Group Password-Authenticated Key Exchange from Identity-Based Cryptosystem

Xun Yi[1], Raylin Tso[2] and Eiji Okamoto[2]

[1]School of Computer Science and Mathematics
Victoria University, PO Box 14428, Melbourne City MC
Victoria 8001, Australia
E-mail: xun.yi@vu.edu.au

[2]Department of Risk Engineering
Graduate School of Systems and Information Engineering
University of Tsukuba, 1-1-1 Tennodai, Tsukuba City
Ibaraki, 305-8573 Japan
E-mail: {raylin, okamoto}@risk.tsukuba.ac.jp

Abstract. Password-authenticated key exchange (PAKE) protocols are designed to be secure even when the secret key used for authentication is a human-memorable password. In this paper, we consider PAKE protocols in the group scenario, in which a group of clients, each of them shares a password with an "honest but curious" server, intend to establish a common secret key (i.e., a group key) with the help of the server. In this setting, the key established is known to the clients only and no one else, including the server. Each client needs to remember passwords only while the server keeps passwords in addition to private keys related to his identity. Towards our goal, we present the first compiler that transforms any group key exchange (KE) protocol secure against a passive eavesdropping to a group PAKE which is secure against an active adversary who controls all communication in the network. This compiler is built on any group KE protocol (e.g., the Burmester-Desmedt protocol), any identity-based encryption (IBE) scheme (e.g., Gentry's scheme), and any identity-based signature (IBS) scheme (e.g., Paterson-Schuldt scheme). It adds only two rounds and $O(1)$ communication (per client) to the original group KE protocol. As long as a group PAKE protocol is constructed by our compiler with a group KE protocol, an IBE scheme and an IBS scheme which have provably security without random oracles, it can be proven to be secure without random oracles.

1 Introduction

Popularity of group-oriented applications and protocols is currently on the increase and, as a result, group communication is taking place in many different settings, from network layer multicasting to application layer tele- and video-conferencing. Securing group communication makes demands of protocols for group authenticated key exchange (AKE), which allows a group of users communicating over an insecure public network to establish a common secret key

(i.e., a group key) and furthermore to be guaranteed that they are indeed sharing this key with each other.

Protocols for 2-party AKE has been extensively investigated in [33, 14, 34, 12, 10, 30–32]. A number of works have considered extending the 2-party Diffie-Hellman protocol [33] to the multi-party setting [39, 51, 28, 52, 9, 46, 47]. Among them, the works of Ingemarsson et al. [39], Burmester and Desmedt [28], and Steiner et al. [52] may be the most well-known. They are merely key exchange (KE) protocols, intended to be secure against a passive adversary only. However, AKE protocols aim to be secure against more powerful adversaries, who - in addition to eavesdropping - control all communication in the network. A number of initial protocols for group AKE were suggested in [40, 18, 7, 8, 53]. But none of these works have rigorous security proofs in a well-defined model.

Bresson et al. [21–23] were the first to define a formal model of security for group AKE and give the first provably secure protocols for this setting. Their model was built on the earlier work of Bellare and Rogaway in the two-party setting [11, 12, 10] and their protocols were based on the work of Steiner et al. [52], which requires $O(n)$ rounds to establish a key among n users, and therefore not scalable. A constant-round group AKE with a security proof in the random oracle model was given in [19]. Katz and Yung [44] were the first to give scalable protocol for group AKE along with a rigorous proof of security in the standard model. They also presented the first efficient compiler that transforms any group KE protocols secure against a passive eavesdropping to authenticated protocols by signing message flows. Their compiler adds only one round to the original protocol. However, this compiler requires each user to have a pair of public and private keys for digital signature. The (high-entropy) private key is not human-memorable and needs additional cryptographic devices to store it.

Bellovin and Merritt [13] were the first to consider AKE based on (low-entropy) password only and introduced a series of so-called "encrypted key exchange" (EKE) protocols for two-party AKE. A password-based AKE (i.e., PAKE) 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. Even though these attacks are not very effective in the case of high-entropy keys, they can be very damaging when the secret key is a password since the attacker has a non-negligible chance of winning. Dictionary attacks are usually divided into two categories: offline and online dictionary attacks. Formal models of security for two-party PAKE were firstly given independently by Bellare, Pointcheval and Rogaway [10], and Boyko, MacKenzie, Patel and Swaminathan [20] in 2000. Since then, protocols for two-party PAKE have been continuously proposed and proven to be secure in either the random oracle model (e.g.,[25, 26, 3–5]) or the standard model (e.g., [37, 42, 41]).

Bresson et al. [24, 27] were the first to adapt a group KE protocol to the password-based scenario. As the original protocol, the first group PAKE protocol was not scalable and practical for large groups. In addition, their security proof required ideal models. Recently, a number of constant-round group PAKE have been proposed in the literature by Abdalla et al. [2, 6], by Bohli et al. [15], and

by Kim, Lee and Lee [45]. All of these constructions are built on the Burmester-Desmedt protocol [28, 29] and are rather efficient. Among them, the works of Abdalla et al. [6] and Bohli et al. [15] enjoy security proofs in the standard model.

Most of existing group PAKE protocols assume that users of a group share the same password, e.g., [24, 27, 2, 6]. In the scenarios where a user wants to participate in many groups, the number of passwords that he would need to remember would be linear in the number of possible groups. In order to limit the number of passwords that each user has to remember, a couple of group PAKE protocols assume that each user shares a password only with a server, which helps users of a group with establishment of a common secret key (i.e., a group key), e.g., [3, 4, 48]. The server is assumed to behave in an "honest but curious" manner. By the knowledge of passwords, the server may attempt to learn the group key. The setting with different passwords seems to be more practical in the real world than the setting with the same password.

More recently, Abdalla et al. [1] presented a protocol compiler that transforms any two-party AKE into a group AKE with two more rounds of communication. Their idea is inspired by the construction of Burmester and Desmedt [28], where the trick of constructing a group key from pairwise agreed keys among users of a group was firstly introduced. In particular, applying this compiler to a two-party PAKE protocol yields a group PAKE protocol. The primary motivation of this compiler was the two-party setting. As implied in [44], a compiler tailored from the group setting scales better than the compiler from two-party setting. This leads a question, is there any protocol compiler that transforms any group KE protocol directly to a group PAKE protocol?

Contribution. To the best of our knowledge, there has not yet been any protocol compiler that can transform any group KE protocol directly into a group PAKE protocol at present. In this paper, we present such a compiler on the basis of the "state-of-the-art" identity-based cryptosystem, a public-key cryptosystem in which an arbitrary string (e.g., user identity) can be used as the public key.

Our compiler employs any group KE protocol secure against passive eavesdropping, any IBE with chosen-ciphertext security and any IBS with existential unforgeability. We assume that clients of a group, each of them shares a password with an "honest but curious" server, intend to establish a common secret key (i.e., a group key) with the help of the server, where the key established is known to the clients only and no one else, including the server.

The basic idea of our compiler is that users of a group firstly run the group KE protocol to establish a group key without any help of the server, and then the server helps users of the group with mutual authentication by the shared password (protected with the IBE scheme), and finally each user authenticates the server, along with partnered users and messages received during the group KE, by the IBS scheme. Since the group key has been established before real participance of the server, it is unknown to the server.

To analyze the security of our compiler, we put forth the first formal model of security for ID-based PAKE in the group setting, by embedding Boneh et

al.'s ID-based model [16][17] into the group PAKE model given by Bresson et al. in [24, 27] and improved by Abdalla et al. [1]. We then 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. By using Burmester-Desmedt group KE protocol [28], Gentry IBE scheme [36], Paterson-Schuldt IBS scheme [49], our compiler can construct a group PAKE with provably security in the standard model.

The protocol compiler for group PAKE given by Abdalla et al. [1] has to assume that each user is honest. Otherwise, it is vulnerable to a collusion attack, where two dishonest users conspire to include a few impersonators between them. Other honest users are unaware of this attack. Although a dishonest user can always disclose the group key to others, this collusion attack is meaningful in the case where users of a group are holding a tele-conference for voting. Our compiler assumes that there exists an "honest but curious" server for authentication only. This assumption is weaker (in both a theoretical and practical sense) than the compiler in [1].

Organization. In Section 2, we introduce the new model for ID-based group PAKE. Next, in Section 3, we describe the underlying cryptographic primitives to build our group PAKE. Then, in Section 4, we present the new ID-based group PAKE compiler. After that, in Section 5, the detail security proof for our protocol is given. We conclude this paper in Section 6.

2 Definitions

A formal model of security for group PAKE was firstly given by Bresson et al. in [24, 25] (based on Bellare et al.'s formal model for 2-party PAKE [11]), and improved by Abdalla et al. in [1]. Boneh and Franklin were the first to define chosen ciphertext security for IBE under chosen identity attack [16, 17]. In this section, we put forward the first model of security for ID-based group PAKE, on the basis of definitions given by Bresson et al., Abdalla et al. and Boneh et al.

Participants, Initialization and Passwords. An ID-based group PAKE protocol involves three kinds of participants: (1) A set of clients (denoted as Client); (2) A set of servers (denoted as Server), who behave in an honest but curious manner; (3) A trusted third party (called the Private Key Generator (PKG)), which generates public parameters and corresponding private keys for servers. We assume that $\mathsf{ClientServerPair}$ is the set of pairs of the client and the server, who share a password. In addition, $\mathsf{User} = \mathsf{Client} \bigcup \mathsf{Server}$ and $\mathsf{Client} \bigcap \mathsf{Server} = \emptyset$.

