Simulatable Certificateless Two-Party Authenticated Key Agreement Protocol

Lei Zhang[a,b,*], Futai Zhang[b,c], Qianhong Wu[a,d], Josep Domingo-Ferrer[a]

[a] *Universitat Rovira i Virgili*
Department of Computer Engineering and Mathematics
UNESCO Chair in Data Privacy
Av. Països Catalans 26, E-43007 Tarragona, Catalonia
[b] *Nanjing Normal University*
School of Computer Science and Technology
Nanjing 210097, P.R. China
[c] *Jiangsu Engineering Research Center on Information Security and Privacy Protection Technology, Nanjing, P.R. China*
[d] *Wuhan University, School of Computer, Key Lab. of Aerospace Information Security and Trusted Computing, Ministry of Education, P.R. China*
email: lei.zhang@urv.cat, zhangfutai@njnu.edu.cn,
{qianhong.wu,josep.domingo}@urv.cat

Abstract

Key agreement (KA) allows two or more users to negotiate a secret session key among them over an open network. Authenticated key agreement (AKA) is a KA protocol enhanced to prevent active attacks. AKA can be achieved using a public key infrastructure (PKI) or identity-based cryptography. However, the former suffers from a heavy certificate management burden while the latter is subject to the so-called key escrow problem. Recently, certificateless cryptography was introduced to mitigate these limitations. In this paper, we first propose a security model for AKA protocols using certificateless cryptography. Following this model, we then propose a simulatable certificateless two-party AKA protocol. Security is proven under the standard computational Diffie-Hellman (CDH) and bilinear Diffie-Hellman (BDH) assumptions. Our protocol is efficient and practical, because it requires only one pairing operation and five multiplications by each party.

Keywords: Information security, Protocol design, Certificateless cryptography, Authenticated key agreement, Provable security.

[*] Corresponding author

1 Introduction

Key agreement (KA) is one of the fundamental cryptographic primitives. It allows two or more parties to establish a secret key over open networks; each party can encrypt any message such that only the parties sharing the secret session key can decrypt the message. This kind of plain KA protocols [11,28] are only secure against passive adversaries who limit themselves to eavesdropping communications between parties. In the real world, the adversary may mount more powerful attacks, *e.g.*, by impersonating one party to communicate with another party. The notion of authenticated key agreement (AKA) [9,23,25] has been proposed to defeat such active adversaries. AKA protocols can be realized using a public-key infrastructure (PKI) or identity-based (ID-based) cryptography. However, PKI-based systems suffer from a heavy certificate management burden while ID-based cryptographic systems require all participants to fully trust an authority. The latter is a very strong assumption, hard to meet in practice, especially over open networks. This paper focuses on AKA protocols which do not suffer from such limitations. In particular, we investigate two-party AKA protocols based on certificateless cryptography.

1.1 Related Work

The first two-party KA protocol in the literature is the Diffie-Hellman protocol [11]. However, the basic Diffie-Hellman protocol does not authenticate the two communicating entities and it is insecure against active attacks, *e.g.*, the man-in-the-middle attack in which an adversary talks to both participants and impersonates one participant in front of the other participant. The notion of authenticated KA aims at ensuring security versus active adversaries who may impersonate one party to cheat another party during the execution of KA protocols. A number of KA protocols [5,9,18,23,25,29] have been proposed to achieve security against active attacks, an essential feature of secure communications in the real world.

The security of KA protocols has to be formally proven before they are deployed. The first work on formal security of KA protocols is due to Bellare and Rogaway [2]. Their model was further investigated by Blake-Wilson *et al.* [5] and Bellare *et al.* [4]. Although these models are generally accepted to formalize the security of KA protocols, few key agreement protocols can be proven secure in these models. The main obstacle in proofs is that, without knowledge of the private key owned by the participants or the ability to solve the underlying hard problems, it is hard for the simulator to answer the *reveal* query which captures the *known-key security* property (see Section 2.2).

Several proposals have been presented to deal with the key reveal query. In [10], Cheng *et al.* introduced the concept of coin queries, which are used to force the adversary to reveal her ephemeral secret and hence make it possible for the simulator to deal with the reveal query. However, the possibility of an adversary breaking a protocol without knowing the ephemeral secret is not modeled. Another approach was proposed by Kudla and Paterson [15]. Their approach relies on Gap assumptions [20] which assume that it is still difficult to solve the computational version of some hard problems, even with the help of a corresponding decisional oracle. However, it is unknown whether such an oracle is available for many hard problems used to build KA protocols. Recently, Chen *et al.* [8] presented a new approach referred to as *built-in decision function* to deal with the key reveal query. This approach is employed to convert a hard decisional problem into an easy decisional one.

AKA protocols can be realized in the PKI- or ID-based cryptographic setting. In the PKI setting, each user has a public key and a partially trusted certificate authority signs it and generates a certificate to bind the user to her public key. Then any two users can authenticate their communication in an explicit or implicit way when they run the KA protocol. To validate the established session key, each party needs to verify the certificate of the other party. It is a heavy burden to generate, deliver, maintain, and verify certificates in open networks. An alternative is to realize AKA protocols in an ID-based cryptographic setting [6,19,21]. In this scenario, a fully trusted authority called Private Key Generator (PKG) generates a private key for each user in the system taking as input the user's identity. The user's identity serves as the public key of the user. Henceforth, AKA protocols are realized similarly to those in the PKI setting. In this way, the system eliminates the certificate management burden but suffers from the key-escrow problem, because all parties must fully trust PKG. Some ID-based AKA protocols [9,18] claim that they are free from *session key escrow* in the sense that PKG cannot recover the agreed session key. One may observe that this is not entirely true, since PKG knows each entity's private key and a malicious PKG can always launch a man-in-the-middle attack to obtain the secret session key.

AKA protocols can also be implemented in the certificateless cryptographic setting. Certificateless cryptography (CLC) [1,7,12,13,14,24,30] was recently introduced to circumvent the key escrow problem inherent to ID-based cryptography and the certificate management burden in traditional PKI-based cryptosystems. In CLC, a partially trusted authority called Key Generation Center (KGC) helps to generate the private key for each user but cannot access the full private key of any user. Hence, CLC avoids the key escrow problem. The public key of a user is computed from KGC's public parameters and a secret value chosen by the user. Hence, CLC does not need an additional certificate to bind the user to her public key. Several certificateless two-party AKA protocols [1,16,17,22,26,27] have been presented. Nevertheless, none of them has been proven secure with a formal proof. As for efficiency, most of the existing protocols suffer from heavy pairing computation during the session key establishment phase. Thus, to really take advantage of certificateless cryptography, it is essential to build certificateless two-party AKA protocols which are provably secure and efficient.

1.2 Contributions

By integrating certificateless cryptography and KA protocols, this paper investigates provably secure and efficient certificateless AKA protocols to facilitate their application. We define a formal security model for certificateless AKA protocols and we propose a novel implementation based on bilinear pairings. Our proposal is equipped with the following useful properties:

- *Key escrow avoidance.* Our protocol is free from the long-term authentication key escrow problem. This is due to the fact that, in our proposal, the KGC does not know the private key serving as the long-term authentication key of each participant. Our protocol is also free from the *session key escrow* problem in the sense that a malicious KGC can impersonate neither party to launch the man-in-the middle attack if she does not replace the users' public keys, unlike previous AKA proposals based on ID-based cryptography. In fact, the proposed protocol achieves a trust level similar to the one of AKA protocols in the traditional PKI setting but it does not suffer from the certificate management burden.
- *Provable security.* Security depends only on some standard assumptions, *i.e.*, the bilinear Diffie-Hellman (BDH) and the computational Diffie-Hellman (CDH) assumptions. Furthermore, in our protocol, the adversary is allowed to ask key *reveal* queries *without* employing any artificial oracles or additional non-standard assumptions (*e.g.*, the Gap assumption) as in previous protocols.
- *Efficiency.* Our protocol offers strong security without sacrificing efficiency. Indeed, the proposed protocol is efficient in terms of number of rounds, communication, and computation. In particular, to establish a session key, each entity is required to execute only one pairing operation and five multiplications.

