to exchange information securely. Before encrypting messages, the client and the server have to share a cryptographic key. The password, chosen from a small space, cannot be used as the key for encryption. A solution for this problem is password-authenticated key exchange for client/server (PAKE-CS).

PAKE-CS is where a client and a server, based only on their knowledge of a password, establish a cryptographic key, such that an attacker who controls the communication channel but does not possess the password cannot impersonate the client or the server in the communication and is constrained as much as possible from guessing the password. Since the password is chosen from a small space, PAKE-CS has to be immune to the dictionary attack, in which an adversary exhaustively tries all possible passwords from a dictionary in order to determine the correct one. Dictionary attacks are either off-line attacks or online attacks. In an off-line attack, an adversary eavesdrops messages exchanged between the client and server, and tries all possible passwords to determine one matching with these messages. In an online attack, an adversary impersonates a client or a server by trying possible passwords one-by-one. Online dictionary attack can be discouraged by restricting the number of password authentication failures.

Initial solutions for PAKE-CS is built on a "hybrid" model in which the client stores the server's public key in addition to share a password with the server. Under this model, Gong, Lomas, Needham,

Yi X., Tso R. and Okamoto E..
Identity-based Password-Authenticated Key Exchange for Client/Server Model.
DOI: 10.5220/0004015900450054
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2012), pages 45-54
ISBN: 978-989-8565-24-2

and Saltzer (Gong et al., 1993) were the first to propose password-based authentication protocols with heuristic resistance to off-line dictionary attacks, and Halevi and Krawczyk (Halevi and Krawczyk, 1999) were the first to give formal definitions and rigorous proofs of security for them. The "hybrid" model relies on the Public Key Infrastructure (PKI), where the public key of the server is certified within a certificate issued by a trusted third party. Exchanging and verifying the public key certificate bring extra computation and communication costs to PAKE-CS.

Bellovin and Merritt (Bellovin and Merritt, 1992) were the first to consider authenticated key exchange based on password only. They introduced a set of so-called "encrypted key exchange" (EKE) protocols, where any two parties, who share a password, exchange messages encrypted by the password, and establish a cryptographic key from them. Although several of the first protocols were flawed, the survived and enhanced EKE protocols effectively amplify a shared password into a shared cryptographic key. Based on EKE, some further works (Gong et al., 1993; Huang, 1996; Wu, 1998) have been done. However, only heuristic and informal security arguments for these protocols were provided. In fact, attacks against many of these protocols have been found (MacKenzie et al., 2000; Patel, 1997). This demonstrates the great importance of rigorous security proofs in a formal, well-defined model.

In 2000, formal models of security for PAKE were firstly given independently by Bellare, Pointcheval and Rogaway (Bellare et al., 2000), and Boyko, MacKenzie, Patel and Swaminathan (Boyko et al., 2000). In the ideal cipher model, Bellare et al. (Bellare et al., 2000) provided a proof of security for the two-flow protocol at the core of Bellovin-Merritt EKE protocol (Bellovin and Merritt, 1992). In the random oracle model, Boyko et al. (Boyko et al., 2000) proved the security of their new Diffie-Hellman-based PAKE while MacKenzie et al. (MacKenzie et al., 2000) provided the security proof of their new RSA-based PAKE. Later, some efficient PAKE protocols (e.g.,(Abdalla and Pointcheval, 2005; Bresson et al., 2003)) were constructed. In 2001, Goldreich and Lindell (Goldreich and Lindell, 2001) introduced another model of security for PAKE and gave the first PAKE protocol which is provably secure under standard cryptographic assumptions. Their protocol does not require any additional setup beyond the password shared by the parties. However, their protocol requires techniques from generic two-party secure computation and concurrent zero-knowledge. This makes their protocol computationally inefficient. A simple version of Goldreich-Lindell protocol was given by Nguyen and Vadhan in (Nguyen and Vadhan, 2004), but it is still not efficient enough to be used in practice.

Katz, Ostrovsky, and Yung (Katz et al., 2001) were the first to give a PAKE protocol which is both practical and provably-secure under standard cryptographic assumption. Katz-Ostrovsky-Yung protocol (simply called KOY protocol) has been proved to be secure in the model of Bellare et al. (Bellare et al., 2000) under the decisional Diffie-Hellman assumption. In KOY protocol, the client and the server exchange the encryptions of the password (on the basis of a common public key), from which a common cryptographic key is agreed and authenticated by one-time digital signature scheme. KOY protocol assumes that a set of common parameters (including the common public key) are available to everyone in the system. This is known as the common reference model, which avoids problems associated with the PKI. This assumption is significantly weaker (in both a theoretical and practical sense) than the "hybrid" model in which clients are required to authenticate the public key for each server with whom they wish to communicate. The public parameters can be "hard-coded" into the implementation of their protocol. Therefore, the requirement of public parameters does not represents a serious barrier to using their protocol in practice.

Afterward, an efficient protocol for PAKE with proof of security based on a pseudorandom function family was given by Jiang and Gong in (Jiang and Gong, 2004), and a protocol satisfying a strong definition of security for PAKE built on (Katz et al., 2001; Gennaro and Lindell, 2003) was proposed in (Canetti et al., 2005).

Our Contribution. The "hybrid" model for PAKE-CS can be used efficiently to establish a cryptographic key between the client and the server who share a password. However, this model needs the PKI and the client has to authenticate the public key of the server before the execution of PAKE-CS. The common reference model for PAKE-CS avoids the PKI, but protocols built on this model, in particular with proofs of security under standard cryptographic assumptions, are usually less efficient than those based on the "hybrid" model.