Prior to any execution of the protocol, we assume that an initialization phase occurs. During initialization, PKG generates public parameters for the protocol, which are available to all participants, and private keys for each server. For any pair $(A, S) \in \mathsf{ClientServerPair}$, the client A and the server S are assumed to share the same password pw_A^S. We assume that the client A chooses pw_A^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. The password pw_A^S is then stored at the server S for authentication.

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.

- sid_U^i and pid_U^i are variables (initialized as NULL) containing the session identity and partner identity for an instance, respectively. The session identity sid_U^i 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 pid_U^i is the set of users with whom U^i believes it is interacting to establish a session key (including U itself).
- acc_U^i and term_U^i are boolean variables (initialized as FLASE) denoting whether a given instance 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 a group key has been established, thus, when an instance U^i accepts, sid_U^i and pid_U^i are no longer NULL.
- used_U^i is a boolean variable (initialized as FLASE) denoting whether an instance has begun executing the protocol. This is a formalism which will ensure each instance is used only once.
- state_U^i (initialized as NULL) records any state necessary for execution of the protocol by a user instance U^i.
- sk_A^i is a variable (initialized as NULL) containing the session key for a client instance A^i. Computation of the session key is, of course, the ultimate goal of the protocol. When A^i accepts (i.e., $\mathsf{acc}_A^i = \mathsf{TRUE}$), sk_A^i is no longer NULL.

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 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 types are as follows:

- $\mathsf{Execute}(A_1^{i_1}, A_2^{i_2}, \cdots, A_n^{i_n}, S^j)$ – If $A_\ell^{i_\ell}$ and S^j have not yet been used (where $A_\ell \in \mathsf{Client}$, $S \in \mathsf{Server}$, $(A_\ell, S) \in \mathsf{ClientServerPair}$, $\ell = 1, 2, \cdots, n$). This oracle executes the protocol among 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 all instances, at each step of protocol execution.

- $\mathsf{Send}(U^i, M)$ – 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.
- $\mathsf{KeyGen}(\mathsf{PKG}, S)$ – This sends the identity of the server S to PKG, which generates private keys $\mathsf{d_S}$ corresponding to S and forwards it to the adversary. This oracle models possible compromising of a server due to, for example, hacking into the server. This implies that all passwords stored in the server are disclosed.
- $\mathsf{Corrupt}(A)$ – This query allows the adversary to learn the passwords of the client A, 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 A are disclosed.
- $\mathsf{Reveal}(A^i)$ – This outputs the current value of session key sk_A^i for a client instance if $\mathsf{acc}_A^i = \mathsf{TRUE}$. This oracle call models possible leakage of session keys due to, for example, improper erasure of session keys after use, or cryptanalysis.
- $\mathsf{Test}(A^i)$ – This oracle does not model any real-world capability of the adversary, but is instead used to define security of the session key of client instance A^i. If $\mathsf{acc}_A^i = \mathsf{TRUE}$, a random bit b is generated. If $b = 0$, the adversary is given sk_A^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.

A passive adversary is given access to the $\mathsf{Execute}$, KeyGen, Reveal, $\mathsf{Corrupt}$, and Test oracles, while an active adversary is additionally given access to the Send oracles. In the definition of $\mathsf{Execute}$ and Send oracle, we reasonably require that $A_1, A_2, \cdots, A_n$ share different passwords with the same server S.

Partnering. The definition of partnering uses the notion of session identity sid, which is the partial transcript of the conversation among the clients and the server. We say that client instances A^i and B^j are partnered if there exists a server instance S^k such that (1) $(A, S), (B, S) \in \mathsf{ClientServerPair}$; (2) $\mathsf{sid}_A^i = \mathsf{sid}_B^j = \mathsf{sid}_S^k \neq \mathsf{NULL}$, and (2) $\mathsf{pid}_A^i = \mathsf{pid}_B^j = \mathsf{pid}_S^k \neq \emptyset$. The notion of partnering will be fundamental in defining both correctness and security.

Correctness. To be viable, a key exchange protocol must satisfy the following notion of correctness: if A^i and B^j are partnered and $\mathsf{acc}_A^i = \mathsf{acc}_B^j = \mathsf{TRUE}$, then it must be the case that $\mathsf{sk}_A^i = \mathsf{sk}_B^j$ (i.e., they conclude with the same session key).

Freshness. 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 client instance A^i is fresh unless one of the following is true at the conclusion of the experiment, namely, at some point,

– The adversary queried $\mathsf{Reveal}(U^j)$ where $U \in \mathsf{Client} \bigcap \mathsf{pid}_A^i$. Note that $j = i$ when $U = A$.
– The adversary queried $\mathsf{KeyGen}(\mathsf{PKG}, S)$ where $S \in \mathsf{Server} \bigcap \mathsf{pid}_A^i$, before a query of the form $\mathsf{Send}(U^j, M)$, where $U \in \mathsf{pid}_A^i$, has taken place, for some message M (or identities).
– The adversary queried $\mathsf{Corrupt}(B)$, where $B \in \mathsf{Client} \bigcap \mathsf{pid}_A^i$, before a query of the form $\mathsf{Send}(U^j, M)$, where $U \in \mathsf{pid}_A^i$, has taken place, for some message M (or identities).

The adversary is thought to succeed only if its Test query is made to a fresh instance. Note that this is necessary for any reasonable definition of security, otherwise, the adversary could always succeed, e.g., submitting a Test query for an instance for which it had already submitted a Reveal query.

Advantage of the Adversary. We say an adversary $\mathcal{A}$ succeeds if it makes a single query $\mathsf{Test}(A^i)$ to a fresh client instance A^i, with $\mathsf{acc}_A^i = \mathsf{TRUE}$ 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 a function in the security parameter k, defined as

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

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). It remains to define what we mean by a secure protocol. Note that a probabilistic polynomial-time (PPT) adversary can always succeed by trying all passwords one-by-one in an online impersonation attack. This is possible since the size of the password dictionary is constant. Informally, a protocol is secure if this is the best an adversary can do. Formally, an instance U^i represents an online 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 $\mathsf{Reveal}(A^j)$ or $\mathsf{Test}(A^j)$, where $A \in \mathsf{Client} \bigcap \mathsf{pid}_U^i$. In particular, instances with which the adversary interacts via Execute, KeyGen, Reveal and Corrupt queries are not counted as online attacks. The number of online attacks represents a bound on the number of passwords the adversary could have tested in an online 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 $Q(k)$ online attacks, there exists a negligible function $\varepsilon(\cdot)$ such that

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

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

Forward Secrecy. We follow the definition of forward secrecy from [43, 1] and consider the weak corrupt model of [11], in which corrupting a client means retrieving his passwords, while asking KeyGen query on a server means retrieving its private keys 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. In addition, we follow the definition of freshness from [1]. The adversary is allowed to ask the Test query on a client instance, where he has known (1) the passwords of the client or any of his partners by Corrupt query; or (2) the private key of the server and all password stored in it by KeyGen query, however, he has not asked any Send query to the instance of the client or any of his partners. In this sense, the above definition of security implies forward secrecy.

3 Cryptographic Building Blocks

3.1 Group Key Exchange

A group key exchange (KE) protocols allow users of a group communicating over an insecure public network to establish a common secret key (i.e., a group key). They are intended to be secure against the passive adversary only. A passive adversary is given access to the Execute (excluding the participance of the server S), Reveal, and Test oracles as defined in Section 2.

We say a passive adversary $\mathcal{A}$ succeeds if it makes a single query $\mathsf{Test}(A^i)$ to a fresh instance A^i (i.e., no Reveal oracle is queried to A^i and his partnered instances), 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 a passive adversary $\mathcal{A}$ in attacking a group KE protocol P is a function in the security parameter k, defined as $\mathsf{Adv}^P_{\mathcal{A}}(k) = 2 \cdot \mathsf{Pr}^P_{\mathcal{A}}[\mathsf{Succ}] - 1$.

A group KE protocol P is secure against passive eavesdropping if no polynomial bounded adversary $\mathcal{A}$ has a non-negligible advantage in attacking it.

The group KE protocols proposed by Ingemarsson et al. [39], Burmester and Desmedt [28], and Steiner et al. [52] may be the most well-known. Among them, Burmester-Desmedt protocol has been shown to be secure against passive eavesdropping in the standard model by Katz and Yung [44].

3.2 Identity-Based Encryption

An identity-based encryption (IBE) scheme is specified by four randomized algorithms: Setup, Extract, Encrypt, Decrypt as follows.

- Setup: On input a security parameter k, it returns params (public system parameters) and master-key (known only to the "Private Key Generator").
- Extract: On inputs params, master-key and a public identity $\mathsf{ID} \in \{0,1\}^*$, it returns a private key d.
- Encrypt: On inputs params, ID, and a message $M \in \mathcal{M}$ (the plaintext space), it returns a ciphertext $C \in \mathcal{C}$ (the ciphertext space).

– Decryption: On inputs params, $C \in \mathcal{C}$, and a private key d, it returns $M \in \mathcal{M}$.