The rest of the paper is organized as follows. Section 2 defines the security model for certificateless two-party AKA protocols. We propose our simulatable certificateless two-party AKA protocol in Section 3. In Section 4, the protocol is formally analyzed under the standard BDH and CDH assumptions. Section 5 concludes the paper.

2 Modeling Certificateless Two-Party AKA Protocols

Before modeling certificateless two-party AKA protocols, we first review the adversaries in CLC. A certificateless two-party AKA protocol should resist the attacks from two types of adversaries known as '*Type I adversary*' and '*Type II adversary*'.

- Type I Adversary $\mathcal{A}_I$: This adversary models an adversary who does not know the master-key of KGC, but who has the ability to replace the public key of any entity with a value of her choice.
- Type II Adversary $\mathcal{A}_{II}$: This adversary models a malicious KGC who knows the master-key, but who cannot replace the target user's public key.

Note that our definitions of Type I and II Adversaries are stronger than those in [1]. In fact, the adversaries in our definitions have a power similar to the "super adversaries" defined in [14], which are the most powerful adversaries considered in the CLC literature.

2.1 Algorithms of a Certificateless Two-Party AKA Protocol

A certificateless two-party AKA protocol consists of six polynomial-time algorithms: Setup, Partial-Private-Key-Extract, Set-Secret-Value, Set-Private-Key, Set-Public-Key and Key-Agreement. These algorithms are defined as follows.

- Setup: This algorithm is run by KGC. It takes as input a security parameter ℓ and returns a master-key and a list of system parameters params.
- Partial-Private-Key-Extract: This algorithm is also run by KGC. It takes as input the parameter list params, master-key and an entity's identity ID_i, to produce the entity's partial private key D_i.
- Set-Secret-Value: This algorithm takes as input a parameter list params and an entity's identity ID_i to produce the entity's secret value x_i.
- Set-Private-Key: This algorithm takes as input a parameter list params, an entity's identity ID_i, a partial private key D_i and a secret value x_i to produce a private key S_i for this entity.
- Set-Public-Key: This algorithm takes as input a parameter list params, an entity's identity ID_i and secret value x_i to produce a public key P_i for the entity.
- Key-Agreement: This is a probabilistic polynomial-time interactive algorithm which involves two entities A and B. The inputs are the system parameters params for both A and B, plus (S_A, ID_A, P_A) for A, and (S_B, ID_B, P_B) for B. Here, S_A, S_B are the respective private keys of A and B; ID_A is the identity of A and ID_B is the identity of B; P_A, P_B are the public keys of A and B, respectively. Eventually, if the protocol does not fail, A and B

obtain a secret session key $K_{AB} = K_{BA} = K$.

2.2 Security Definitions

Before formally defining the security of certificateless two-party AKA protocols, we first informally discuss the relevant security requirements to give an intuition. The most important requirement in AKA protocols is *key secrecy*, which means that any (active or passive) attacker cannot learn any information about the negotiated session key between the two designated parties in the protocol. Besides the key secrecy required by AKA protocols, the following security properties are also useful in practice.

- *Known-key security*: If the protocol does not fail, both parties will share the same secret session key and, if some session keys were leaked, this should not compromise the key secrecy of any other session key.
- *Unknown key-share*: If A thinks she is sharing a key with an entity B, then it should not happen that A is actually sharing that key with another entity C, where $C \neq B$.
- *Key-compromise impersonation*: The compromise of A's private key will allow an adversary to impersonate A, but it should not allow the adversary to establish a session key with A by masquerading as a different entity B.
- *Forward secrecy*: If the private keys of A and B are compromised, the secrecy of previously established session keys by those entities is not compromised; this is sometimes also called *perfect forward secrecy.* A weaker notion is *partial forward secrecy*: the compromise of either A's private key or B's private key does not endanger the secrecy of previously established session keys.
- *Key control*: Neither entity should be able to force the session key to a preselected value.

Motivated by the security model of Chen *et al.* [8] derived from [2,4,5], we present a security model for AKA protocols in the setting of CLC. The model is defined by the following game between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A} \in \{\mathcal{A}_I, \mathcal{A}_{II}\}$. The adversary controls the communications from and to the protocol participants. Each participant i has an identity ID_i. The adversary's behavior is modeled by a number of oracles maintained by $\mathcal{CH}$. We use the oracle $\prod_{i,j}^{s}$ to represent the s-th instance of participant i and partner participant j in a session.

At the beginning of the game, $\mathcal{CH}$ runs the Setup algorithm, takes as input a security parameter ℓ to obtain the $\mathsf{master\text{-}key}$ and the system parameter list params. If $\mathcal{A}$ is a Type I adversary $\mathcal{A}_I$, $\mathcal{CH}$ sends params to $\mathcal{A}$ and keeps the $\mathsf{master\text{-}key}$ secret; otherwise, $\mathcal{A}$ is a Type II adversary $\mathcal{A}_{II}$, and $\mathcal{CH}$ sends params with master-key to $\mathcal{A}$.

$\mathcal{A}$ is modeled by a probabilistic polynomial-time Turing Machine. All communications go through the adversary $\mathcal{A}$. Participants only respond to the queries by $\mathcal{A}$ and do not communicate directly among themselves. $\mathcal{A}$ can relay, modify, delay, interleave or delete all the message flows in the system. Note that $\mathcal{A}$ can act as a *benign* adversary, which means that $\mathcal{A}$ is deterministic and restricts her action to choosing a pair of oracles $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$ and then faithfully conveying each message flow from one oracle to the other. Furthermore, $\mathcal{A}$ may ask a polynomially bounded number of the following queries, including one Test query defined as follows:

- Create(ID_i): This allows $\mathcal{A}$ to ask $\mathcal{CH}$ to set up a new participant i with identity ID_i. On receiving such a query, $\mathcal{CH}$ generates the public/private key pair for i.
- Public-Key(ID_i): $\mathcal{A}$ can request the public key of a participant i whose identity is ID_i. To respond, $\mathcal{CH}$ outputs the public key P_i of participant i.
- Partial-Private-Key(ID_i): $\mathcal{A}$ can request the partial private key of a participant i whose identity is ID_i. To respond, $\mathcal{CH}$ outputs the partial private key D_i of participant i.
- Corrupt(ID_i): $\mathcal{A}$ can request the private key of a participant i whose identity is ID_i. To respond, $\mathcal{CH}$ outputs the private key S_i of participant i.
- Public-Key-Replacement(ID_i, P_i'): For a participant i whose identity is ID_i, $\mathcal{A}$ can choose a new public key P_i' and then set P_i' as the new public key of this participant. $\mathcal{CH}$ will record these replacements which will be used later.
- Send$(\prod_{i,j}^{n}, M)$: $\mathcal{A}$ can send a message M of her choice to an oracle, say $\prod_{i,j}^{n}$, in which case participant i assumes that the message has been sent by participant j. $\mathcal{A}$ may also make a special Send query with $M = \lambda$ to an oracle $\prod_{i,j}^{n}$, which instructs i to initiate a protocol run with j. An oracle is an initiator oracle if the first message it has received is λ. If an oracle does not receive a message λ as its first message, then it is a responder oracle.
- Reveal$(\prod_{i,j}^{n})$: $\mathcal{A}$ can ask a particular oracle to reveal the session key (if any) it currently holds to $\mathcal{A}$.
- Test$(\prod_{I,J}^{T})$: At some point, $\mathcal{A}$ may choose one of the oracles, say $\prod_{I,J}^{T}$, to ask a single Test query. This oracle must be fresh (see Definition 2). To answer the query, the oracle flips a fair coin $b \in \{0, 1\}$, and returns the session key held by $\prod_{I,J}^{T}$ if $b = 0$, or a random sample from the distribution of the session key if $b = 1$.

An oracle $\prod_{i,j}^{n}$ will be in one of the following states:

- *Accepted:* This oracle is in the state *Accepted* if it has accepted the request to establish a session key.
- *Rejected:* This oracle is in the state *Rejected* if it has rejected the request to establish a session key and has aborted the protocol.