In the client/server model, the client is usually a human user who can remember the password from a small space only. However, the server is a machine which can keep secret keys from a large space. Based on this feature, identity-based group and three-party PAKE protocols, in which a group of clients, each of them shares his password with an "honest but curious" server, establish a group key with the help of

the server, have been proposed in (Yi et al., 2009; Yi et al., 2011). In that setting, the key established is known to the clients only and no one else, including the server.

In this paper, we consider a two-party setting, where a client and a server, who share a password, establish a cryptography key on the basis of identity-based encryption. The essential difference between this setting and the group or three-party PAKE (Yi et al., 2009; Yi et al., 2011) is that the established key in this setting is known to the server. We propose an identity-based model for this setting, where the client needs to remember a password only while the server keeps the password in addition to a private key related to its identity. This model is constructed by embedding the model for identity-based encryption (IBE) (Boneh and Franklin, 2001) into the model for PAKE (Bellare et al., 2000). In this model, we assume that n private key generators cooperate to generate public parameters and private keys for servers.

Based on the identity-based model, we construct an efficient PAKE-CS protocol. Our protocol can be built on any IBE scheme, such as Waters's scheme (Waters, 2005), which allows multiple private key generators to generate private keys for users.

The basic structure of our protocol is Diffie-Hellman key exchange between the client and the server. But we use the password to authenticate the request message from the client and confirm the established cryptographic key. In order to deter from offline dictionary attack, the client encrypts the password with an IBE scheme on the basis of the identity of the server.

Our protocol has an advantage over the KOY protocol (Katz et al., 2001). The KOY protocol does not achieve explicit authentication (that is, a party does not know whether its intended partner has successfully computed a matching session key). The explicit authentication has to be added later on using standard techniques as described in (Bellare et al., 2000). Thus, the KOY protocol needs four-round communications to achieve explicit authentication. But our protocol uses only two-round communications to achieve explicit authentication.

In terms of the number of communication rounds, an identity-based PAKE-CS protocol is more efficient than other PAKE-CS protocols. Different from a pure identity-based key agreement protocol, such as RFC 6539 (Cakulev et al., 2012), which requires each party to have a (random) private key related to his identity, the client in our protocol needs to remember passwords only (no cryptographic key of any kinds), and the server keeps passwords in addition to a private key related to his identity.

2 MODEL AND DEFINITIONS

A formal model of security for PAKE was given by Bellare, Pointcheval and Rogaway in (Bellare et al., 2000), and improved by Katz, Ostrovsky, and Yung in (Katz et al., 2001). Boneh and Franklin defined chosen ciphertext security for IBE systems under a chosen identity attack in (Boneh and Franklin, 2001). In this section, we give the ID-based model for PAKE-CS, a combination of definitions given in (Bellare et al., 2000; Katz et al., 2001; Boneh and Franklin, 2001).

Participants, Initialization and Passwords. An ID-based PAKE-CS 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) n private key generators $PKG_1, PKG_2, \cdots, PKG_n$ (denoted as PKG), which cooperate to generate public parameters and private keys for servers. Private key generators are not servers.

We assume that a honest private key generator follows the exact protocol, but a dishonest private key generator may perform attacks on the protocol to retrieve the cryptographic key established between the client and the server. In addition, let $\mathsf{ClientServerPair}$ be the set of pairs of the client and the server, who share a password, and let $\mathsf{User} = \mathsf{Client} \bigcup \mathsf{Server}$. We assume that $\mathsf{Client} \bigcap \mathsf{Server} = \emptyset$.

Prior to any execution of the protocol, we assume that an initialization phase occurs. During initialization, PKG cooperate to generate public parameters for the protocol, which are available to all participants, and private key for each server, which is given to the appropriate server.

For any pair $(C,S) \in \mathsf{ClientServerPair}$, the client C and the server S are assumed to share the same password pw_C^S, which is what C remembers to log into S. We assume that the client C chooses pw_C^S independently and uniformly at random from a "dictionary" $\mathcal{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ of size N, where N is a fixed constant which is independent of the security parameter. It is then stored at the server S for authentication.

In this model, not every pair of client and server share passwords. A client may share different passwords with different servers.

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 has associated with it the following variables, initialized as NULL or FALSE (as appropriate) during the initialization phase.

- $\mathsf{sid}_U^i, \mathsf{pid}_U^i$ and sk_U^i (initialized as NULL) are variables containing the session identity, partner identity, and session key for an instance U^i, respectively. The session identity is simply a way to keep track of the different executions of a particular user U. The partner identity denotes the identity of the user with whom U^i believes it is interacting (including U^i itself).
- acc_U^i and term_U^i (initialized as FALSE) are boolean variables denoting whether a given instance U^i has been accepted, terminated, or authenticated, 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 accepts, sid_U^i, pid_U^i and sk_U^i are no longer NULL.
- state_U^i (initialized as NULL) records any state necessary for execution of the protocol by U^i.
- used_U^i (initialized as FALSE) 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 and the adversary's interaction with the users (more specifically, with various instances) or the PKG is modeled via access to oracles which we describe now. 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 calls include:

- $\mathsf{Send}(U^i, M)$ – This sends message M to instance U^i. Assuming $\mathsf{term}_U^i = \mathsf{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of U^i (i.e., the message sent by the instance) is given to the adversary, who receives the updated values of $\mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{acc}_U^i$, and term_U^i. This oracle call models the active attack to a protocol.
- $\mathsf{Execute}(C^i, S^j)$ – If C^i and S^j have not yet been used (where $(C,S) \in \mathsf{ClientServerPair}$), this oracle execute 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 both instances, at each step of protocol execution.
- $\mathsf{Corrupt}(C)$ – This query allows the adversary to learn the passwords of the client C, which models the possibility of subverting a client by, for example, witnessing a user type in his password, or installing a "Trojan horse" on his machine. This implies that all passwords held by C are disclosed.
- $\mathsf{Corrupt}(S)$ – This query allows the adversary to learn the private key of the server S, which models the possibility of compromising a server by, for example, hacking into the server. This implies that all passwords held by S are disclosed as well.
- $\mathsf{KeyGen}(PKG_i, S)$ – This sends the identity of the server S to the PKG_i, which generates one component of the private key corresponding to the identity of S and forwards it to the adversary. This oracle models the possibility of a private key generator being an adversary.
- $\mathsf{Reveal}(U^i)$ – This outputs the current value of session key sk_U^i if $\mathsf{acc}_U^i = \mathsf{TRUE}$. This oracle call models possible leakage of session keys due to, for example, improper erasure of session keys after use, compromise of a host computer, or cryptanalysis.
- $\mathsf{Test}(U^i)$ – This oracle does not model any real-world capability of the adversary, but is instead used to define security. If $\mathsf{acc}_U^i = \mathsf{TRUE}$, a random bit b is generated. If $b = 0$, the adversary is given sk_U^i, 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. We say that a client instance C^i and a server instance S^j are partnered if (1) $\mathsf{pid}_C^i = \mathsf{pid}_S^j \neq \mathsf{NULL}$; and (2) $\mathsf{sid}_C^i = \mathsf{sid}_S^j \neq \mathsf{NULL}$; and (3) $\mathsf{sk}_C^i = \mathsf{sk}_S^j \neq \mathsf{NULL}$; and (4) $\mathsf{acc}_C^i = \mathsf{acc}_S^i = \mathsf{TRUE}$. The notion of partnering will be fundamental in defining both correctness and security.

Correctness. To be viable, an authenticated key exchange protocol must satisfy the following notion of correctness: At the presence of both passive and active adversaries, for any pair of client and server instances C^i and S^j, if $\mathsf{sid}_C^i = \mathsf{sid}_S^j \neq \mathsf{NULL}$ and $\mathsf{acc}_C^i =$

$\mathsf{acc}_S^j = \mathsf{TRUE}$, then it must be the case that $\mathsf{sk}_C^i = \mathsf{sk}_S^j \neq \mathsf{NULL}$ (i.e., both conclude with the same session key) and $\mathsf{pid}_C^i = \mathsf{pid}_S^j \neq \mathsf{NULL}$ (i.e., both conclude with the same pair). The notion of correctness has no restriction on the adversary's oracle accesses.

Advantage of the Adversary. Informally, the adversary succeeds if it can guess the bit b used by the Test oracle. Before formally defining the adversary's success, we must first define a notion of freshness. A user instance A^i (either a client or a server) is fresh if none of the following is true at the conclusion of the experiment, namely, at some point,

- The adversary queried $\mathsf{Reveal}(A^i)$ or $\mathsf{Reveal}(B^j)$ with the instances A^i and B^j being partnered;
- The adversary queried $\mathsf{KeyGen}(PKG_i, S)$ ($i = 1, 2, \cdots, n$) where there exists a server instance $S^j \in \mathsf{pid}_A^i$, before a query of the form $\mathsf{Send}(U^\ell, M)$, where $U^\ell \in \mathsf{pid}_A^i$, has taken place, for some message M (or identities);
- The adversary queried $\mathsf{Corrupt}(A)$ or $\mathsf{Corrupt}(B)$ where there exists a instance $B^j \in \mathsf{pid}_A^i$, before a query of the form $\mathsf{Send}(U^\ell, M)$, where $U^\ell \in \mathsf{pid}_A^i$, has taken place, for some message M (or identities).

Note that passive adversaries have no access to any Send oracles. Therefore, a user instance is fresh to a passive adversary as long as the first event did not happen.

The adversary is thought to succeed only if its Test query is made to a fresh instance. We say an adversary $\mathcal{A}$ succeeds if it makes a single query $\mathsf{Test}(U^i)$ to a fresh instance U^i, with $\mathsf{acc}_U^i = \mathsf{TRUE}$ 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

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

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) and k is a security parameter.

Formally, an instance U^i represents an on-line attack if both the following are true at the time of the Test query: (1) at some point, the adversary queried $\mathsf{Send}(U^i, *)$, and (2) at some point, the adversary queried $\mathsf{Reveal}(U^i)$ or $\mathsf{Test}(U^i)$. In particular, instances with which the adversary interacts via Execute, KeyGen, Corrupt and Reveal queries 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 protocol for password-authenticated key exchange 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

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

where k is a security parameter.

The above definition ensures that the adversary can (essentially) do no better than guess a single password during each on-line attack. Calls to the Execute, KeyGen, Corrupt and Reveal oracles, which are not included in $Q(k)$, are of no help to the adversary in breaking the security of the protocol.

Forward Secrecy. We follow the definition of forward secrecy from (Katz et al., 2003) and consider the weak corrupt model of (Bellare et al., 2000), in which corrupting a client means retrieving his passwords, while corrupting a server means retrieving its private key and all passwords stored in it. Forward secrecy is then achieved if such queries do not give the adversary any information about previous agreed session keys.

3 IDENTITY-BASED PAKE-CS PROTOCOL

The high-level depiction of the protocol is illustrated in Fig. 1, and a more detailed description follows. A completely formal specification of the protocol will appear in Section 5, where we give a proof of security for the protocol in the security model described in Section 2.

We present the protocol by describing initialization and execution. We let k be the security parameter given to the setup algorithm.

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

Parameter Generation: On input k, (1) PKG 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 $\mathsf{master\text{-}key}^E = \{\mathsf{mk}_1, \mathsf{mk}_2, \cdots, \mathsf{mk}_n\}$, where ^{E} stands for encryption and mk_i is known only to PKG_i; (2) PKG choose a large Mersenne prime $q = 2^p - 1$, where p is a prime and construct a finite field F_{2^p}, where each element α is corresponding to a binary vector $(\alpha_1, \alpha_2, \cdots, \alpha_p)$, and $F_{2^p}^*$ is a large cyclic multiplicative group (denoted as G) with a prime order q. Assume g is a generator of G; (3) PKG select two hash functions $h : \{0,1\}^* \to \mathbb{Z}_q$ and $H : \{0,1\}^* \to \mathcal{M}$ (where $\mathcal{M}$ is the plaintext space of IBE) from a collision-resistant hash family.