Chosen ciphertext security is the standard acceptable notion of security for a public key encryption scheme. An IBE scheme is semantically secure against the adaptive chosen ciphertext attack with "multi-challenge" if no polynomial bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game:

– *Initialize*: The challenger runs the Setup algorithm, gives params to the adversary, but keeps the master-key to itself.
– *Phase* 1: The adversary adaptively asks a number of different queries $q_1, q_2, \cdots, q_m$, where q_i is either $\mathsf{Extract}(\mathsf{ID}_i)$ or $\mathsf{Decrypt}(\mathsf{ID}_i, C_i)$.
– *Challenge*: Once the adversary decides that Phase 1 is over, it outputs a sequence of equal length plaintext pairs $(M_\ell^{(0)}, M_\ell^{(1)})_{\ell=1,2,\cdots,\lambda} \in \mathcal{M}^2$ and an identity ID on which it wishes to be challenged, where ID must not appear in Phase 1. The challenger picks a random bit $b \in \{0,1\}$ and sends $C_\ell = \mathsf{Encrypt}(\mathsf{ID}, M_\ell^{(b)})$ $(\ell = 1, 2, \cdots, \lambda)$ as the challenge to the adversary.
– *Phase* 2: The adversary issues more queries $q_{m+1}, q_{m+2}, \cdots, q_n$ adaptively as in Phase 1, except that the adversary may not request a private key for ID or the decryption of (ID, C_ℓ).
– *Guess*: Finally, the adversary outputs a guess $b' \in \{0,1\}$ and wins the game if $b' = b$.

We define the adversary $\mathcal{A}$'s advantage in attacking the IBE scheme as a function of the security parameter k, $\mathsf{Adv}_{\mathcal{A}}^{E}(k) = |\mathsf{Pr}_{\mathcal{A}}^{E}[b' = b] - 1/2|$, where the probability is over the random bits used by the challenger and the adversary. The most efficient identity-based encryption schemes are currently based on bilinear pairings on elliptic curves, such as the Weil or Tate pairings. Boneh and Franklin [16, 17] were the first to give an IBE scheme from Weil pairing and prove it to be adaptive chosen-ciphertext security (with single challenge) in the random oracle model. More recently, several new IBE schemes from pairing (e.g., [54][36]) were proposed and proven to be adaptive chosen-ciphertext security (with single challenge) in the standard model. Following their proofs of security, one can prove their schemes actually to be secure against the adaptive chosen-ciphertext attack with multi-challenge.

3.3 Identity-Based Signature

An identity-based signature (IBS) scheme can be described by four algorithms Setup, Extract, Sign, Verify as follows.

– Setup: On input a security parameter k, it returns params (public system parameters) and master-key (known only to the "Private Key Generator").
– Extract: Given params, master-key and a public identity $\mathsf{ID} \in \{0,1\}^*$, it returns a private key d_{ID}.
– Sign: Given a message M, params, ID and a private key d_{ID}, it generates a signature σ of the user (with identity ID) on M.

– Verify: Given a signature σ, a message M, and $\mathsf{params}, \mathsf{ID}$, it outputs accept if σ is a valid signature of the user (with identity ID) on M, and outputs reject otherwise.

An IBS scheme is existential unforgeability under the chosen message attack [38] if no polynomial bounded adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the following game:

– *Initialize*: The challenger runs the Setup algorithm, gives params to the adversary, but keeps the master-key to itself.
– *Queries*: The adversary adaptively asks a number of different queries $q_1, q_2, \cdots, q_m$, where q_i is either $\mathsf{Extract}(\mathsf{ID}_i)$ or $\mathsf{Sign}(\mathsf{ID}_i, M)$.
– *Forgery*: Once the adversary decides that queries are over, it outputs a message M', an identity ID' and a string σ'. The adversary succeeds (denoted as Succ) if $\mathsf{Verify}(\mathsf{ID}', M', \sigma') = 1$, where ID' cannot appear in Extract queries and (ID', M') cannot appear in Sign queries.

We define the adversary $\mathcal{A}$'s advantage in attacking the IBS scheme as a function of the security parameter k, $\mathsf{Adv}^S_{\mathcal{A}}(k) = \mathsf{Pr}^S_{\mathcal{A}}[\mathsf{Succ}]$, where the probability is over the random bits used by the challenger and the adversary.

A generic approach to construct IBS schemes is to use an ordinary (i.e., non-identity-based) signature scheme and simply attach a certificate containing the public key of the signer to the signature [35]. An IBS scheme with provable security in the standard model was given by Paterson and Schuldt in [49].

4 An Efficient Compiler for Group PAKE

4.1 Description of the Compiler

In this section, we present an efficient compiler transforming any group KE protocol P to a group PAKE protocol P'. Following the communication model given in [43], we assume that every message is sent - via point-to-point links - to every user of the group taking part in the execution of the protocol, in other word, U^i sends each message to all users in pid^i_U. For simplicity, we refer to this as "broadcasting message", but stress that we do not assume a broadcast channel and, in particular, an active adversary can deliver different messages to different users of the group or refuse to deliver a message to some of participants.

Given a group KE protocol P, our compiler constructs a group PAKE protocol P' as shown in Fig. 1, in which n clients $A_1, A_2, \cdots, A_n$ (in lexicographic order) wish to establish a common secret key (i.e., a group key) with the help of a server S. A completely formal specification of the group PAKE protocol will appear in Section 5, where we give a proof of security for the protocol in the security model described in Section 2.

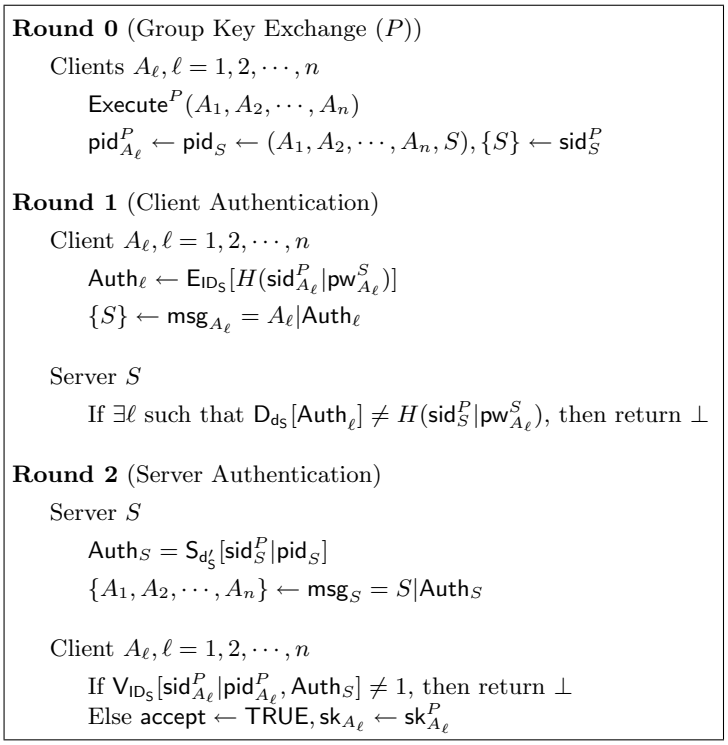

Fig. 1. ID-based group PAKE protocol P'

We present the protocol by describing initialization and execution. The cryptographic building blocks of our protocol include a group KE protocol, an IBE scheme and an IBS scheme. We let k be the security parameter given to the setup algorithm.

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

Parameter Generation: On input k, (1) PKG runs Setup^P of the group KE protocol P to generate system parameters, denoted as params^P; (2) PKG runs Setup^E of the IBE scheme to generate public system parameters for the IBE scheme, denoted as params^E, and the secret $\mathsf{master\text{-}key}^E$ for itself; (3) PKG runs Setup^S of the IBS scheme to generate public system parameters for the IBS scheme, denoted as params^S, and the secret $\mathsf{master\text{-}key}^S$ for itself; In addition, PKG chooses a hash function, $H : \{0,1\}^* \rightarrow \mathcal{M}$ (the plaintext space of IBE), from a collision-resistant family. The public system parameters for the protocol P' is $\mathsf{params} = \{H\} \bigcup \mathsf{params}^P \bigcup \mathsf{params}^E \bigcup \mathsf{params}^S$ and the secret ($\mathsf{master\text{-}key}^E$, $\mathsf{master\text{-}key}^S$) is known only to PKG.

Key Generation: On input the identity ID_S of a server $S \in \mathsf{Server}$, params, and ($\mathsf{master\text{-}key}^E$, $\mathsf{master\text{-}key}^S$), PKE runs $\mathsf{Extract}^E$ of the IBE scheme and sets the decryption key of S to be $\mathsf{d_S}$, and runs $\mathsf{Extract}^S$ of the IBS scheme and sets the signing key of S to be $\mathsf{d'_S}$.

Password Generation: On input $(A, S) \in \mathsf{ClientServerPair}$, a string pw_A^S, the password, is uniformly drawn by the client A 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. For a group of clients $A_1, A_2, \cdots, A_n$ (in lexicographic order), where there exists a server S such that $(A_\ell, S) \in \mathsf{ClientServerPair}$, when A_ℓ (having password $\mathsf{pw}_{A_\ell}^S$) $(\ell = 1, 2, \cdots, n)$ agree to establish a common secret key (i.e., a group key) via S, they firstly run the group KE protocol P to establish a session key. Note that, during the execution of P, the server S is provided the (ordered) concatenation of all "broadcast" messages, denoted as sid_S^P.

Next, each client A_ℓ computes an IBE encryption of $H(\mathsf{sid}_{A_\ell}^P | \mathsf{pw}_{A_\ell}^S)$ based on the identity ID_S of the server, denoted as $\mathsf{Auth}_\ell = \mathsf{E}_{\mathsf{ID}_S}[H(\mathsf{sid}_{A_\ell}^P | \mathsf{pw}_{A_\ell}^S)]$, where $\mathsf{sid}_{A_\ell}^P$ stands for the (ordered) concatenation of all "broadcast" messages that A_ℓ has viewed during the execution of P. Then $\mathsf{msg}_{A_\ell} = A_\ell | \mathsf{Auth}_\ell$ is provided to the server S.