- *State *:* This oracle is in the state * if it has not made any decision to accept or reject.
- *Opened:* This oracle is in the state *Opened* if it has answered a Reveal query.

Definition 1 (Matching conversation) *Let the session ID be the concatenation of the messages in a session. Two oracles $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$ are said to have a matching conversation with each other if they have the same session ID.*

Definition 2 (Fresh oracle) *An oracle $\prod_{i,j}^{n}$ is fresh if (1) $\prod_{i,j}^{n}$ is in the state Accepted; (2) $\prod_{i,j}^{n}$ is not in the state Opened; (3) party $j \neq i$ is not corrupted; (4) there is no oracle $\prod_{j,i}^{t}$ in the state Opened having a matching conversation with $\prod_{i,j}^{n}$; (5) if $\mathcal{A}$ is a Type I adversary, $\mathcal{A}$ has never requested the partial private key of participant j; and, if $\mathcal{A}$ is a Type II adversary, $\mathcal{A}$ has never replaced the public key of participant j.*

Note that this definition allows the participant i to be corrupted, and thus can be used to address the *key-compromise impersonation* property as well as the *partial forward secrecy* property.

After a Test query, the adversary can continue to query the oracles except that it cannot make a Reveal query to the test oracle $\prod_{I,J}^{T}$ or to $\prod_{J,I}^{t}$ who has a matching conversation with $\prod_{I,J}^{T}$ (if it exists), and it cannot corrupt participant J. In addition, if $\mathcal{A}$ is a Type I adversary, $\mathcal{A}$ cannot request the partial private key of the participant J; and if $\mathcal{A}$ is a Type II adversary, $\mathcal{A}$ cannot replace the public key of the participant J. At the end of the game, $\mathcal{A}$ must output a guess bit b'. $\mathcal{A}$ wins if and only if $b' = b$. $\mathcal{A}$'s advantage to win the above game, denoted by $Advantage^{\mathcal{A}}(k)$, is defined as:

$$Advantage^{\mathcal{A}}(k) = |\Pr[b' = b] - 1/2|.$$

Definition 3 *A certificateless two-party AKA protocol is said to be secure if:*

(1) In the presence of a benign adversary on $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$, both oracles always agree on the same session key, and this key is distributed uniformly at random;
(2) For any adversary, $Advantage^{\mathcal{A}}(k)$ is negligible.

Similarly to [9], if a certificateless two-party AKA protocol is secure under Definition 3, it achieves implicit mutual key authentication and the following general security properties: *known-key security*, *key-compromise impersonation*, *unknown key-share* and *partial forward secrecy.* The *key control* property can be easily achieved and we do not elaborate on it here.

3 An Efficient Certificateless Two-party AKA Protocol

Our protocol is implemented in bilinear pairings which have been extensively exploited to devise efficient cryptosystems [6,28]. We briefly review the basic notions of bilinear pairings. Let G_1 be an additive group of prime order q and G_2 be a multiplicative group of the same order. A mapping $e : G_1 \times G_1 \longrightarrow G_2$ is called a bilinear map if it satisfies the following properties:

(1) Bilinearity: $e(aP, bQ) = e(P, Q)^{ab}$ for all $P, Q \in G_1, a, b \in Z_q^*$.
(2) Non-degeneracy: There exists $P, Q \in G_1$ such that $e(P, Q) \neq 1$.
(3) Computability: There exists an efficient algorithm to compute $e(P, Q)$ for any $P, Q \in G_1$.

3.1 Protocol Description

Motivated by [15,29], we construct a secure certificateless two-party AKA protocol as follows.

- Setup: On input a security parameter ℓ, this algorithm runs as follows.

(1) Select a cyclic additive group G_1 of prime order q, a cyclic multiplicative group G_2 of the same order, a generator P of G_1, and a bilinear map $e : G_1 \times G_1 \longrightarrow G_2$.
(2) Choose a random master-key $s \in Z_q^*$ and set $P_0 = sP$;
(3) Choose cryptographic hash functions $H_1 : \{0,1\}^* \longrightarrow G_1$, $H_2 : \{0,1\}^{*2} \times G_1^3 \times G_2 \times G_1^4 \longrightarrow \{0,1\}^l$.

The system parameters are params=$(G_1, G_2, e, P, P_0, H_1, H_2, l)$. The master-key is $s \in Z_q^*$.

- Partial-Private-Key-Extract: This algorithm takes as input params, the master-key s and an entity's identity $ID_i \in \{0,1\}^*$, and generates the partial private key for the entity as follows.

(1) Compute $Q_i = H(ID_i)$.
(2) Output the partial private key $D_i = sQ_i$.

- Set-Secret-Value: This algorithm takes as input params, an entity's identity ID_i, and a random value $x_i \in Z_q^*$. It outputs x_i as the entity's secret value.
- Set-Private-Key: This algorithm takes as input params, an entity's identity ID_i, the entity's partial private key D_i and the entity's secret value $x_i \in Z_q^*$. The output of the algorithm is the private key $S_i = (x_i, D_i)$.
- Set-Public-Key: This algorithm takes as input params, an entity's identity ID_i, and the entity's secret value $x_i \in Z_q^*$ to produce the entity's public key $P_i = x_i P$.
- Key-Agreement: Assume that an entity A with identity ID_A has private key $S_A = (x_A, D_A)$ and public key $P_A = x_A P$, an entity B with identity ID_B

has private key $S_B = (x_B, D_B)$ and public key $P_B = x_B P$. A and B run the protocol as follows:

(1) A randomly chooses $r_A \in Z_q^*$, computes $R_A = r_A P$ and sends (ID_A, P_A, R_A) to B.
(2) When B receives (ID_A, P_A, R_A) from A, she selects $r_B \in Z_q^*$ at random, computes $R_B = r_B P$, and sends (ID_B, P_B, R_B) to A. Finally, B computes the session key $K_{BA} = H_2(ID_A, ID_B, R_A, R_B, r_B R_A, e(R_A + Q_A, r_B P_0 + D_B), P_A, P_B, x_B R_A, r_B P_A)$, where $Q_A = H(ID_A)$.
(3) When A receives (ID_B, P_B, R_B) from B, he computes the session key $K_{AB} = H_2(ID_A, ID_B, R_A, R_B, r_A R_B, e(R_B + Q_B, r_A P_0 + D_A), P_A, P_B, r_A P_B, x_A R_B)$, where $Q_B = H(ID_B)$.
In such a protocol run, the session ID of the protocol instance is

$$ID_A || ID_B || P_A || P_B || R_A || R_B.$$

We briefly check the correctness of the protocol. Since

$$\begin{aligned} e(R_A + Q_A, r_B P_0 + D_B) &= e(R_A + Q_A, r_B P + Q_B)^s \\ &= e(sR_A + sQ_A, r_B P + Q_B) \\ &= e(r_A P_0 + D_A, R_B + Q_B) \\ &= e(R_B + Q_B, r_A P_0 + D_A), \end{aligned}$$

it is easy too see $K_{AB} = K_{BA}$ holds. Hence, the correctness of the protocol holds.

4 Security Analysis

The security of our protocol relies on the standard CDH and BDH assumptions, which are briefly reviewed in the sequel.

Computational Diffie-Hellman (CDH) Problem in G_1: Given a generator P of G_1 and (aP, bP) for unknown $a, b \in Z_q^*$, compute abP. The CDH assumption states that the probability of any polynomial-time algorithm to solve the CDH problem is negligible.

Bilinear Diffie-Hellman (BDH) Problem: Given a tuple $(P, aP, bP, cP) \in G_1$ for random unknown $a, b, c \in Z_q^*$, compute $e(P, P)^{abc}$. The BDH assumption states that the probability of any polynomial-time algorithm to solve the BDH problem is negligible.

Assuming that the CDH and BDH problems are hard, we now prove the security of the above key agreement protocol according to Definition 3.