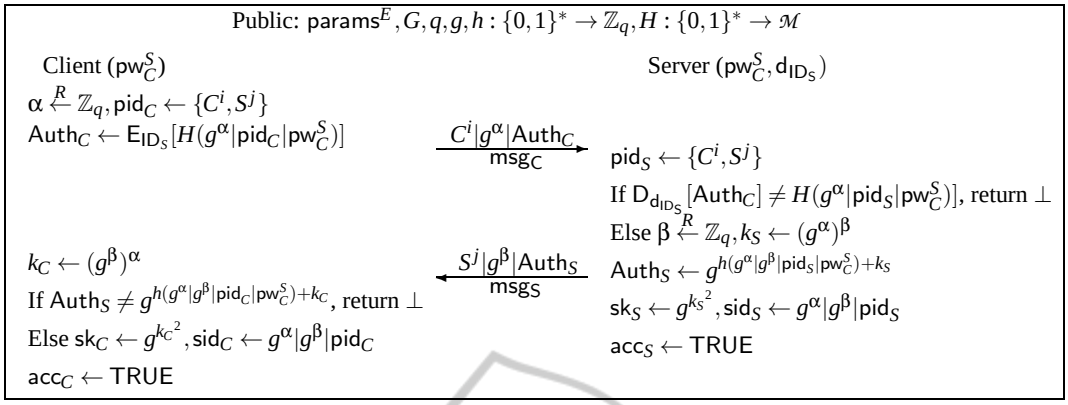

Figure 1: ID-based PAKE-CS protocol P.

The public system parameters for the protocol P are $\mathsf{params} = \mathsf{params}^E \bigcup \{h, H, G, q, g\}$ and the secret parameter is $\mathsf{master\text{-}key}^E$.

Key Generation: On input the identity ID_S of a server $S \in \mathsf{Server}$, params, and $\mathsf{master\text{-}key}^E$, PKE runs $\mathsf{Extract}^E$ of the IBE scheme and sets the decryption key of S to be $\mathsf{d}_{\mathsf{ID}_S} = \{d_{S,1}, d_{S,2}, \cdots, d_{S,n}\}$ where $d_{S,i}$ is generated by PKG_i with params and mk_i.

Password Generation: On input $(C,S) \in \mathsf{ClientServerPair}$, a string pw_C^S, the password, is uniformly drawn by the client C from the dictionary $\mathsf{Password} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$, and then store it in the server S.

Protocol Execution. A client and a server, where $(C,S) \in \mathsf{ClientServerPair}$ (i.e., they share a password pw_C^S), execute the protocol as follows. The client C firstly randomly chooses $\alpha \in \mathbb{Z}_q$, and computes Auth_C, which is an encryption of $H(g^\alpha|\mathsf{pid}_C|\mathsf{pw}_C^S)$ on the basis of the identity of the server ID_S, denoted as $\mathsf{E}_{\mathsf{ID}_S}[H(g^\alpha|\mathsf{pid}_C|\mathsf{pw}_C^S)]$, where $\mathsf{pid}_C = \{C^i, S^j\}$. Please note that the identity of the server, like an e-mail address, is meaningful and easy to remember and keep. Then the client C sends $\mathsf{msg}_C = C^i|g^\alpha|\mathsf{Auth}_C$ to the server S as the first message of the protocol.

Upon receiving the message msg_C, S firstly decrypts the ciphertext with its private key $\mathsf{d}_{\mathsf{ID}_S}$ and then verifies the password. The password is correct if

$$\mathsf{D}_{\mathsf{d}_{\mathsf{ID}_S}}[\mathsf{Auth}_C] = H(g^\alpha|\mathsf{pid}_S|\mathsf{pw}_C^S) \quad (1)$$

where $\mathsf{pid}_S = \{C^i, S^j\}$.

If (1) holds, the server S randomly chooses $\beta \in \mathbb{Z}_q$ and computes $k_S = g^{\alpha\beta}$. Let $\mathsf{Auth}_S = g^{h(g^\alpha|g^\beta|\mathsf{pid}_S|\mathsf{pw}_C^S)+k_S}$, the server S replies to the client C with $\mathsf{msg}_S = S^j|g^\beta|\mathsf{Auth}_S$, and finally computes the session key $\mathsf{sk}_S = g^{k_S{}^2}$ and keeps the session identifier $\mathsf{sid}_S = g^\alpha|g^\beta|\mathsf{pid}_S$.

Upon receiving msg_S, the client C firstly computes $k_C = g^{\alpha\beta}$ and then verifies if

$$\mathsf{Auth}_S = g^{h(g^\alpha|g^\beta|\mathsf{pid}_C|\mathsf{pw}_C^S)+k_C} \quad (2)$$

If (2) holds, the client C computes the final session key $\mathsf{sk}_C = g^{k_C{}^2}$ and keeps the session identifier $\mathsf{sid}_C = g^\alpha|g^\beta|\mathsf{pid}_C$.

Remark: Due to the special structure of G, the group element k_S (or k_C) is corresponding to a binary vector $(k_1, k_2, \cdots, k_p)$, which is treated as an integer from Z_q^*.