Upon receiving the messages msg_{A_ℓ} $(\ell = 1, 2, \cdots, n)$, the server S decrypts the ciphertexts with its decryption key $\mathsf{d_S}$, and verifies whether

$$\mathsf{D}_{\mathsf{d_S}}[\mathsf{Auth}_\ell] = H(\mathsf{sid}_S^P | \mathsf{pw}_{A_\ell}^S) \tag{1}$$

where sid_S^P stands for the (ordered) concatenation of all "broadcast" messages that the server S has viewed during the execution of P.

If equation (1) holds for $\ell = 1, 2, \cdots, n$, the server S uses its signing key $\mathsf{d'_S}$ to generate a signature $\mathsf{Auth}_S = \mathsf{S}_{\mathsf{d'_S}}[\mathsf{sid}_S^P | \mathsf{pid}_S]$, and then broadcasts $\mathsf{msg}_S = S | \mathsf{Auth}_S$.

Upon receiving $\mathsf{msg_S}$, each client A_ℓ checks if

$$\mathsf{V}_{\mathsf{ID_S}}[\mathsf{sid}_{A_\ell}^P | \mathsf{pid}_{A_\ell}^P, \mathsf{Auth}_S] = 1 \tag{2}$$

If equation (2) holds, A_ℓ accepts the session key $\mathsf{sk}_{A_\ell}^P$ established during the execution of P as the authenticated group key sk_{A_ℓ}.

4.2 Correctness, Explicit Authentication and Efficiency

Correctness. In an honest execution of the protocol, clients $A_1, A_2, \cdots, A_n$ compute the identical group key because P itself is a group KE protocol.

Explicit Authentication. By verifying (1) involving the password $\mathsf{pw}_{A_\ell}^S$, the server S can make sure that each client A_ℓ is authenticated and the messages sent and received by A_ℓ are authenticated. By verifying (2) involving the signature of the server, each client A_ℓ is convinced of the authenticity of the server S and the received messages. Note that the server can always determine the password of each client (by offline dictionary attack) because the server has the decryption key $\mathsf{d_S}$. Therefore, it is not necessary for each client to verify whether the server knows his password or not. If both (1) and (2) hold, all clients are authenticated and all messages exchanged during the execution of P are authenticated. Thereby, the session key derived from the authenticated messages

is authenticated as well. This shows that the group PAKE protocol P' achieves explicit authentication, that is, a party knows that its intended partners have successfully computed a matching session key (i.e, a group key).

Efficiency Consideration. The efficiency of the group PAKE protocol depends on performance of the underlying group KE protocol, IBE and IBS schemes. Only two rounds are added to the original group KE protocol P. In these two rounds, each client sends out one message and receives one message only. In fact, each client A_ℓ can actually sends the message msg_{A_ℓ} to the server S in the unicast manner instead of "broadcast". This compiler adds only $O(1)$ communication (per client) to the original group KE protocol. In terms of the number of communication rounds, our compiler is better than Abdalla et al.'s compiler [1] if the underlying group KE protocol of our compiler is also Burmester-Desmedt protocol. Note that current 2-party PAKE protocols require at least three rounds to achieve explicit authentication. In this case, the group PAKE protocol constructed by Abdalla et al.'s compiler has at least 5 rounds, while our group PAKE protocol has 4 rounds only.

Our compiler can be made more efficient if each client A_ℓ includes $\mathsf{msg}_{A_\ell} = A_\ell|\mathsf{Auth}_\ell$ in his last "broadcast" message of the group KE protocol. In this case, our compiler adds only one more round to the original group KE protocol.

5 Proof of Security

We follow the methods of the security proofs given by Katz et al. in [44, 42] to prove the security of our compiler without random oracles.

First of all, we provide a formal specification of the group PAKE protocol by specifying the initialization phase and the oracles to which the adversary has access, as shown in Fig. 2−4.

During the initialization phase for security parameter k, algorithm $\mathsf{Initialize}$ generates $\mathsf{params} = \{H\} \bigcup \mathsf{params}^P \bigcup \mathsf{params}^E \bigcup \mathsf{params}^S$ and the secret ($\mathsf{master\text{-}key}^E, \mathsf{master\text{-}key}^S$) at first. Furthermore, the sets Client, Server, $\mathsf{ClientServerPair}$ are determined. Passwords for clients are chosen at random, and then stored at corresponding servers.

$\mathsf{Initialize}(1^k)$
$(\mathsf{params}^{P,E,S}, \mathsf{master\text{-}key}^{E,S}) \xleftarrow{R} \mathsf{Setup}^{P,E,S}(1^k)$
$(\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}) \xleftarrow{R} \mathsf{UserGen}(1^k), H \xleftarrow{R} \mathsf{CRHF}(1^k)$
For each $i \in \{1, 2, \cdots\}$ and each $U \in \mathsf{User}$
  $\mathsf{acc}_U^i \leftarrow \mathsf{term}_U^i \leftarrow \mathsf{used}_U^i \leftarrow \mathsf{FALSE}, \mathsf{sid}_U^i \leftarrow \mathsf{pid}_U^i \leftarrow \mathsf{sk}_U^i \leftarrow \mathsf{NULL}$
For each $S \in \mathsf{Server}$, $\mathsf{d}_\mathsf{S}, \mathsf{d}'_\mathsf{S} \leftarrow \mathsf{Extract}^{E,S}(\mathsf{ID}_S, \mathsf{params}^{E,S}, \mathsf{master\text{-}key}^{E,S})$
For each $(A, S) \in \mathsf{ClientServerPair}$, $\mathsf{pw}_A^S \xleftarrow{R} \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\} \subset \mathbb{Z}_q$
Return $\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}, H, \mathsf{params}^{P,E,S}$

Fig. 2. Specification of the initialize

$\mathsf{Execute}(A_1^{i_1}, \cdots, A_n^{i_n}, S^j)$, where $A_\ell \in \mathsf{Client}, S \in \mathsf{Server}$
If $(\exists \ell$ such that $(A_\ell, S) \notin \mathsf{ClientServerPair} \vee \mathsf{used}_{A_\ell}^{i_\ell}) \vee \mathsf{used}_S^j$, return $\perp$
$\mathsf{used}_{A_\ell}^{i_\ell} \leftarrow \mathsf{used}_S^j \leftarrow \mathsf{TRUE}, \mathsf{pid}_{A_\ell}^{i_\ell} \leftarrow \mathsf{pid}_S^j \leftarrow \{A_1, \cdots, A_n, S\}, \ell = 1, 2, \cdots, n$
$\mathsf{Execute}^P(A_1^{i_1}, A_2^{i_2}, \cdots, A_n^{i_n})$
$\mathsf{Auth}_\ell \leftarrow \mathsf{E}_{\mathsf{ID_S}}[H(\mathsf{sid}_{A_\ell}^{P,i_\ell} | \mathsf{pw}_{A_\ell}^S)], \mathsf{msg}_{A_\ell} \leftarrow A_\ell | \mathsf{Auth}_\ell, \ell = 1, 2, \cdots, n$
$\mathsf{Auth}_S \leftarrow \mathsf{S}_{\mathsf{d}'_\mathsf{S}}[\mathsf{sid}_S^{P,j} | \mathsf{pid}_S^j], \mathsf{msg}_S \leftarrow S | \mathsf{Auth}_S$
$\mathsf{sid}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sid}_{A_\ell}^{P,i_\ell} | \mathsf{msg}_{A_\ell} | \mathsf{msg}_S, \mathsf{sid}_S^j \leftarrow \mathsf{sid}_S^{P,j} | \mathsf{msg}_{A_1} | \mathsf{msg}_{A_2} | \cdots | \mathsf{msg}_{A_n} | \mathsf{msg}_S$
$\mathsf{acc}_{A_\ell}^{i_\ell} \leftarrow \mathsf{term}_{A_\ell}^{i_\ell} \leftarrow \mathsf{acc}_S^j \leftarrow \mathsf{term}_S^j \leftarrow \mathsf{TRUE}, \mathsf{sk}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sk}_{A_\ell}^{P,i_\ell}, \ell = 1, 2, \cdots, n$
Return $\mathsf{status}_{A_1}^{i_1}, \cdots, \mathsf{status}_{A_n}^{i_n}, \mathsf{status}_S^j$

$\mathsf{KeyGen}(\mathsf{PKG}, S)$
Return $\mathsf{d_S}, \mathsf{d}'_\mathsf{S}$ and pw_A^S for any A

$\mathsf{Corrupt}(A)$
Return pw_A^S for any S

$\mathsf{Reveal}(A^i)$
Return sk_A^i

$\mathsf{Test}(A^i)$
$b \xleftarrow{R} \{0, 1\}, \mathsf{sk}' \xleftarrow{R} \mathbb{Z}_q^*$. If $b = 1$ return sk' else return sk_A^i

Fig. 3. Specification of the Execute, KeyGen, Corrupt, Reveal, Test oracles

$\mathsf{Send}_0(A_\ell^{i_\ell}, (A_1^{i_1}, \cdots, A_n^{i_n}, S^j))$
If $(A_\ell, S) \notin \mathsf{ClientServerPair} \vee \mathsf{used}_{A_\ell}^{i_\ell}$, return $\perp$
$\mathsf{used}_{A_\ell}^{i_\ell} \leftarrow \mathsf{TRUE}, \mathsf{pid}_{A_\ell}^{i_\ell} \leftarrow \{A_1, \cdots, A_n, S\}$
$\cdots\cdots\cdots$