Lemma 1 *In the presence of a benign adversary on $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$, both oracles always agree on the same session key as if there was no adversary, and this key is distributed uniformly at random.*

Proof. Suppose that the two participants i and j follow the protocol and $\mathcal{A}$ is benign. In this case, both oracles receive correctly formatted messages looking exactly as originally sent by the other oracle; hence, they will agree on the same session key. Since r_i and r_j are randomly selected by participants i and j, respectively, the session key can be considered as the output of the hash function H_2 on a random input. Based on the properties of cryptographic hash functions, the session key is uniformly distributed over $\{0,1\}^l$. $\square$

Lemma 2 *Under the assumption that the BDH problem is intractable, the advantage of a Type I adversary against our certificateless two-party AKA protocol is negligible in the random oracle model [3]. Specifically, suppose that, in the attack, a Type I adversary $\mathcal{A}$, who makes at most q_{H_2} times H_2 queries and creates at most q_c participants, wins the game with advantage $Advantage^{\mathcal{A}}(k)$. Then there exists an algorithm $\mathcal{CH}$ to solve the BDH problem with advantage $\frac{1}{q_c^2 \cdot q_s \cdot q_{H_2}} Advantage^{\mathcal{A}}(k)$, where q_s is the maximal number of sessions each participant may be involved in.*

Proof. Suppose that there exists a Type I adversary $\mathcal{A}$ who can win the game defined in Section 2 with a non-negligible advantage $Advantage^{\mathcal{A}}(k)$ in polynomial time t. Then we show that there is an algorithm $\mathcal{CH}$ that solves the BDH problem with a non-negligible probability. Suppose that $\mathcal{CH}$ is given an arbitrary input (P, aP, bP, cP) of the BDH problem. We show how $\mathcal{CH}$ can use $\mathcal{A}$ to solve the BDH problem, *i.e.* to compute $e(P,P)^{abc}$.

All queries by the adversary $\mathcal{A}$ now pass through $\mathcal{CH}$. $\mathcal{CH}$ sets $P_0 = aP$ and selects the system parameters $\mathsf{params}=(G_1, G_2, e, P, P_0, H_1, H_2, l)$. params is sent to $\mathcal{A}$. In this way, the game is initialized.

H_1 query: $\mathcal{CH}$ maintains an initially empty list $\mathbf{H_1}$ which contains tuples of the form (ID_i, d_i, Q_i). Suppose $\mathcal{A}$ can query H_1 for at most q_{H_1} times. $\mathcal{CH}$ chooses distinct $I, J \in [1, q_{H_1}]$. On receiving a query $H_1(ID_i)$, the same answer from the list $\mathbf{H_1}$ will be given if the request has been asked before. If $ID_i \neq ID_J$, $\mathcal{CH}$ picks a random $d_i \in Z_q^*$ and computes $Q_i = d_i P$; while $ID_i = ID_J$, $\mathcal{CH}$ sets $d_i = null$, $Q_i = bP$. In both cases, $\mathcal{CH}$ returns Q_i as the answer and adds (ID_i, d_i, Q_i) to $\mathbf{H_1}$.

$\mathsf{Create}(ID_i)$: $\mathcal{CH}$ maintains an initially empty list $\mathbf{C}$ consisting of tuples of the form (ID_i, D_i, x_i, P_i). For simplicity, we assume that all the Create queries are distinct. On receiving a Create query on ID_i, $\mathcal{CH}$ first makes an H_1 query $H_1(ID_i)$ to obtain a tuple (ID_i, d_i, Q_i), then does as follows.

- If $ID_i \neq ID_J$, choose a random $x_i \in Z_q^*$, and compute the public key $P_i = x_iP$ and the partial private key $D_i = d_iP_0$. The tuple (ID_i, D_i, x_i, P_i) is added to $\mathbf{C}$.
- If $ID_i = ID_J$, choose a random $x_i \in Z_q^*$, set $P_i = x_iP$, set $D_i = null$, then add (ID_i, D_i, x_i, P_i) to $\mathbf{C}$.

Without loss of generality, we assume that, before asking the following queries, $\mathcal{A}$ has already asked some Create queries on the related identities.

H_2 query: Suppose $\mathcal{A}$ can query H_2 for at most q_{H_2} times. $\mathcal{CH}$ maintains a list $\mathbf{H_2}$ which contains tuples of the form $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$. On receiving a query $H_2(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i)$, the same answer from the list $\mathbf{H_2}$ will be given if the request has been asked before. Otherwise, $\mathcal{CH}$ does as follows.

- If there exists a tuple $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$ on $\mathbf{S}$ (maintained in the Send query below) such that $ID_i \neq ID_J$ and
  · $\prod_{i,j}^n$ is an initiator, $ID_i^a = ID_i, ID_i^b = ID_j$, $R_i^a = M_{i,j}^n$, $R_i^b = M_{j,i}^n, X_i = r_{i,j}^n M_{j,i}^n$, $y_i = e(M_{j,i}^n + Q_j,\ r_{i,j}^n P_0 + D_i)$, $P_i^a = P_i^n$, $P_i^b = P_j^n$, $Y_i = r_{i,j}^n P_j^n$, $e(Z_i, P) = e(P_i^n, M_{j,i}^n)$, $SK_{i,j}^n \neq null$ (where D_i is the partial private key of the participant whose identity is ID_i), or
  · $\prod_{i,j}^n$ is a responder, $ID_i^a = ID_j, ID_i^b = ID_i$, $R_i^a = M_{j,i}^n, R_i^b = M_{i,j}^n$, $X_i = r_{i,j}^n M_{j,i}^n$, $y_i = e(M_{j,i}^n + Q_j, r_{i,j}^n P_0 + D_i)$, $P_i^a = P_j^n, P_i^b = P_i^n, e(Y_i, P) = e(P_i^n, M_{j,i}^n)$, $Z_i = r_{i,j}^n P_j^n$, $SK_{i,j}^n \neq null$,

  set $h_i = SK_{i,j}^n$, add $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i,\ y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ to $\mathbf{H_2}$ and return h_i as the answer.
- Else if there exists a tuple $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$ on $\mathbf{S}$ such that $ID_i = ID_J$ and
  · $\prod_{i,j}^n$ is an initiator and $ID_i^a = ID_i, ID_i^b = ID_j$, $R_i^a = M_{i,j}^n, R_i^b = M_{j,i}^n$, $e(X_i, P) = e(M_{i,j}^n, M_{j,i}^n)$, $P_i^a = P_i^n, P_i^b = P_j^n$, $e(Y_i, P) = e(M_{i,j}^n, P_j^n)$, $e(Z_i, P) = e(P_i^n, M_{j,i}^n)$, $SK_{i,j}^n \neq null$,
  · or $\prod_{i,j}^n$ is a responder and $ID_i^a = ID_j, ID_i^b = ID_i$, $R_i^a = M_{j,i}^n, R_i^b = M_{i,j}^n$, $e(X_i, P) = e(M_{i,j}^n, M_{j,i}^n)$, $P_i^a = P_j^n, P_i^b = P_i^n$, $e(Y_i, P) = e(P_i^n, M_{j,i}^n)$, $e(Z_i, P) = e(M_{i,j}^n,\ P_j^n)$, $SK_{i,j}^n \neq null$,

  compute

$$\begin{aligned} y_i^n &= e(M_{j,i}^n + d_jP, aM_{i,j}^n + D_i) \\ &= e(\frac{1}{r_{i,j}^n a} X_i + d_jP, r_{i,j}^n aaP + abP) \\ &= e(X_i, aP + \frac{1}{r_{i,j}^n} bP) e(d_jaP, r_{i,j}^n aP + bP) \end{aligned}$$

  (where d_j can be found in $\mathbf{H_1}$ of the form (ID_j, d_j, Q_j)). If $y_i^n = y_i$, set $h_i = SK_{i,j}^n$, return h_i as the answer and add $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ to $\mathbf{H_2}$. Otherwise, randomly choose $h_i \in \{0,1\}^l$, return h_i as the answer and add $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ to $\mathbf{H_2}$.

- Else, randomly choose $h_i \in \{0,1\}^l$, return h_i as the answer and add $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ to $\mathbf{H_2}$.