Correctness. In an execution of the protocol, if $\mathsf{sid}_C = g^{\alpha_C}|g^{\beta_C}|\mathsf{pid}_C = \mathsf{sid}_S = g^{\alpha_S}|g^{\beta_S}|\mathsf{pid}_S \neq \mathsf{NULL}$, then $\alpha_C = \alpha_S$, $\beta_C = \beta_S$, and $\mathsf{pid}_C = \mathsf{pid}_S = \{C,S\}$. If $\mathsf{acc}_C = \mathsf{acc}_S = \mathsf{TRUE}$, then $\mathsf{sk}_C = g^{k_C{}^2}$ where $k_C = g^{\alpha_C\beta_C}$, and $\mathsf{sk}_S = g^{k_S{}^2}$ where $k_S = g^{\alpha_S\beta_S}$. If $\mathsf{sid}_C = \mathsf{sid}_S \neq \mathsf{NULL}$ and $\mathsf{acc}_C = \mathsf{acc}_S = \mathsf{TRUE}$, then $\mathsf{sk}_C = \mathsf{sk}_S$.

Explicit Authentication. Generally speaking, a key exchange protocol achieves explicit authentication if a party knows that its intended partner has successfully computed a matching session key. By verifying equation (1), the server knows that g^α does come from the client. Therefore, the server makes sure that as long as the client receives $\mathsf{msg}_S = S|g^\beta|\mathsf{Auth}_S$, he must compute a matching session key $\mathsf{sk}_C = \mathsf{sk}_S$. By verifying equation (2), the client is sure that g^β does come from the server and knows that the server has successfully computed the matching session key $\mathsf{sk}_S = \mathsf{sk}_C$. In the settings where the client needs to remember passwords only, previous PAKE protocols achieve implicit authentication only (e.g., (Katz et al., 2001)) or explicit authentication with three-round communications (e.g., (Jiang and Gong, 2004)). Our protocol achieves explicit authentication with only two-round communications.

Efficiency Consideration. The efficiency of our protocol depends on performance of the underlying IBE

system. If we employ Waters' IBE scheme (Waters, 2005), where $e(g_1, g_2)$ are included in the public parameters, the client needs to compute 7 exponentiations without pairing. In addition, the client needs to send and receive 6 group elements. In the KOY protocol, the client needs to compute 15 exponentiations plus signature generation and exchange 10 group elements plus a signature and a verification key. Therefore, on the side of the client, our protocol is more efficient than the KOY protocol.

Example. According to RFC 5091 (Boyen and Martin, 2007), if a security parameter k is 1024, the ciphertext size of the Waters' IBE scheme is about $1024 + 160 \times 2 = 1344$ bits. In addition, we use a large Mersenne prime $q = 2^{1279} - 1$, where 1279 is a prime as well, to construct the group G. With reference to Fig. 1 in this setting, the size of the message msg_C sent to the server from the client is about $1344 + 1279 = 2623$ bits, while the size of the message msg_S replied to the client from the server is about $1279 \times 2 = 2558$ bits.

4 PROOF OF SECURITY

First of all, we provide a formal specification of the protocol by specifying the initialization phase and the oracles to which the adversary has access. A formal specification of the Initialize, Execute, KeyGen, Reveal, Test, and Send oracles appears in Fig. 2−4. The description of the Execute oracle matches the high-level protocol described in Fig. 1, but additional details (for example, the updating of state information) are included. We let status_U^i denote the vector of values $(\mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{acc}_U^i, \mathsf{term}_U^i)$ associated with instance U^i.

Theorem 1. Assume that (1) the IBE scheme, where there exist n private key generators and at least one of them is honest, is secure against the chosen-ciphertext attack; (2) the decisional Diffie-Hellman (DDH) problem and the squaring decisional Diffie-Hellman (SDDH) problem are hard over a cyclic group G with a prime order q and a generator g; (3) CRHF is a collision-resistant hash family; then the protocol described in Fig. 1 is a secure protocol for PAKE-CS.

Remark. The SDDH problem here is to distinguish between two distributions (g, g^a, g^{a^2}) and (g, g^a, z), where a is randomly chosen from Z_q^* and z is randomly chosen from G.

Proof. We follow the methodology of the security proof given by Katz et al. in (Katz et al., 2001).

Given an adversary $\mathcal{A}$, we imagine a simulator that runs the protocol for $\mathcal{A}$. Preciously, the simulator begins by running algorithm $\mathsf{Initialize}(1^k)$ as shown in Fig. 2 and giving the public output of the algorithm to $\mathcal{A}$. When $\mathcal{A}$ queries an oracle, the simulator responds to $\mathcal{A}$ by executing the appropriate algorithm as shown in Fig. 3−4.

In particular, when the adversary queries the Test oracle, the simulator chooses (and records) the random bit b.

When the adversary completes its execution and outputs a bit b', the simulator can tell whether the adversary succeeds by checking whether (1) a single Test query was made, for some instance U^i; (2) acc_U^i was true at the time of Test query; (3) instance U^i is fresh; and (4) $b' = b$. Success of the adversary is denoted by event Succ. For any experiment P, we define $\mathsf{Adv}_{\mathcal{A}}^P(k) = 2Pr_{\mathcal{A}}^P[\mathsf{Succ}] - 1$, where $Pr_{\mathcal{A}}^P[\cdot]$ denotes the probability of an event when the simulator interacts with $\mathcal{A}$ in accordance with experiment P.