$\mathsf{Send}'_0(A_\ell^{i_\ell}, S^j)$
If $\neg \mathsf{used}_{A_\ell}^{i_\ell} \vee \mathsf{term}_{A_\ell}^{i_\ell} \vee (S \notin \mathsf{pid}_{A_\ell}^{i_\ell})$, return $\perp$
$\mathsf{Auth}_\ell \leftarrow \mathsf{E}_{\mathsf{ID_S}}[H(\mathsf{sid}_{A_\ell}^{P,i_\ell} | \mathsf{pw}_{A_\ell}^S)]$
$\mathsf{MsgOut} \leftarrow A_\ell | \mathsf{Auth}_\ell, \mathsf{state}_{A_\ell}^i \leftarrow (\mathsf{sid}_{A_\ell}^{P,i_\ell}, \mathsf{sk}_{A_\ell}^{P,i_\ell}, \mathsf{MsgOut})$
Return $\mathsf{status}_{A_\ell}^{i_\ell}$

$\mathsf{Send}'_1(S^j, (A_\ell | \mathsf{Auth}_\ell)_{\ell=1,2,\cdots,n})$
If $(\exists \ell$ such that $(A_\ell, S) \notin \mathsf{ClientServerPair}) \vee \mathsf{used}_S^j$, return $\perp$
$\mathsf{used}_S^j \leftarrow \mathsf{TRUE}, \mathsf{pid}_S^j \leftarrow \{A_1, A_2, \cdots, A_n, S\}$
If $\exists \ell$ such that $\mathsf{D}_{\mathsf{d_S}}(\mathsf{Auth}_\ell) \neq H(\mathsf{sid}_S^{P,j} | \mathsf{pw}_{A_\ell}^S)$, return status_S^j
$\mathsf{Auth}_S \leftarrow \mathsf{S}_{\mathsf{d}'_\mathsf{S}}[\mathsf{sid}_S^{P,j} | \mathsf{pid}_S^j], \mathsf{acc}_S^j \leftarrow \mathsf{term}_S^j \leftarrow \mathsf{TRUE}$
$\mathsf{MsgOut} \leftarrow S | \mathsf{Auth}_S, \mathsf{sid}_S^j \leftarrow \mathsf{sid}_S^{P,j} | \mathsf{MsgIn} | \mathsf{MsgOut}$
Return status_S^j

$\mathsf{Send}'_2(A_\ell^{i_\ell}, S | \mathsf{Auth}_S)$
If $\neg \mathsf{used}_{A_\ell}^{i_\ell} \vee \mathsf{term}_{A_\ell}^{i_\ell} \vee (S \notin \mathsf{pid}_{A_\ell}^{i_\ell})$, return $\perp$
$\mathsf{state}_{A_\ell}^{i_\ell} \leftarrow (\mathsf{sid}_{A_\ell}^{P,i_\ell}, \mathsf{sk}_{A_\ell}^{P,i_\ell}, \mathsf{FirstMsgOut})$
If $\mathsf{V}_{\mathsf{ID_S}}[\mathsf{sid}_{A_\ell}^{P,i_\ell} | \mathsf{pid}_{A_\ell}^{i_\ell}, \mathsf{Auth}_S] \neq 1$, return $\mathsf{status}_{A_\ell}^{i_\ell}$
$\mathsf{sid}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sid}_{A_\ell}^{P,i_\ell} | \mathsf{FirstMsgOut} | \mathsf{MsgIn}, \mathsf{acc}_{A_\ell}^{i_\ell} \leftarrow \mathsf{term}_{A_\ell}^{i_\ell} \leftarrow \mathsf{TRUE}, \mathsf{sk}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sk}_{A_\ell}^{P,i_\ell}$
Return $\mathsf{status}_{A_\ell}^{i_\ell}$

Fig. 4. Specification of the Send oracles

The description of the Execute oracle matches the high-level protocol described in Fig. 1, but additional details (for example, the updating of state infor-

mation) are included. We let status_U^i denote the vector of values ($\mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{acc}_U^i$, term_U^i) associated with instance U^i.

Given an adversary $\mathcal{A}$, we imagine a simulator that runs the protocol for $\mathcal{A}$. More preciously, the simulator begins by running algorithm $\mathsf{Initialize}(1^k)$ (which includes choosing passwords for clients) and giving the public output of the algorithm to $\mathcal{A}$. When $\mathcal{A}$ queries an oracle, the simulator also responds by executing the appropriate algorithm. The simulator also records all state information defined during the course of the experiment. 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 client 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 $\mathsf{Adv}_{\mathcal{A}}^{P'}(k) = 2 \cdot \mathsf{Pr}_{\mathcal{A}}^{P'}[\mathsf{Succ}] - 1$.

Based on the model described in Section 2, we have

Theorem 1. Assume that (1) the group KE protocol is secure against passive eavesdropping; (2) the IBE scheme is secure against the chosen-ciphertext attack with multi-challenge; (3) the IBS scheme is existential unforgeability under the chosen-message attack; (4) CRHF is a collision-resistant hash family; then the protocol P' described in Fig. 1 is a secure group PAKE protocol.

We begin with some terminology that will be used 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 will 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 before except that the adversary does not succeed, and the experiment is aborted, if any of the following occur:

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

It is immediate that events 1 occurs with only negligible probability, event 2 occurs with negligible probability assuming the security of CRHF as a collision-resistant hash family. Put everything together, we are able to see that

Claim 1. $|\mathsf{Adv}_{\mathcal{A}}^{P'_0}(k) - \mathsf{Adv}_{\mathcal{A}}^{P'_1}(k)|$ is negligible.

Experiment P_2': In this experiment, we modify the simulator's responses to Send_1' and Send_2' queries.

Before describing this change we introduce some terminology. The simulator first runs the protocol initialization as shown in Fig. 2. For a query $\mathsf{Send}_1'(S^j, (\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n}))$, where $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$ is adversarially-generated, if equation (1) holds for $\ell = 1, 2, \cdots, n$, then $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$ is said to be valid. Otherwise, $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$ is said to be invalid. Similarly, for a query $\mathsf{Send}_2'(A_\ell^{i_\ell}, \mathsf{msg}_S)$ where msg_S is adversarially-generated, if equation (2) holds, then msg_S is said to be valid. Otherwise, msg_S is said to be invalid. Informally, valid messages use correct passwords or private keys while invalid messages do not.

Given this terminology, we continue with our description of experiment P_2'. When the adversary makes oracle queries $\mathsf{Send}_1'(S^j, (\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n}))$, the simulator examines $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$. If it is adversarially-generated and valid, the query is answered as in experiment P_1' except that acc_S^j is assigned the special value ∇. In any other case, (i.e., $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$ is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment P_1'. When the adversary makes oracle queries $\mathsf{Send}_2'(A_\ell^{i_\ell}, \mathsf{msg}_S)$, the simulator examines msg_S. If msg_S is adversarially-generated and valid, the query is answered as in experiment P_1' except that $\mathsf{acc}_{A_\ell}^{i_\ell}$ is assigned the special value ∇. In any other case, (i.e., msg_S is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment P_1'.

Finally, the definition of the adversary's success in P_2' is changed. If the adversary ever queries Send_1' or Send_2' with $\mathsf{acc}_S^j = \nabla$ or $\mathsf{acc}_{A_\ell}^{i_\ell} = \nabla$, the simulator halts and the adversary succeeds. Otherwise the adversary's success is determined as in experiment P_1'.

The distribution on the adversary's view in experiments P_1' and P_2' are identical up to the point when the adversary queries Send_1' or Send_2' with $\mathsf{acc}_S^j = \nabla$ or $\mathsf{acc}_{A_\ell}^{i_\ell} = \nabla$. If such a query is never made, the distributions on the view are identical. Therefore, we have

Claim 2. $\mathsf{Adv}_{\mathcal{A}}^{P_1'}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_2'}(k)$.

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 $\mathsf{Execute}$ and Send_0' oracle are handled differently.

For each $\mathsf{Execute}$ and each Send_0' query, each value Auth_ℓ is computed as $\mathsf{Auth}_\ell = \mathsf{E}_{\mathsf{ID}_S}[H(\mathsf{sid}_{A_\ell}^{P,i_\ell} | \mathsf{pw'}_{A_\ell}^S)]$ where $\mathsf{pw'}_{A_\ell}^S$ is randomly chosen from $\mathsf{Z}_q^* - \mathsf{Password}$ (i.e., it is not a valid password). The following bounds the effect this transformation can have on the adversary's advantage.

Claim 3. If the IBE scheme is secure against the chosen-ciphertext attack with multi-challenge, $|\mathsf{Adv}_{\mathcal{A}}^{P_2'}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_3'}(k)|$ is negligible.

Assume that there exist m servers $S_1, S_2, \cdots, S_m$. The claim is proved by m sub-experiments $P_3'^{(1)}, P_3'^{(2)}, \cdots, P_3'^{(m)} = P_3'$. In experiment $P_3'^{(t)}$, only $\mathsf{Execute}$

and Send'_0 related to the server S_t is handled differently. Let $P_3'^{(0)} = P'_2$, we only need to prove $|\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t-1)}}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t)}}(k)|$ (where $1 \le t \le m$) to be negligible.

The proof relies on the chosen-ciphertext security with multi-challenge of the IBE scheme. If $|\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t-1)}}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t)}}(k)|$ is non-negligible, we show that the simulator can use $\mathcal{A}$ as a subroutine to perform the chosen-ciphertext attack to the IBE as follows.