Public-Key(ID_i): On receiving this query, $\mathcal{CH}$ first searches for a tuple (ID_i, D_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i, then returns P_i as the answer.

Partial-Private-Key(ID_i): Whenever $\mathcal{CH}$ receives this query, if $ID_i = ID_J$, $\mathcal{CH}$ aborts (Event 1); else, $\mathcal{CH}$ searches for a tuple (ID_i, D_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i and returns D_i as the answer.

Corrupt(ID_i): Whenever $\mathcal{CH}$ receives this query, if $ID_i = ID_J$, $\mathcal{CH}$ aborts (Event 2); else, $\mathcal{CH}$ searches for a tuple (ID_i, D_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i and if $x_i = null$, $\mathcal{CH}$ returns $null$; otherwise, $\mathcal{CH}$ returns (x_i, D_i) as the answer.

Public-Key-Replacement(ID_i, P_i'): On receiving this query, $\mathcal{CH}$ searches for a tuple (ID_i, D_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i, then updates P_i to P_i' and sets $x_i = null$.

Send$(\prod_{i,j}^n, M)$: $\mathcal{CH}$ maintains an initially empty list $\mathbf{S}$ consisting of tuples of the form $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$, which means that $M_{j,i}^n$ is the incoming message, P_j^n is the public key of the participant j that $\prod_{i,j}^n$ received, P_i^n is the current public key owned by participant i; $r_{i,j}^n, M_{i,j}^n$ are defined below. On receiving a query Send$(\prod_{i,j}^n, M)$ (if $M \neq \lambda$, $\mathcal{CH}$ sets $M_{j,i}^n = M$; otherwise, at the end of the protocol, a message will be returned; if $\prod_{i,j}^n$ is in *Accepted*, then $\mathcal{CH}$ sets this message to be $M_{j,i}^n$), the same answer from the list $\mathbf{S}$ will be given if the query has been asked before; if the query has not been asked, $\mathcal{CH}$ does as follows:

- If $n = T$, $ID_i = ID_I$ and $ID_j = ID_J$, set $SK_{i,j}^n = r_{i,j}^n = null$, set $M_{i,j}^n = cP$, return $M_{i,j}^n$ as the answer and add $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$ to $\mathbf{S}$.
- Else, if $ID_i \neq ID_J$, choose random $r_{i,j}^n \in Z_q^*$, compute $M_{i,j}^n = r_{i,j}^n P$, return $M_{i,j}^n$ as the answer, set $SK_{i,j}^n = null$ and add $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$ to $\mathbf{S}$.
- Else, choose a random $r_{i,j}^n \in Z_q^*$, compute $M_{i,j}^n = r_{i,j}^n P_0$, return $M_{i,j}^n$ as answer, set $SK_{i,j}^n = null$, and add $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$ to $\mathbf{S}$.

Reveal$(\prod_{i,j}^n)$: On receiving a query Reveal$(\prod_{i,j}^n)$, $\mathcal{CH}$ first calls Send$(\prod_{i,j}^n, M)$, then looks up the list $\mathbf{S}$ for a tuple $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$. If $SK_{i,j}^n \neq null$, $\mathcal{CH}$ returns $SK_{i,j}^n$ as the answer. Otherwise, $\mathcal{CH}$ searches for a tuple (ID_i, D_i, x_i, P_i) which is indexed by ID_i in $\mathbf{C}$ and does as follows.

- Abort if $n = T$, $ID_i = ID_I$ and $ID_j = ID_J$, or $\prod_{i,j}^n$ is the oracle who has a matching conversation with $\prod_{I,J}^{T}$ (Event 3).

- Else if $ID_i \neq ID_J$
  - · If $\prod_{i,j}^{n}$ is an initiator and there is a tuple $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ in $\mathbf{H_2}$ such that $ID_i = ID_i^a, ID_j = ID_i^b$, $M_{i,j}^n = R_i^a$, $M_{j,i}^n = R_i^b$, $r_{i,j}^n M_{j,i}^n = X_i, e(M_{j,i}^n + Q_j, r_{i,j}^n P_0 + D_i) = y_i, P_i^n = P_i^a, P_j^n = P_i^b, r_{i,j}^n P_j^n = Y_i$, $e(P_i^n, M_{j,i}^n) = e(Z_i, P)$, set $SK_{i,j}^n = h_i$ and return $SK_{i,j}^n$ as the answer.
  - · Else if $\prod_{i,j}^{n}$ is a responder and there is a tuple $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ in $\mathbf{H_2}$ such that $ID_j = ID_i^a, ID_i = ID_i^b$, $M_{j,i}^n = R_i^a, M_{i,j}^n = R_i^b$, $r_{i,j}^n M_{j,i}^n = X_i$, $e(M_{j,i}^n + Q_j, r_{i,j}^n P_0 + D_i) = y_i$, $P_j^n = P_i^a, P_i^n = P_i^b$, $e(P_i^n, M_{j,i}^n) = e(Y_i, P)$, $r_{i,j}^n P_j^n = Z_i$, set $SK_{i,j}^n = h_i$ and return $SK_{i,j}^n$ as the answer.
  - · Otherwise, randomly sample $SK_{i,j}^n \in \{0,1\}^l$ and return $SK_{i,j}^n$ as the answer.
- Else $(ID_i = ID_J)$
  - · If $\prod_{i,j}^{n}$ is an initiator and there exists a tuple $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ in $\mathbf{H_2}$ such that $ID_i = ID_i^a, ID_j = ID_i^b$, $M_{i,j}^n = R_i^a, M_{j,i}^n = R_i^b$, $e(M_{i,j}^n, M_{j,i}^n) = e(X_i, P)$, $P_i^n = P_i^a, P_j^n = P_i^b$, $e(M_{i,j}^n, P_j^n) = e(Y_i, P)$, $e(P_i^n, M_{j,i}^n) = e(Z_i, P)$, compute y_i^n as follows:

$$\begin{aligned} y_i^n &= e(M_{j,i}^n + d_j P, aM_{i,j}^n + D_i) \\ &= e(\frac{1}{r_{i,j}^n a} X_i + d_j P, r_{i,j}^n aaP + abP) \\ &= e(X_i, aP + \frac{1}{r_{i,j}^n} bP) e(d_j aP, r_{i,j}^n aP + bP) \end{aligned}$$

    If $y_i^n = y_i$, set $SK_{i,j}^n = h_i$ and return $SK_{i,j}^n$ as the answer. Otherwise, randomly sample $SK_{i,j}^n \in \{0,1\}^l$ and return $SK_{i,j}^n$ as the answer.
  - · Else if $\prod_{i,j}^{n}$ is a responder and there exists a tuple $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ in $\mathbf{H_2}$ such that $ID_j = ID_i^a, ID_i = ID_i^b$, $M_{j,i}^n = R_i^a, M_{i,j}^n = R_i^b$, $e(M_{i,j}^n, M_{j,i}^n) = e(X_i, P)$, $P_j^n = P_i^a, P_i^n = P_i^b$, $e(P_i^n, M_{j,i}^n) = e(Y_i, P)$, $e(R_i^b, P_i^a) = e(Z_i, P)$, set $y_i^n = e(X_i, aP + \frac{1}{r_{i,j}^n} bP) e(d_j aP, r_{i,j}^n aP + bP)$. If $y_i^n = y_i$, set $SK_{i,j}^n = h_i$ and return $SK_{i,j}^n$ as the answer. Otherwise, randomly sample $SK_{i,j}^n \in \{0,1\}^l$, return $SK_{i,j}^n$ as the answer.

Test($\prod_{I,J}^{T}$): At some point, $\mathcal{A}$ will ask a Test query on some oracle. If $\mathcal{A}$ does not choose one of the oracles $\prod_{I,J}^{T}$ to ask the Test query, then $\mathcal{CH}$ aborts (Event 4). Otherwise, $\mathcal{CH}$ simply outputs a random value ω in $\{0,1\}^l$.