$\mathsf{Initialize}(1^k)$
$(\mathsf{params}^E, \mathsf{master\text{-}key}^E) \xleftarrow{R} \mathsf{Setup}^E(1^k)$
$(\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}) \xleftarrow{R} \mathsf{UserGen}(1^k)$
$(G, q, g) \xleftarrow{R} \mathsf{GGen}(1^k), \{h, H\} \xleftarrow{R} \mathsf{CRHF}(1^k)$
For each $i \in \{1, 2, \cdots\}$ and each $U \in \mathsf{User} = \mathsf{Client} \cup \mathsf{Server}$
  $\mathsf{acc}_U^i \leftarrow \mathsf{term}_U^i \leftarrow \mathsf{used}_U^i \leftarrow \mathsf{FALSE}$
  $\mathsf{sid}_U^i \leftarrow \mathsf{pid}_U^i \leftarrow \mathsf{sk}_U^i \leftarrow \mathsf{NULL}$
For $\forall S \in \mathsf{Server}$, $\mathsf{d}_{\mathsf{ID}_S} \leftarrow \mathsf{Extract}^E(\mathsf{ID}_S, \mathsf{params}^E, \mathsf{master\text{-}key}^E)$
For $\forall (C, S) \in \mathsf{ClientServerPair}$, $\mathsf{pw}_C^S \xleftarrow{R} \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$
Return $\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}, h, H, G, q, g, \mathsf{params}^E$

Figure 2: Specification of the initialize.

$\mathsf{Execute}(C^i, S^j)$
  If $(C, S) \notin \mathsf{ClientServerPair} \vee \mathsf{used}_C^i \vee \mathsf{used}_S^j$, return $\perp$
  $\mathsf{used}_C^i \leftarrow \mathsf{used}_S^j \leftarrow \mathsf{TRUE}, \mathsf{pid}_C^i \leftarrow \mathsf{pid}_S^j \leftarrow \{C^i, S^j\}$
  $\alpha \xleftarrow{R} \mathbb{Z}_q, \mathsf{Auth}_C \leftarrow \mathsf{IBE}_{\mathsf{ID}_S}[H(g^\alpha | \mathsf{pid}_C^i | \mathsf{pw}_C^S)]$
  $\mathsf{msg}_C \leftarrow C^i | g^\alpha | \mathsf{Auth}_C$
  $\beta \xleftarrow{R} \mathbb{Z}_q, K \leftarrow g^{\alpha\beta}, \mathsf{Auth}_S \leftarrow g^{h(g^\alpha | g^\beta | \mathsf{pid}_S^j | \mathsf{pw}_C^S) + K}$
  $\mathsf{msg}_S \leftarrow S^j | g^\beta | \mathsf{Auth}_S$
  $\mathsf{acc}_C^i \leftarrow \mathsf{term}_C^i \leftarrow \mathsf{acc}_S^j \leftarrow \mathsf{term}_S^j \leftarrow \mathsf{TRUE}$
  $\mathsf{sid}_C^i \leftarrow \mathsf{sid}_S^j \leftarrow g^\alpha | g^\beta | \{C^i, S^j\}$
  $\mathsf{sk}_C^i \leftarrow \mathsf{sk}_S^j \leftarrow g^{K^2}$
  Return $\mathsf{status}_C^i, \mathsf{status}_S^j$

$\mathsf{KeyGen}(PKG_i, S)$
  Return $d_{S,i}$

$\mathsf{Corrupt}(S)$
  Return d_{ID_S} and pw_C^S for any C

$\mathsf{Reveal}(U^i)$
  Return sk_U^i

$\mathsf{Corrupt}(C)$
  Return pw_C^S for any S

$\mathsf{Test}(U, i)$
  $b \xleftarrow{R} \{0, 1\}, \mathsf{sk}' \xleftarrow{R} G$; If $b = 1$ return sk' else return sk_U^i

Figure 3: Specification of Execute, KeyGen, Reveal and Test.

$\mathsf{Send}_0(C^i, S^j)$
If $(C,S) \notin \mathsf{ClientServerPair} \vee \mathsf{used}_C^i$, return $\perp$
$\mathsf{used}_C^i \leftarrow \mathsf{TRUE}, \mathsf{pid}_C^i \leftarrow \{C^i, S^j\}$
$\alpha \xleftarrow{R} \mathbb{Z}_q, \mathsf{Auth}_C \leftarrow \mathsf{IBE}_{\mathsf{ID}_S}[H(g^\alpha | \mathsf{pid}_C^i | \mathsf{pw}_C^S)]$
$\mathsf{MsgOut} \leftarrow C^i | g^\alpha | \mathsf{Auth}_C, \mathsf{state}_C^i \leftarrow (\alpha, \mathsf{pid}_C^i, \mathsf{MsgOut})$
Return $\mathsf{MsgOut}, \mathsf{status}_C^i$

$\mathsf{Send}_1(S^j, C^i | g^\alpha | \mathsf{Auth}_C)$
If $(C,S) \notin \mathsf{ClientServerPair} \vee \mathsf{used}_S^j$, return $\perp$
$\mathsf{used}_S^j \leftarrow \mathsf{TRUE}, \mathsf{pid}_S^j \leftarrow \{C^i, S^j\}$
If $\mathsf{D}_{\mathsf{d}_{\mathsf{ID}_S}}[\mathsf{Auth}_C] \neq H(g^\alpha | \mathsf{pid}_S | \mathsf{pw}_C^S)$, return status_S^j
Else $\beta \xleftarrow{R} \mathbb{Z}_q, K \leftarrow g^{\alpha\beta}, \mathsf{Auth}_S \leftarrow g^{h(g^\alpha | g^\beta | \mathsf{pid}_S^j | \mathsf{pw}_C^S) + K}$
$\mathsf{MsgOut} \leftarrow S^j | g^\beta | \mathsf{Auth}_S$
$\mathsf{acc}_S^j \leftarrow \mathsf{term}_S^j \leftarrow \mathsf{TRUE}, \mathsf{sk}_S^j \leftarrow g^{K^2}, \mathsf{sid}_S^j \leftarrow g^\alpha | g^\beta | \mathsf{pid}_S^j$
Return $\mathsf{MsgOut}, \mathsf{status}_S^j$