Let $S = S_t$, the simulator is given public parameters params^E for an instance of the IBE scheme, and may repeatedly query the encryption oracle $\mathsf{E}_{\mathsf{ID}_S,\tilde{b}}[\cdot,\cdot]$ (where $\tilde{b} \in \{0,1\}$), which is defined as follows.

$$\mathsf{E}_{\mathsf{ID}_S,\tilde{b}}[M_0, M_1] = \mathsf{E}_{\mathsf{ID}_S}(M_{\tilde{b}})$$

where M_0, M_1 are any two messages with equal length, and $\tilde{b}$ is randomly-chosen bits (unknown to simulator).

The simulator begins by running a modified initialization protocol as follows.

$\mathsf{Initialize}'(1^k, \mathsf{params}^E)$–

$(\mathsf{params}^{P,S}, \mathsf{master\text{-}key}^S) \xleftarrow{R} \mathsf{Setup}^{P,S}(1^k), H \xleftarrow{R} \mathsf{CRHF}(1^k)$

$(\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}) \xleftarrow{R} \mathsf{UserGen}(1^k)$,

For each $S \in \mathsf{Server}$, $\mathsf{d}'_S \leftarrow \mathsf{Extract}^S(\mathsf{ID}_S, \mathsf{params}^S, \mathsf{master\text{-}key}^S)$

For each $(A, S) \in \mathsf{ClientServerPair}$, $\mathsf{pw}_A^S \xleftarrow{R} \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$

Return $\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}, H, \mathsf{params}^{P,E,S}$

The simulator responds to $\mathsf{Corrupt}$, Reveal, Test queries, Send queries in the protocol P, Send'_2 queries as in the experiment $P_3'^{(t-1)}$. It responds to $\mathsf{Execute}$ and Send'_0 queries unrelated to the server S_t as in the experiment $P_3'^{(t-1)}$, too. However, it responds to KeyGen oracle and the decryption oracle $\mathsf{D}_{\mathsf{d}_S}$ in the Send'_1 by querying the challenger of the IBE scheme. With $\mathsf{D}_{\mathsf{d}_S}$, the simulator is able to tell if an adversarially-generated message $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$ is valid or not. Furthermore, it is able to generate the signature Auth_S since it knows the signing private key d'_S of each server. Thus, the simulator can answer various Send'_1 queries.

The simulator responds to $\mathsf{Execute}$ and Send'_0 queries related to S_t as shown in Fig. 5 and Fig. 6, respectively. Each time the simulator responds to $\mathsf{Execute}$ (or Send'_0) query related to S_t, it constructs the values Auth_ℓ using the encryption oracle $\mathsf{E}_{\mathsf{ID}_S,\tilde{b}}[\cdot,\cdot]$ of the IBE scheme by querying it with $H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw}_{A_\ell}^S)$ and $H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw}'^S_{A_\ell})$ as its two "messages".

When $\tilde{b} = 0$, the actions of the $\mathsf{Execute}$ and Send'_0 oracles are exactly as in experiment $P_3'^{(t-1)}$, while if $\tilde{b} = 1$ the actions of the $\mathsf{Execute}$ and Send'_0 oracles are exactly as in experiment $P_3'^{(t)}$. In particular, for an $\mathsf{Execute}$ (or Send'_0) query related to the server S_t, when $\tilde{b} = 0$ then $\mathsf{Auth}_\ell = H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw}_{A_\ell}^S)$, while if $\tilde{b} = 1$ then $\mathsf{Auth}_\ell = H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw}'^S_{A_\ell})$. If the adversary $\mathcal{A}$ has queried $\mathsf{KeyGen}(\mathsf{PKG}, S_t)$, the simulator aborts. In a Test query, the simulator lets $\tilde{b} = 0$ if the adversary $\mathcal{A}$ succeeds (i.e., $b' = b$). Otherwise, let $\tilde{b}$ be a random choice of $\{0,1\}$.

$\mathsf{Execute}(A_1^{i_1}, \cdots, A_n^{i_n}, S^j)$, where $S = S_t$

If $(\exists \ell$ such that $(A_\ell, S) \notin \mathsf{ClientServerPair} \vee \mathsf{used}_{A_\ell}^{i_\ell}) \vee \mathsf{used}_S^k$, return $\perp$

$\mathsf{used}_{A_\ell}^{i_\ell} \leftarrow \mathsf{used}_S^j \leftarrow \mathsf{TRUE}, \mathsf{pid}_{A_\ell}^{i_\ell} \leftarrow \mathsf{pid}_S^k \leftarrow \{A_1, \cdots, A_n, S\}, \ell = 1, 2, \cdots, n$

$\mathsf{Execute}^P(A_1^{i_1}, A_2^{i_2}, \cdots, A_n^{i_n})$

$\mathsf{Auth}_\ell \leftarrow \mathsf{E}_{\mathsf{ID_S},\tilde{\mathsf{b}}}[H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw}_{A_\ell}^S), H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw'}_{A_\ell}^S)], \mathsf{msg}_{A_\ell} \leftarrow A_\ell|\mathsf{Auth}_\ell$ for any ℓ

$\mathsf{Auth}_S \leftarrow \mathsf{S}_{\mathsf{d'_S}}[\mathsf{sid}_S^{P,j}|\mathsf{pid}_S^j], \mathsf{msg}_S \leftarrow S|\mathsf{Auth}_S$

$\mathsf{sid}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{msg}_{A_\ell}|\mathsf{msg}_S, \mathsf{sid}_S^j \leftarrow \mathsf{sid}_S^{P,j}|\mathsf{msg}_{A_1}|\mathsf{msg}_{A_2}|\cdots|\mathsf{msg}_{A_n}|\mathsf{msg}_S$
$\mathsf{acc}_{A_\ell}^{i_\ell} \leftarrow \mathsf{term}_{A_\ell}^{i_\ell} \leftarrow \mathsf{acc}_S^j \leftarrow \mathsf{term}_S^j \leftarrow \mathsf{TRUE}, \mathsf{sk}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sk}_{A_\ell}^{P,i_\ell}, \ell = 1, 2, \cdots, n$
Return $\mathsf{status}_{A_1}^{i_1}, \cdots, \mathsf{status}_{A_n}^{i_n}, \mathsf{status}_S^j$

Fig. 5. The modified Execute oracle for proof of Claim 3

$\mathsf{Send}'_0(A_\ell^{i_\ell}, S^j)$ where $S = S_t$

If $\neg\mathsf{used}_{A_\ell}^{i_\ell} \vee \mathsf{term}_{A_\ell}^{i_\ell} \vee (S \notin \mathsf{pid}_{A_\ell}^{i_\ell})$, return $\perp$

$\mathsf{Auth}_\ell \leftarrow \mathsf{E}_{\mathsf{ID},\tilde{b}}[H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw}_{A_\ell}^S), H(\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pw'}_{A_\ell}^S)]$

$\mathsf{MsgOut} \leftarrow A_\ell|\mathsf{Auth}_\ell, \mathsf{state}_{A_\ell}^i \leftarrow (\mathsf{sid}_{A_\ell}^{P,i_\ell}, \mathsf{sk}_{A_\ell}^{P,i_\ell}, \mathsf{MsgOut})$

Return $\mathsf{status}_{A_\ell}^{i_\ell}$

$\mathsf{Send}'_1(S^j, (A_\ell|\mathsf{Auth}_\ell)_{\ell=1,2,\cdots,n})$ where $S = S_t$

If $(\exists \ell$ such that $(A_\ell, S) \notin \mathsf{ClientServerPair}) \vee \mathsf{used}_S^j$, return $\perp$

$\mathsf{used}_S^j \leftarrow \mathsf{TRUE}, \mathsf{pid}_S^j \leftarrow \{A_1, A_2, \cdots, A_n, S\}$

If MsgIn is adversarially-generated

If $\mathsf{D}_{\mathsf{d_S}}(\mathsf{Auth}_\ell) = H(\mathsf{sid}_S^{P,j}|\mathsf{pw}_{A_\ell}^S)$ for $\ell = 1, 2, \cdots, n$, then $\mathsf{acc}_S^j \leftarrow \nabla$

Else $\mathsf{acc}_S^j \leftarrow \mathsf{term}_S^j \leftarrow \mathsf{TRUE}, \mathsf{Auth}_S \leftarrow \mathsf{S}_{\mathsf{d'_S}}[\mathsf{sid}_S^{P,j}|\mathsf{pid}_S^j)]$

$\mathsf{MsgOut} \leftarrow S|\mathsf{Auth}_S, \mathsf{sid}_S^j \leftarrow \mathsf{sid}_S^{P,j}|\mathsf{MsgIn}|\mathsf{MsgOut}$

Return status_S^j

$\mathsf{Send}'_2(A_\ell^{i_\ell}, S|\mathsf{Auth}_S)$

If $\neg\mathsf{used}_{A_\ell}^{i_\ell} \vee \mathsf{term}_{A_\ell}^{i_\ell} \vee (S \notin \mathsf{pid}_{A_\ell}^{i_\ell})$, return $\perp$

$\mathsf{state}_{A_\ell}^{i_\ell} \leftarrow (\mathsf{sid}_{A_\ell}^{P,i_\ell}, \mathsf{sk}_{A_\ell}^{P,i_\ell}, \mathsf{FirstMsgOut})$

If MsgIn is adversarially-generated

If $\mathsf{V}_{\mathsf{ID_S}}[\mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{pid}_{A_\ell}^{i_\ell}, \mathsf{Auth}_S] = 1$, then $\mathsf{acc}_{A_\ell}^{i_\ell} \leftarrow \nabla$