Once $\mathcal{A}$ finishes her queries and returns her guess bit (there must be a tuple $(\prod_{I,J}^{T}, null, cP, M_{J,I}^T, P_I^T, P_J^T, SK_{I,J}^T)$ in $\mathbf{S}$), $\mathcal{CH}$ proceeds as follows. If $\prod_{I,J}^{T}$ is an initiator oracle and there is a tuple $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ in $\mathbf{H_2}$ such that $R_i^a = cP, R_i^b = M_{j,i}^n$ or $\prod_{I,J}^{T}$ is a responder and there is a tuple in $\mathbf{H_2}$ such that $R_i^b = cP, R_i^a = M_{j,i}^n$ (if M is the received message, then it holds that $M = M_{j,i}^n$), $\mathcal{CH}$ checks whether $e(R_i^a, R_i^b) = e(X_i, P)$ holds. If no such tuple exists, $\mathcal{CH}$ aborts (Event 5). Otherwise, $\mathcal{CH}$ computes

$$\begin{aligned} y_i^n &= e(M + Q_J, cP_0 + D_I) \\ &= e(M + bP, acP + l_I aP) \\ &= e(P, P)^{abc} e(bP, l_I aP) e(M, acP + l_I aP) \\ &= e(P, P)^{abc} e(bP, l_I aP) e(X_i, aP) e(l_I M, aP) \\ &= e(P, P)^{abc} \cdot u \end{aligned}$$

Algorithm $\mathcal{CH}$ randomly chooses y_i from $\mathbf{H_2}$ and returns y_i/u as the response to the BDH challenge.

Claim 1 *If $\mathcal{CH}$ does not abort the above game, $\mathcal{A}$ cannot find any inconsistency between the simulation and the real world.*

Proof. The simulations of all the random oracles are valid and the messages of the oracles are uniformly distributed in the message space. The simulator makes use of the programmability of random oracle H_2 and the bilinearity of the pairing to instantiate a Decisional Diffie-Hellman (DDH) oracle and keep the consistency with responses to the H_2 queries and the Reveal queries. Hence, $\mathcal{A}$ cannot notice any difference with the real world. □

Claim 2 *Let* Event 6 *be that $\kappa = e(M + Q_J, cP_0 + D_I)$ was not queried on H_2. Then* $\Pr[\overline{\text{Event 5}} \wedge \overline{\text{Event 6}}] \geq$ *Advantage*$^{\mathcal{A}}(k)$.

Proof. Because H_2 is a random oracle, if Event 5 or Event 6 happens, $\mathcal{A}$ can only win the game in two ways:

- Case 1: $\mathcal{A}$ randomly guesses whether ω is $SK_{I,J}^T$ or not, if $SK_{I,J}^T$ has not been decided yet.
- Case 2: $\mathcal{A}$ knows that it has revealed an oracle $\prod_{i,j}^n$ which has the same session key as the challenge oracle $\prod_{I,J}^T$.

Note here that Event 5 or Event 6 could happen in Case 2 in general. This is because we make use of the programmability of random oracle H_2 and $\mathcal{CH}$ may have responded to the Reveal query without knowing the corresponding κ in the simulation. Since the random oracle should respond uniquely to each query in the simulation, the adversary can surely win the game if it knows the session key, even if it was not generated by a query to the random oracle H_2.

One may notice that the identifiers and the transcript are used as the inputs of H_2 to generate the session key. Hence, $\prod_{i,j}^n$ and $\prod_{I,J}^T$ obtain the same session key with probability greater than $1/2^\ell$ only if $\prod_{i,j}^n$ is either the chosen fresh oracle $\prod_{I,J}^T$ or an oracle $\prod_{J,I}^m$ which has the same transcript of $\prod_{I,J}^T$ for some m. According to the rules of the game, $\mathcal{A}$ is allowed to reveal neither oracle. That is, neither the fresh test oracle nor the oracle with matching conversation should be revealed. Since the identifiers and the transcript are part of the input of H_2, in this protocol, Case 2 happens with probability at most $1/2^\ell$.

It follows that

$$\Pr[\mathcal{A}\ wins|\textsf{Event 5} \vee \textsf{Event 6}] = 1/2$$

Hence, we have

$$\begin{aligned}
Advantage^{\mathcal{A}}(k) + 1/2 &= \Pr[\mathcal{A}\ wins] \\
&= \Pr[\mathcal{A}\ wins|\textsf{Event 5} \vee \textsf{Event 6}]\Pr[\textsf{Event 5} \vee \textsf{Event 6}] \\
&+ \Pr[\mathcal{A}\ wins|\overline{\textsf{Event 5} \vee \textsf{Event 6}}]\Pr[\overline{\textsf{Event 5} \vee \textsf{Event 6}}] \\
&\le 1/2 + \Pr[\overline{\textsf{Event 5} \vee \textsf{Event 6}}] \\
&= 1/2 + \Pr[\overline{\textsf{Event 5}} \wedge \overline{\textsf{Event 6}}]
\end{aligned}$$

The claim follows. □

Let Event 7 be that, in the attack, adversary $\mathcal{A}$ indeed chose oracle $\prod_{I,J}^{T}$ as the challenge oracle. It is clear that Events 1, 2, 3, 4 will not happen if Event 7 happens and vice versa. Therefore,

$$\Pr[\overline{\textsf{Event 1} \vee \textsf{Event 2} \vee \textsf{Event 3} \vee \textsf{Event 4}}] = \Pr[\textsf{Event 7}] \ge \frac{1}{q_c^2 \cdot q_s}$$

Let Event 8 be that $\mathcal{A}$ did not abort in the game, and Event 9 be that $\mathcal{CH}$ found the correct y_i. Accordingly, we have

$$\begin{aligned}
\Pr[\mathcal{CH}\ wins] &= \Pr[\textsf{Event 8} \wedge \overline{\textsf{Event 6}} \wedge \textsf{Event 9}] \\
&= \Pr[\textsf{Event 7} \wedge \overline{\textsf{Event 5}} \wedge \overline{\textsf{Event 6}} \wedge \textsf{Event 9}] \\
&\ge \frac{1}{q_c^2 \cdot q_s \cdot q_{H_2}} \Pr[\overline{\textsf{Event 5}} \wedge \overline{\textsf{Event 6}}] \\
&\ge \frac{1}{q_c^2 \cdot q_s \cdot q_{H_2}} Advantage^{\mathcal{A}}(k).
\end{aligned}$$

□

Lemma 3 *The advantage of a Type II adversary against our certificateless two-party AKA protocol is negligible in the random oracle model assuming that the CDH problem is hard. Specifically, suppose that, in the attack, a Type II adversary $\mathcal{A}$, which makes at most q_{H_2} H_2 queries and creates at most q_c participants, wins the game with advantage $Advantage^{\mathcal{A}}(k)$. Then there exists an algorithm $\mathcal{CH}$ to solve the CDH problem with advantage $\frac{1}{q_c^2 \cdot q_s \cdot q_{H_2}} Advantage^{\mathcal{A}}(k)$, where q_s is the maximal number of sessions each participant may be involved in.*

Proof. Suppose that there exists a Type II adversary $\mathcal{A}$ who can win the game defined in Section 2 with a non-negligible advantage $Advantage^{\mathcal{A}}(k)$ in polynomial time t. Then we show that there is an algorithm $\mathcal{CH}$ that solves the CDH problem with non-negligible probability. Suppose $\mathcal{CH}$ is given an arbitrary input (P, aP, bP) of the CDH problem. We show how $\mathcal{CH}$ can use $\mathcal{A}$ to solve the CDH problem, *i.e.* to compute abP.

All queries of the adversary $\mathcal{A}$ now pass through $\mathcal{CH}$. $\mathcal{CH}$ chooses master-key $s \in Z_q^*$ and computes $P_0 = sP$. $\mathcal{CH}$ sets the system parameter as params=$(G_1, G_2, e, P, P_0, H_1, H_2, l)$. params and s are sent to $\mathcal{A}$. In this way, the game is initialized.