$\mathsf{Send}_2(C^i, S^j | g^\beta | \mathsf{Auth}_S)$
$\mathsf{state}_C^i \leftarrow (\alpha, \mathsf{pid}_C^i, \mathsf{FirstMsgOut})$
If $\neg \mathsf{used}_C^i \vee \mathsf{term}_C^i \vee (S^j \notin \mathsf{pid}_C^i)$, return $\perp$
$K \leftarrow g^{\alpha\beta}$. If $\mathsf{Auth}_S \neq g^{h(g^\alpha | g^\beta | \mathsf{pid}_C^i | \mathsf{pw}_C^S) + K}$, return status_C^i
Else $\mathsf{acc}_C^i \leftarrow \mathsf{term}_C^i \leftarrow \mathsf{TRUE}, \mathsf{sk}_C^i \leftarrow g^{K^2}, \mathsf{sid}_C^i \leftarrow g^\alpha | g^\beta | \mathsf{pid}_C^i$
Return status_C^i

Figure 4: Specification of the Send oracles.

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

We begin with some terminology that will be used throughout the proof. A given msg is called oracle-generated if it was output by the simulator in response to some oracle query (whether a Send or Execute query). The message is said to be adversarially-generated otherwise. An adversarially-generated message must not be the same as any oracle-generated message.

Experiment P_1. In this experiment, the simulator interacts with the adversary as before except that any of the following never occurs:

1. At any point during the experiment, an oracle-generated message (e.g., msg_C or msg_S) is repeated.
2. At any point during the experiment, a collision occurs in the hash functions h, 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 and 2 occur with only negligible probability assuming the security of CRHF as a collision-resistant hash family. Therefore,

Claim 1. If CRHF is a collision-resistant hash family, then $|\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_1}(k)|$ 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 Execute oracles are handled differently: for any $\mathsf{Execute}(C^i, S^j)$ oracle, the value of K is replaced with a random value from G.

The difference between experiments P_2 and P_1 is bounded by the probability that an adversary solves the DDH problem. More precisely, we have

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

Experiment P_3. In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_2 except that the adversary's queries to Execute oracles are handled differently: for any $\mathsf{Execute}(C^i, S^j)$ oracle, the session key sk_C^i and sk_S^j are replaced with a random element from G.

The difference between experiments P_3 and P_2 is bounded by the probability to solve the SDDH problem. More precisely, we have

Claim 3. If the SDDH problem is hard over (G, q, g), then $|\mathsf{Adv}_{\mathcal{A}}^{P_2}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_3}(k)|$ is negligible.

In experiment P_3, the adversary's probability of correctly guessing the bit b used by the Test oracle is exactly 1/2 if the Test query is made to a fresh instance invoked by an Execute oracle. Therefore, all passive adversaries (including the PKG) cannot win the game, even if they can query $\mathsf{KeyGen}(PKG_i, *)$ for $i = 1, 2, \cdots, n$ and Corrupt oracles. The remainder of the proof discusses instances invoked by Send oracles.

Experiment P_4. In this experiment, we modify the simulator's responses to Send_1 and Send_2 queries.

At first, we introduce some terminology. For a query $\mathsf{Send}_1(S^j, \mathsf{msg}_C)$ (or $\mathsf{Send}_2(C^i, \mathsf{msg}_S)$), where msg_C (or msg_S) is adversarially-generated, if equation (1) (or (2)) holds, then msg_C (or msg_S) is said to be valid. Otherwise, msg_C (or msg_S) is said to be invalid. Informally, valid messages use correct passwords while invalid messages do not.

When the adversary makes an oracle query $\mathsf{Send}_1(S^j, \mathsf{msg}_C)$ to a fresh sever instance S^j, the simulator examines msg_C. If it is adversarially-generated and valid, the simulator halts and acc_S^j is assigned the special value ∇. In any other case, (i.e., msg_C is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment P_3. When the adversary makes an oracle query $\mathsf{Send}_2(C^i, \mathsf{msg}_S)$ to a fresh client instance C^i, the simulator examines msg_S. If msg_S is adversarially-generated and valid, the simulator halts and acc_C^i is assigned the special value ∇. In any other case, the

query is answered exactly as in experiment P_3. Now, we change the definition of the adversary's success in P_4. If the adversary ever queries $\mathsf{Send}_1(S^j,*)$ to a fresh server instance S^j with $\mathsf{acc}_S^j = \nabla$ or $\mathsf{Send}_2(C^i,*)$ 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_3.

The distribution on the adversary's view in experiments P_3 and P_4 are identical up to the point when the adversary queries Send_1 or Send_2 to a fresh user instance with $\mathsf{acc}_S^j = \nabla$ or $\mathsf{acc}_C^i = \nabla$. If such a query is never made, the distributions on the view are identical. Therefore, we have

Claim 4. $\mathsf{Adv}_{\mathcal{A}}^{P_3}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_4}(k)$.

In experiment P_4, the adversary $\mathcal{A}$ succeeds if one of the following occurs: (1) $\mathsf{acc}_S^j = \nabla$ (let Succ_1 denote this event); (2) $\mathsf{acc}_C^i = \nabla$ (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 user instance A^i. To evaluate $\mathsf{Pr}_{\mathcal{A}}^{P_4}[\mathsf{Succ}_1 \vee \mathsf{Succ}_2]$, we do the following experiments.

Experiment P_5. In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_4 except that the adversary's queries to $\mathsf{Execute}$ and Send_0 oracles are handled differently: for $\mathsf{Execute}(C^i,S^j)$ (after which there exist any query of the form $\mathsf{Send}(U^\ell,*)$ where $U \in \{C,S\}$) or $\mathsf{Send}_0(C^i,S^j)$ queries to the fresh instances C^i and S^j, Auth_C is computed as $\mathsf{E}_{\mathsf{ID}_S}[H(g^\alpha \,|\mathsf{pid}_C^i|\mathsf{pw}')]$ where pw' is randomly chosen from $\mathcal{D}$.