Else $\mathsf{sid}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sid}_{A_\ell}^{P,i_\ell}|\mathsf{FirstMsgOut}|\mathsf{MsgIn}, \mathsf{acc}_{A_\ell}^{i_\ell} \leftarrow \mathsf{term}_{A_\ell}^{i_\ell} \leftarrow \mathsf{TRUE}, \mathsf{sk}_{A_\ell}^{i_\ell} \leftarrow \mathsf{sk}_{A_\ell}^{P,i_\ell}$

Return $\mathsf{status}_{A_\ell}^{i_\ell}$

Fig. 6. The modified Send oracles for the proof of Claim 3

Assume the probability of the adversary to succeeds in $P_3^{\prime(t-1)}$ and $P_3^{\prime(t)}$ are $p_3^{\prime(t-1)}$ and $p_3^{\prime(t)}$, respectively, and the probability of the simulator aborting is p_a,

then the advantage of the simulator to succeed in guessing $\tilde{b}$ is

$$\begin{aligned} & |Pr(\tilde{b}=0|b'=b)-\frac{1}{2}|\cdot(1-p_a) \\ \geq & |\frac{p_3'^{(t-1)}}{p_3'^{(t-1)}+p_3'^{(t)}}-\frac{1}{2}|\cdot\frac{1}{m} \\ \geq & \frac{|p_3'^{(t-1)}-p_3'^{(t)}|}{4m}=\frac{|\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t-1)}}(k)-\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t)}}(k)|}{8m} \end{aligned}$$

Note that the adversary A can ask at most $m-1$ KeyGen queries. Otherwise, there is no fresh instance for the Test query.

Because m is the number of servers (which is constant), if $|\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t-1)}}(k)-\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t)}}(k)|$ is non-negligible, the simulator can use the adversary $\mathcal{A}$ to perform the chosen-ciphertext attack with multi-challenge to the IBE with a non-negligible advantage. However, the IBE is assumed to be secure against the chosen-ciphertext attack with multi-challenge. So $|\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t-1)}}(k)-\mathsf{Adv}_{\mathcal{A}}^{P_3'^{(t)}}(k)|$ must be negligible and Claim 3 is true.

Next, we consider separately the case when the adversary $\mathcal{A}$ asks its Test query to an instance initialized via an $\mathsf{Execute}$ query, and the case when $\mathcal{A}$ asks its Test query to an instance initialized via a Send query. More formally, let Ex be the event that $\mathcal{A}$ makes its query $\mathsf{Test}(A_\ell^{i_\ell})$ to an instance $A_\ell^{i_\ell}$ such that $\mathcal{A}$ never made a query of the form $\mathsf{Send}(U^i,*)$ where $U\in\mathsf{pid}_{A_\ell}^{i_\ell}$ (therefore did make an $\mathsf{Execute}$ query involving this instance).

Let $\mathsf{Se}=\overline{\mathsf{Ex}}$, then

$$\mathsf{Pr}_{\mathcal{A}}^{P_3'}[\mathsf{Succ}]\leq\mathsf{Pr}_{\mathcal{A}}^{P_3'}[\mathsf{Succ}\wedge\mathsf{Se}]+\mathsf{Pr}_{\mathcal{A}}^{P_3'}[\mathsf{Succ}|\mathsf{Ex}]\cdot(1-\mathsf{Pr}_{\mathcal{A}}^{P_3'}[\mathsf{Succ}\wedge\mathsf{Se}])$$

To evaluate $\mathsf{Pr}_{\mathcal{A}}^{P_3'}[\mathsf{Succ}|\mathsf{Ex}]$ and $\mathsf{Pr}_{\mathcal{A}}^{P_3'}[\mathsf{Succ}\wedge\mathsf{Se}]$, we do the following experiment.

Experiment P_4'. In this experiment, the simulator responds to all oracle queries as in experiment P_3' except that it begins by running the initialization specified in Fig. 2.

It is obvious that the distribution of the adversary's view on experiments P_3' and P_4' are identical.

Claim 4. If the group KE protocol P is secure against the passive eavesdropping and $\mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Ex}]>0$, then $\mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ}|\mathsf{Ex}]=1/2+\varepsilon'$ where ε' is negligible.

If ε' is non-negligible, the simulator can use $\mathcal{A}$ to construct a passive adversary $\mathcal{A}'$ attacking the protocol P as follows.

For any $\mathsf{Execute}$ query asked by the adversary $\mathcal{A}$, the simulator has the adversary $\mathcal{A}'$ to query $\mathsf{Execute}^P$ in the protocol P at first and then outputs a transcript as in experiment P_3' except from no session key assigned. In case the adversary $\mathcal{A}$ queries Reveal oracle, the simulator has $\mathcal{A}'$ to query Reveal^P in the protocol P and forwards the result to $\mathcal{A}$. Since the $\mathsf{master\text{-}key}^{E,S}$ and passwords

are generated by the simulator itself in the initialization, the simulator is able to answer all other oracle queries, such as Send queries, as in experiment P_3'.

When the adversary $\mathcal{A}$ queries $\mathsf{Test}(A_\ell^{i_\ell})$ oracle, the simulator checks if $\mathcal{A}$ has ever made any query of form $\mathsf{Send}(U^i, *)$ where $U \in \mathsf{pid}_{A_\ell}^{i_\ell}$ (i.e., whether event Se has occurred). If so, the simulation aborts and outputs a random bit. Otherwise (namely, if event Ex has occurred), the simulator has $\mathcal{A}'$ to query $\mathsf{Test}^P(A_\ell^{i_\ell})$ in the protocol P, forward the question to $\mathcal{A}$, and output whatever $\mathcal{A}$ outputs. The simulation is perfect for $\mathcal{A}'$ unless the simulator aborts due to the occurrence of event Se. Therefore,

$$\mathsf{Pr}_{\mathcal{A}'}^{P}[\mathsf{Succ}] = \mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ}|\mathsf{Ex}] \cdot \mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Ex}] + \frac{1}{2} \cdot \mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Se}]$$

Because $\mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ}|\mathsf{Ex}] = 1/2 + \varepsilon'$, we have $\mathsf{Pr}_{\mathcal{A}'}^{P}[\mathsf{Succ}] - 1/2 = \varepsilon' \cdot \mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Ex}]$. If ε' is non-negligible, $\mathsf{Adv}_{\mathcal{A}'}^{P}(k) = |2 \cdot \mathsf{Pr}_{\mathcal{A}'}^{P}[\mathsf{Succ}] - 1|$ is non-negligible because $\mathsf{Pr}[\mathsf{Ex}]$ is constant. This is in contradiction with the assumption that the group KE protocol P is secure against the passive eavesdropping. Therefore, ε' must be negligible. Claim 4 follows.

In case that the event Se has occurred, the adversary $\mathcal{A}$ succeeds if one of the following occurs:

- The adversary queries $\mathsf{Send}_1'(S^j, (\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n}))$ for adversarially-generated and valid $(\mathsf{msg}_{A_1}, \cdots, \mathsf{msg}_{A_n})$, that is, $\mathsf{acc}_S^j = \nabla$.
- The adversary queries $\mathsf{Send}_2'(A_\ell^{i_\ell}, \mathsf{msg}_S)$ for adversarially-generated and valid msg_S, that is, $\mathsf{acc}_{A_\ell}^{i_\ell} = \nabla$.

Let Succ_1, and Succ_2 denote the above two events, respectively, then

$$\mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ} \wedge \mathsf{Se}] \leq \mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ}_1] + \mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ}_2]$$

Because Auth_ℓ and $\mathsf{sk}_{A_\ell}^{i_\ell}$ for any ℓ and Auth_S for any S are all independent of passwords chosen by the simulator in this experiment, the adversary's view is independent of passwords until Succ_1 occurs. The probability that Succ_1 occurs is at most $Q(k)/N$, where $Q(k)$ is the number of online attacks made by the adversary $\mathcal{A}$.

To evaluate $\mathsf{Pr}_{\mathcal{A}}^{P_4'}[\mathsf{Succ}_2]$, we do the following experiment.

Experiment P_5'. The simulator responds to all oracle queries as in experiment P_4' except that it begins by running a modified initialization as follows.

$\mathsf{Initialize}''(1^k, \mathsf{params}^S)$–

$(\mathsf{params}^{P,E}, \mathsf{master\text{-}key}^E) \xleftarrow{R} \mathsf{Setup}^{P,E}(1^k), H \xleftarrow{R} \mathsf{CRHF}(1^k)$

$(\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}) \xleftarrow{R} \mathsf{UserGen}(1^k)$

For each $S \in \mathsf{Server}$, $\mathsf{d_S} \leftarrow \mathsf{Extract}^E(\mathsf{ID}_S, \mathsf{params}^E, \mathsf{master\text{-}key}^E)$

For each $(A, S) \in \mathsf{ClientServerPair}$, $\mathsf{pw}_A^S \xleftarrow{R} \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\} \subset \mathbb{Z}_q$

Return $\mathsf{Client}, \mathsf{Server}, \mathsf{ClientServerPair}, H, \mathsf{params}^{P,E,S}$

It is obvious that the distribution of the adversary's view on experiments P_4' and P_5' are identical.

Claim 5. If the IBS has existential unforgeability under chosen-message attack, then $\varepsilon'' = \mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ}_2]$ is negligible.

If ε'' is non-negligible, the simulator can use the adversary $\mathcal{A}$ to construct a forger $\mathcal{A}'$ attacking the IBS scheme as follows.