H_1 *query*: $\mathcal{CH}$ maintains an initially empty list $\mathbf{H_1}$ which contains tuples of the form (ID_i, Q_i). Suppose $\mathcal{A}$ can make at most q_{H_1} times H_1 queries. $\mathcal{CH}$ chooses $I, J \in [1, q_{H_1}]$ at random. On receiving a query $H_1(ID_i)$, the same answer from the list $\mathbf{H_1}$ will be given if the request has been asked before. Otherwise, $\mathcal{CH}$ picks a random $Q_i \in G_1^*$, returns Q_i as the answer and adds (ID_i, Q_i) to $\mathbf{H_1}$.

H_2 *query*: Suppose $\mathcal{A}$ can make at most q_{H_2} times H_2 queries. $\mathcal{CH}$ maintains an initially empty list $\mathbf{H_2}$ which contains tuples of the form $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$. On receiving a query $H_2(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i)$, the same answer from the list $\mathbf{H_2}$ will be given if the request has been asked before. Otherwise, $\mathcal{CH}$ does as follows.

- If there exists a tuple $(\prod_{i,j}^n, r_{i,j}^n, M_{i,j}^n, M_{j,i}^n, P_i^n, P_j^n, SK_{i,j}^n)$ in $\mathbf{S}$ such that
  · $\prod_{i,j}^n$ is an initiator and $ID_i^a = ID_i, ID_i^b = ID_j$, $R_i^a = M_{i,j}^n, R_i^b = M_{j,i}^n$, $X_i = r_{i,j}^n M_{j,i}^n$, $y_i = e(M_{j,i}^n + Q_j, sM_{i,j}^n + sH_1(ID_i))$, $P_i^a = P_i^n, P_i^b = P_j^n$, $Y_i = r_{i,j}^n P_j^n$, $e(Z_i, P) = e(P_i^a, R_i^b)$, $SK_{i,j}^n \neq null$, or
  · $\prod_{i,j}^n$ is a responder and $ID_i^a = ID_j, ID_i^b = ID_i$, $R_i^a = M_{j,i}^n, R_i^b = M_{i,j}^n$, $X_i = r_{i,j}^n M_{j,i}^n$, $y_i = e(M_{j,i}^n + Q_j, sM_{i,j}^n + sH_1(ID_i))$, $P_i^a = P_j^n, P_i^b = P_i^n, e(Y_i, P) = e(P_i^b, R_i^a)$, $Z_i = r_{i,j}^n P_j^n$, $SK_{i,j}^n \neq null$,

  set $h_i = SK_{i,j}^n$, add $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ to $\mathbf{H_2}$ and return h_i as the answer.
- Else, randomly choose $h_i \in \{0,1\}^l$, return h_i as the answer and add $(ID_i^a, ID_i^b, R_i^a, R_i^b, X_i, y_i, P_i^a, P_i^b, Y_i, Z_i, h_i)$ to $\mathbf{H_2}$.

Create(ID_i): Without loss of generality, we assume that all Create queries are distinct. $\mathcal{CH}$ maintains an initially empty list $\mathbf{C}$ of the form (ID_i, x_i, P_i). On receiving a Create query on ID_i, $\mathcal{CH}$ does as follows.

- If $ID_i \neq ID_J$, choose a random $x_i \in Z_q^*$, compute the public key $P_i = x_i P$, and add (ID_i, x_i, P_i) to $\mathbf{C}$.
- Else if $ID_i = ID_J$, set $P_i = aP$ and add $(ID_i, null, P_i)$ to $\mathbf{C}$.

Without loss of generality, we assume that, before making the following queries, $\mathcal{A}$ has already made Create queries on some related identities.

Public-Key(ID_i): On receiving this query, $\mathcal{CH}$ first searches for a tuple (ID_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i, and then returns P_i as the answer.

Corrupt(ID_i): On receiving this query, $\mathcal{CH}$ searches for a tuple (ID_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i. If $ID_i = ID_J$, $\mathcal{CH}$ aborts (Event 1); otherwise, $\mathcal{CH}$ returns $(x_i, D_i = sH_1(ID_i))$ as the answer.

Public-Key-Replacement(ID_i, P'_i): On receiving this query, if $ID_i = ID_J$, $\mathcal{CH}$ aborts (Event 2); otherwise, $\mathcal{CH}$ searches for a tuple (ID_i, x_i, P_i) in $\mathbf{C}$ which is indexed by ID_i, then updates P_i to P'_i and sets $x_i = null$.

Send$(\prod^n_{i,j}, M)$: $\mathcal{CH}$ maintains an initially empty list $\mathbf{S}$ which contains tuples $(\prod^n_{i,j}, r^n_{i,j}, M^n_{i,j}, M^n_{j,i}, P^n_i, P^n_j, SK^n_{i,j})$. On receiving query Send$(\prod^n_{i,j}, M)$ (if $M \neq \lambda$, $\mathcal{CH}$ sets $M^n_{j,i} = M$; otherwise, at the end of the protocol, a message will be returned; if $\prod^n_{i,j}$ is in *Accepted*, then $\mathcal{CH}$ sets this message to be $M^n_{j,i}$), $\mathcal{CH}$ does as follows

- If $n = T$, $ID_i = ID_I$ and $ID_j = ID_J$, set $SK^n_{i,j} = r^n_{i,j} = null$, $M^n_{i,j} = bP$, return $M^n_{i,j}$ as the answer and add $(\prod^n_{i,j}, r^n_{i,j}, M^n_{i,j}, M^n_{j,i}, P^n_i, P^n_j, SK^n_{i,j})$ to $\mathbf{S}$.
- Otherwise, choose a random $r^n_{i,j} \in Z^*_q$, compute $M^n_{i,j} = r^n_{i,j}P$, return $M^n_{i,j}$ as answer, set $SK^n_{i,j} = null$ and add $(\prod^n_{i,j}, r^n_{i,j}, M^n_{i,j}, M^n_{j,i}, P^n_i, P^n_j, SK^n_{i,j})$ to $\mathbf{S}$.

Reveal$(\prod^n_{i,j})$: On receiving this query, $\mathcal{CH}$ looks up the list $\mathbf{S}$ for a tuple $(\prod^n_{i,j}, r^n_{i,j}, M^n_{i,j}, M^n_{j,i}, P^n_i, P^n_j, SK^n_{i,j})$. If $SK^n_{i,j} \neq null$, $\mathcal{CH}$ returns $SK^n_{i,j}$ as the answer. Otherwise, $\mathcal{CH}$ does as follows.

- Abort if $n = T$, $ID_i = ID_I$ and $ID_j = ID_J$, or $\prod^n_{i,j}$ is the oracle who has a matching conversation with $\prod^T_{I,J}$ (Event 3).
- Else if there exists a tuple $(ID^a_i, ID^b_i, R^a_i, R^b_i, X_i, y_i, P^a_i, P^b_i, Y_i, Z_i, h_i)$ in $\mathbf{H_2}$ such that
  · $\prod^n_{i,j}$ is an initiator and $ID_i = ID^a_i, ID_j = ID^b_i$, $M^n_{i,j} = R^a_i, M^n_{j,i} = R^b_i$, $r^n_{i,j}M^n_{j,i} = X_i$, $e(M^n_{j,i} + Q_j, sM^n_{i,j} + sH_1(ID_i)) = y_i, P^n_i = P^a_i, P^n_j = P^b_i$, $r^n_{i,j}P^n_j = Y_i$, $e(P^a_i, R^b_i) = e(Z_i, P)$, or
  · $\prod^n_{i,j}$ is a responder and $ID_j = ID^a_i, ID_i = ID^b_i$, $M^n_{j,i} = R^a_i, M^n_{i,j} = R^b_i$, $r^n_{i,j}M^n_{j,i} = X_i$, $e(M^n_{j,i} + Q_j, sM^n_{i,j} + sH_1(ID_i)) = y_i$, $P^n_j = P^a_i, P^n_i = P^b_i$, $e(P^b_i, R^a_i) = e(Y_i, P)$, $r^n_{i,j}P^n_j = Z_i$,

  set $SK^n_{i,j} = h_i$ and return $SK^n_{i,j}$ as the answer.
- Else, randomly sample $SK^n_{i,j} \in \{0,1\}^l$ and return $SK^n_{i,j}$ as the answer.