Claim 5. If the IBE scheme where there are n private key generators and at least one of them is honest is secure against the chosen-ciphertext attack, then $|\mathsf{Adv}_{\mathcal{A}}^{P_4}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_5}(k)|$ is negligible.

Experiment P_6. In this experiment, the simulator interacts with $\mathcal{A}$ as in experiment P_5 except that the adversary's queries to Send_1 and Send_2 oracles are handled differently: for $\mathsf{Send}_1(S^j,*)$ or $\mathsf{Send}_2(C^i,*)$ queries, Auth_S is replaced by a random element from G.

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

For any adversarially-generated $\mathsf{Send}_1(S^j,*)$ or $\mathsf{Send}_2(C^i,*)$ queries to the fresh instances C^i and S^j in experiment P_6, all $\mathsf{Execute}$ and Send queries are independent of the password pw_C^S in the view of the adversary. In order to win the game by Succ_1 or Succ_2, the adversary has to try all passwords one-by-one in an online impersonation attack. The probability that Succ_1 or Succ_2 occurs is at most $Q(k)/N$, where $Q(k)$ is the number of online attacks made by $\mathcal{A}$.

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

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

and therefore $\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_6}(k) + \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$. This completes the proof of the theorem.

5 CONCLUSIONS

In this paper, we have presented an identity-based PAKE protocol for client/service model. Our protocol is based on identity-based encryption and needs only two-round communication between the client and the server to achieve explicit authentication. Our future work is to machine-validate our security proofs using a cryptographic proof-assistant, such as EasyCrypt (Barthe et al., 2011).

REFERENCES

Abdalla, M. and Pointcheval, D. (2005). Simple password-based encrypted key exchange protocols. In *Proc. CT-RSA 2005*, pages 191–208.

Barthe, G., Grgoire, B., Heraud, S., and Bguelin, S. Z. (2011). Computer-aided security proofs for the working cryptographer. In *Proc. Crypto'11*, pages 71–90.

Bellare, M., Pointcheval, D., and Rogaway, P. (2000). Authenticated key exchange secure against dictionary attacks. In *Proc. Eurocrypt'00*, pages 139–155.

Bellovin, S. M. and Merritt, M. (1992). Encrypted key exchange: Password-based protocol secure against dictionary attack. In *Proc. 1992 IEEE Symposium on Research in Security and Privacy*, pages 72–84.

Boneh, D. and Franklin, M. (2001). Identity based encryption from the weil pairing. In *Proc. Crypto'01*, pages 213–229.

Boyen, X. and Martin, L. (2007). Identity-based cryptography standard (ibcs) 1: Supersingular curve implementations of the bf and bb1 cryptosystems. In *RFC 5091*.

Boyko, V., Mackenzie, P., and Patel, S. (2000). Provably secure password-authenticated key exchange using diffie-hellman. In *Proc. Eurocrypt'00*, pages 156–171.

Bresson, E., Chevassut, O., and Pointcheval, D. (2003). Security proofs for an efficient password-based key exchange. In *Proc. CCS'03*.

Cakulev, V., Sundaram, G., and Broustis, I. (2012). Ibake: Identity-based authenticated key exchange. In *RFC 6539*.

Canetti, R., Halevi, S., Katz, J., Lindell, Y., and MacKenzie, P. (2005). Universally composable password-based key exchange. In *Proc. Eurocrypt'05*, pages 404–421.

Gennaro, R. and Lindell, Y. (2003). A framework for password-based authenticated key exchange. In *Proc. Eurocrypt'03*, pages 524 –543.

Goldreich, O. and Lindell, Y. (2001). Session-key generation using human passwords only. In *Proc. Crypto'01*, pages 408–432.

Gong, L., Lomas, T. M. A., Needham, R. M., and Saltzer, J. H. (1993). Protecting poorly-chosen secret from guessing attacks. *IEEE J. on Selected Areas in Communications*, 11(5):648–656.

Halevi, S. and Krawczyk, H. (1999). Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):230–268.

Huang, H. F. (1996). Strong password-only authenticated key exchange. *ACM Computer Communication Review*, 26(5):5–20.

Jiang, S. and Gong, G. (2004). Password based key exchange with mutual authentication. In *Proc. SAC'04*, pages 267–279.

Katz, J., Ostrovsky, R., and Yung, M. (2001). Efficient password-authenticated key exchange using human-memorable passwords. In *Proc. Eurocrypt'01*, pages 457–494.

Katz, J., Ostrovsky, R., and Yung, M. (2003). Forward secrecy in password-only key exchange protocols. In *Proc. SCN'03*, pages 29–44.

Lamport, L. (1981). Password authentication with insecure communication. *Comm. of the ACM*, 24(11):770–772.

MacKenzie, P., Patel, S., and Swaminathan, R. (2000). Password-authenticated key exchange based on rsa. In *Proc. Asiacrypt'00*, pages 599–613.

Nguyen, M. H. and Vadhan, S. P. (2004). Simpler session-key generation from short random passwords. In *Proc. Theory of Cryptography'04*, pages 428–445.

Patel, S. (1997). Number-theoretic attack on secure password scheme. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 236–247.

Waters, B. (2005). Efficient identity-based encryption without random oracles. In *Proc. Eurocrypt'05*, pages 114–127.

Wu, T. (1998). The secure remote password protocol. In *Proc. Internet Society Symp. on Network and Distributed System Security*, pages 97–111.

Yi, X., Tso, R., and Okamoto, E. (2009). Id-based group password-authenticated key exchange. In *Proc. IWSEC'09*, pages 192–211.

Yi, X., Tso, R., and Okamoto, E. (2011). Three-party password-authenticated key exchange without random oracles. In *Proc. SECRYPT'11*, pages 15–24.