For KeyGen queries asked by the adversary $\mathcal{A}$, the simulator has the adversary $\mathcal{A}'$ to query the challenger of the IBS scheme. Each time the simulator responds to Send_1' query asked by $\mathcal{A}$, it checks if the messages are valid or not (note that it knows the decryption key $\mathsf{d_S}$). If so, the simulator has $\mathcal{A}'$ to query the signing oracle $\mathsf{S}_{\mathsf{d}_S'}$ of the IBS scheme on the message $H(\mathsf{sid}_S^{P,j}|\mathsf{pid}_S^j)$ and returns the signature Auth_S to $\mathcal{A}$. In addition, the simulator responds to all other oracles as in experiment P_4'.

In case that the adversary $\mathcal{A}$ queried $\mathsf{Send}_2'(A_\ell^{i_\ell}, \mathsf{msg}_S)$ for adversarially-generated and valid msg_S (that is, $\mathsf{acc}_{A_\ell}^{i_\ell} = \nabla$), the adversary $\mathcal{A}'$ uses it to forge a signature of the signer S on the message msg_S. It is in contradiction with the assumption that the IBS scheme has existential unforgeability. Therefore, $\varepsilon'' = \mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ}_2]$ must be negligible and the claim follows.

The preceding discussion implies that

$$\begin{aligned}\mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ}] &\leq \mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ} \wedge \mathsf{Se}] + \mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ}|\mathsf{Ex}] \cdot (1 - \mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ} \wedge \mathsf{Se}]) \\ &\leq (\frac{Q(k)}{N} + \varepsilon'') + (\frac{1}{2} + \varepsilon')(1 - \frac{Q(k)}{N} - \varepsilon'') \\ &\leq \frac{1}{2} + \frac{Q(k)}{2N} + \frac{\varepsilon''}{2} + (1 - \frac{Q(k)}{N} - \varepsilon'')\varepsilon'\end{aligned}$$

and thus

$$2 \cdot \mathsf{Pr}_{\mathcal{A}}^{P_5'}[\mathsf{Succ}] - 1 \leq \frac{Q(k)}{N} + \varepsilon'' + 2 \cdot (1 - \frac{Q(k)}{N} - \varepsilon'')\varepsilon'$$

This means that the adversary's advantage $\mathsf{Adv}_{\mathcal{A}}^{P_5'}(k)$ in experiment P_5' is at most $Q(k)/N$ plus negligible quantity.

The sequence of claims proved above show that

$$\mathsf{Adv}_{\mathcal{A}}^{P_0'}[\mathsf{Succ}] \leq \mathsf{Adv}_{\mathcal{A}}^{P_5'}(k) + \varepsilon''' \leq \frac{Q(k)}{N} + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$ and therefore the adversary's advantage in P_0' is at most $Q(k)/N$ plus some negligible quantity. This complete the proof of the theorem.

6 Conclusion

In this paper, we present the first compiler to transform any group KE protocol to a group PAKE protocol from identity-based cryptosystem. In addition, we

provide a rigorous proof of security for our compiler. As long as our group PAKE protocol is built on a group KE protocol, and IBE and IBS schemes with provable security without random oracles, it can be proven to be secure without random oracles. One could prove that our protocol achieves key privacy with respect to the server, as in [3].

References

1. M. Abdalla, J. M. Bohli, M. I. G. Vasco, R. Steinwandt. (Password) authenticated key establishment: From 2-party to group. In *Proc. TCC'07*, pages 499-514, 2007.
2. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-based group key exchange in a constant number of rounds. In *Proc. PKC'06*, pages 427-442, 2006.
3. M. Abdalla, P. A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *Proc. PKC'05*, pages 65-84, Jan. 2005.
4. M. Abdalla, P. A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. *IEE Proceedings in Information Security*, 153(1): 27-39, Mar. 2006.
5. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Proc. CT-RSA 2005*, pages 191 - 208, Feb. 2005.
6. M. Abdalla and D. Pointcheval. A scalable password-based group key exchange in the standard model. In *Proc. Asiacrypt'06*, pages 332-347, 2006.
7. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *Proc. CCS'98*, pages 17-26, 1998.
8. G. Ateniese, M. Steiner, and G. Tsudik. New multi-party authentication services and key agreement protocol. *IEEE Journal on Selected Areas in Communications*, (18)4: 628-639, 2000.
9. C. Becker and U. Wille. Communication complexity of group key distribution. In *Proc. CCS'98*, pages 1-6, 1998.
10. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocol. In *Proc. 30th Annual ACM Symposium on Theory of Computing*, pages 419-428, 1998.
11. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proc. Eurocrypt'00*, pages 139-155, May 2000.
12. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proc. Crypto'93*, pages 232-249, 1993.
13. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocol secure against dictionary attack. In *Proc. 1992 IEEE Symposium on Research in Security and Privacy*, pages 72-84, May 1992.
14. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5): 679-693, 1993.
15. J. M. Bohli, M. I. G. Vasco, and R. Steinwandt. Password-authenticated constant-round group key establishment with a common reference string. Cryptology ePrint Archive, Report 2006/214, 2006. http://eprint.iacr.org/.
16. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Proc. Crypto'01*, pages 213-229, 2001.
17. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586-615, 2003.

18. C. Boyd. On key agreement and conference key agreement. In *Proc. ACISP'97*, pages 294-302, 1997.
19. C. Boyd and J. M. G. Nieto. Round-optimal contributory conference key agreement. In *Proc. PKC'03*, pages 161-174, 2003.
20. V. Boyko, P. Mackenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Proc. Eurocrypt'00*, pages 156-171, May 2000.
21. E. Bresson, O. Chevassut, and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange - the dynamic case. In *Proc. Asiacrypt'01*, pages 290-309, 2001.
22. E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proc. CCS'01*, pages 255-264, 2001.
23. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In *Proc. Eurocrypt'02*, pages 321-336, 2002.
24. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attack. In *Proc. Asiacrypt'02*, pages 497-514, 2002.
25. E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. In *Proc. CCS'03*, pages 241-250, 2003.
26. E. Bresson, O. Chevassut, and D. Pointcheval. New security results on encrypted key exchange. In *Proc. PKC'04*, pages 145-158, 2004.
27. E. Bresson, O. Chevassut, and D. Pointcheval. A security solution for IEEE 802.11s ad-hoc mode: password-authentication and group-Diffie-Hellman key exchange. *International Journal of Wireless and Mobile Computing*, 2(1): 4-13, 2007.
28. M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Proc. Eurocrypt'94*, pages 275-286, 1995.
29. M. Burmester and Y. Desmedt. A secure and scalable group key exchange system. *Information Processing Letters*, 94(3): 137-143, 2005.
30. R. Canetti and H. Krawczyk. Key-exchange protocols and their use for building secure channels. In *Proc. Eurocrypt'01*, pages 453-474, 2001.
31. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Proc. Eurocrypt'02*, pages 337-351, 2002.
32. R. Canetti and H. Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *Proc. Crypto'02*, pages 143-161, 2002.
33. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 32(2): 644-654, 1976.
34. W. Diffie, P. van Oorschot and M. Wiener. Authentication and authenticated key exchange. *Designs, Codes, and Cryptography*, 2(2): 107-125, 1992.
35. D. Galindo, J. Herranz and E. Kiltz. On the generic construction of identity-based signatures with additional properties. In *Proc. Asiacrypt'06*, pages 178-193, 2006.
36. C. Gentry. Practical identity-based encryption without random oracle. In *Proc. Eurocrypt'06*, pages 445-464, 2006.
37. O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In *Proc. Crypto'01*, pages 408-432, 2001.
38. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM J. Computing*, 17(2): 281-308, 1988.
39. I. Ingemarsson, D. T. Tang, and C. K. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5): 714-720, 1982.
40. M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Proc. Asiacrypt'96*, pages 36-49, 1996.
41. S. Jiang and G. Gong. Password based key exchange with mutual authentication. In *Proc. Selected Areas in Cryptography'04*, pages 267-279, 2004.

42. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Proc. Eurocrypt'01*, pages 457-494, 2001.
43. J. Katz, R. Ostrovsky, and M. Yung. Forward secrecy in password-only key exchange protocols. In *Proc. 3rd International Conference on Security in Communication Networks (SCN'03)*, pages 29-44, 2003.
44. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Proc. Crypt0'03*, pages 110-125, 2003.
45. H. J. Kim, S. M. Lee, and D. H. Lee. Constant-round authenticated group key exchange for dynamic groups. In *Proc. Asiacrypt'04*, pages 245-259, 2004.
46. Y. Kim, A. Perig, and G. Tsudik. Simper and fault-tolerant key agreement for dynamic collaborative groups. In *Proc. CCS'00*, pages 235-244, 2000.
47. Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *Proc. IFIP TC11 16th Annual Working Conference on Information Security (IFIP/SEC)*, pages 229-244, 2001.
48. J. O. Kown, I. R. Jeong, K. Sakurai and D. H. Lee. Password-authenticated multiparty key exchange with different passwords. Cryptology ePrint Archive, Report 2006/476, http://eprint.iacr.org.
49. K. G. Paterson and J. C. N. Schuldt, "Efficient identity-based signatures secure in the standard model", In *Proc. ACISP'06*, pages 207-222, 2006.
50. S. Patel. Number-theoretic attack on secure password scheme. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 236-247, 1997.
51. D. Steer, L. Strawczynski, W. Diffie and M. Wiener. A secure audio teleconference system. In *Proc. Crypto'98*, pages 520-528, 1998.
52. M. Steiner, G. Tsudik, and M. Widner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8): 769-780, 2000.
53. W. G. Tzeng. A practical and secure fault-tolerant conference key agreement protocol. In *Proc. PKC'00*, pages 1-13, 2000.
54. B. Waters. Efficient identity-based encryption without random oracles. In *Proc. Eurocrypt'05*, pages 114-127, 2005.