Test$(\prod^T_{I,J})$: At some point, $\mathcal{A}$ will ask a Test query to some oracle. If $\mathcal{A}$ does not choose one of the oracles $\prod^T_{I,J}$ to ask the test query, then $\mathcal{CH}$ aborts (Event 4). Else if $\mathcal{A}$ does pick $\prod^T_{I,J}$ for the Test query, $\mathcal{CH}$ simply outputs a random value in $\{0,1\}^l$.

Once $\mathcal{A}$ finishes the queries and returns her guess bit (a tuple $(\prod_{I,J}^T, null, cP, M_{J,I}^T, P_I^T, P_J^T, SK_{I,J}^T)$ must be in $\mathbf{S}$), $\mathcal{CH}$ proceeds as follows. For every tuple in $\mathbf{H_2}$ which is indexed by $(R_i^a, R_i^b, P_i^a, P_i^b, Y_i^b, Z_i)$, if $\prod_{I,J}^T$ is an initiator oracle and there is a tuple in $\mathbf{H_2}$ such that $P_i^b = aP, R_i^a = bP$, $\mathcal{CH}$ randomly chooses Y_j from $\mathbf{H_2}$ and returns Y_j as the response to the CDH challenge; otherwise $\prod_{I,J}^T$ is a responder and there is a tuple in $\mathbf{H_2}$ such that $P_i^a = aP, R_i^b = bP$, so $\mathcal{CH}$ randomly chooses Z_j from $\mathbf{H_2}$ and returns Z_j as the response to the CDH challenge. If there exists no such tuple in $\mathbf{H_2}$, $\mathcal{CH}$ aborts (Event 5).

Claim 3 *If $\mathcal{CH}$ does not abort the game, $\mathcal{A}$ cannot find any inconsistency between the simulation and the real world.*

Proof. The simulations of all the random oracles are valid and the messages of the oracles are uniformly distributed in the message space. So the adversary cannot notice any difference with the real world. □

Claim 4 *Let* Even 6 *be that abP was not queried on H_2. Then* $\Pr[\overline{\text{Event 5}} \wedge \overline{\text{Event 6}}] \geq Advantage^{\mathcal{A}}(k)$.

Proof. The analysis is similar to that of Claim 2. We omit it here to avoid repetition. □

Let Event 7 be the event that, in the attack, adversary $\mathcal{A}$ chooses oracle $\prod_{I,J}^T$ as the challenger oracle. It is clear that Events 1, 2, 3, 4 will not happen if Event 7 happens and vice versa. Therefore,

$$\Pr[\overline{\text{Event 1} \vee \text{Event 2} \vee \text{Event 3} \vee \text{Event 4}}] = \Pr[\text{Event 7}] \geq \frac{1}{q_c^2 \cdot q_s}$$

Let Event 8 be the event that $\mathcal{A}$ does not abort in the game, and Event 9 be the event that $\mathcal{CH}$ found the correct solution abP of the CDH problem. Accordingly, we have

$$\begin{aligned}\Pr[\mathcal{CH}\ wins] &= \Pr[\text{Event 8} \wedge \overline{\text{Event 6}} \wedge \text{Event 9}] \\ &= \Pr[\text{Event 7} \wedge \overline{\text{Event 5}} \wedge \overline{\text{Event 6}} \wedge \text{Event 9}] \\ &\geq \frac{1}{q_c^2 \cdot q_s \cdot q_{H_2}} \Pr[\overline{\text{Event 5}} \wedge \overline{\text{Event 6}}] \\ &\geq \frac{1}{q_c^2 \cdot q_s \cdot q_{H_2}} Advantage^{\mathcal{A}}(k).\end{aligned}$$

□

Theorem 1 *Our protocol is a secure certificateless two-party AKA protocol.*

Proof. The theorem follows directly from Lemmas 1, 2 and 3. □

Theorem 2 *Our protocol has the perfect forward secrecy property if the CDH problem in G_1 is hard.*

Proof. Suppose that A and B established a session key K using our certificateless two-party AKA protocol, and later, their private keys S_A and S_B were compromised. Let r_A and r_B be the secret random numbers used by A and B, respectively, during the establishment of their session key. It is easy to see that, to compute the established session key K, the adversary who owns S_A, S_B, $R_A = r_A P$ and $R_B = r_B P$ for unknown r_A, r_B must know the value of $r_A r_B P$. However, to compute the value of $r_A r_B P$ without the knowledge of either r_A or r_B, the adversary must have the ability to solve the CDH problem in G_1. Under the CDH assumption, this probability is negligible. Hence, our protocol has the *perfect forward secrecy* property. $\square$

5 Conclusion

Certificateless two-party AKA protocols are important tools to secure communications over open networks. We have presented a formal definition of the security model for two-party AKA protocols in the certificateless cryptographic setting. Following this definition, we have instantiated an efficient certificateless two-party AKA protocol. The proposed protocol has been proven secure under some well-studied assumptions. Our scheme is also efficient and practical, because each party needs only one pairing operation and five multiplications to negotiate a session key.

Acknowledgments and disclaimer

This work is supported by the Spanish Government through projects TSI2007-65406-C03-01 "E-AEGIS" and CONSOLIDER INGENIO 2010 CSD2007-00004 "ARES", and by the Government of Catalonia under grant 2009 SGR 1135, and by the Chinese NSF projects 60673070, 60970114, 60970115 and 60970116 and the Natural Science Foundation of Jiangsu province (No. BK2006217) of China. The fourth author is partly supported as an ICREA Acadèmia researcher by the Government of Catalonia. The views of the author with the UNESCO Chair in Data Privacy do not necessarily reflect the position of UNESCO nor commit that organization.

References

[1] S.S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, in: Proceedings of the ASIACRYPT 2003, LNCS, vol. 2894, Springer-Verlag, 2003, pp. 452-473.

[2] M. Bellare, P. Rogaway, Entity authentication and key distribution, in: Proceedings of the CRYPTO 1993, LNCS, vol. 773, Springer-Verlag, 1994, pp. 232-249.

[3] M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in: Proceedings of the ACM CCCS 1993, ACM, 1993, pp. 62-73.

[4] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in: Proceedings of the EUROCRYPT 2000, LNCS, vol. 1807, Springer-Verlag, 2000, pp. 139-155.

[5] S. Blake-Wilson, D. Johason, A. Menezes, Key agreement protocols and their security analysis, in: Proceedings of the Crytography and Coding 1997, LNCS, vol. 1355, Springer-Verlag, 1997, pp. 30-45.

[6] D. Boneh, F. Franklin, Identity-based encryption from the Weil pairing, in: Proceedings of the CRYPTO 2001, LNCS, vol.2139, Springer-Verlag, 2001, pp. 213-229.

[7] S. Chang, D.S. Wong, Y. Mu, Z. Zhang, Certificateless threshold ring signature, Information Sciences (2009), `doi:10.1016/j.ins.2009.06.017`.

[8] L. Chen, Z. Cheng, N. Smart, Identity-based key agreement protocols from pairings, International Journal of Information Security, 6(4) (2007) 213-241.

[9] L. Chen, C. Kudla, Identity based authenticated key agreement from pairings, Cryptology ePrint Archive, Report 2002/184, 2002. `http://eprint.iacr.org/2002/184`.

[10] Z. Cheng, M. Nistazakis, R. Comley, L. Vasiu, On the indistinguishability-based security model of key agreement protocols-simple cases, Cryptology ePrint Archive, Report 2005/129. `http://eprint.iacr.org/2005/129`.

[11] W. Diffie, M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory, 22(6) (1976) 644-654.

[12] S. Duan, Certificateless undeniable signature scheme, Information Sciences, 178(3) (2008) 742-755.

[13] L. Harn, J. Ren, C. Lin, Design of DL-based certificateless digital signatures, Journal of Systems and Software, 82(5) (2009) 789-793.

[14] X. Huang, Y. Mu, W. Susilo, D.S. Wong, W. Wu, Certificateless signature revisited, in: Proceedings of the ACISP 2007,LNCS, vol. 4586, Springer-Verlag, 2007, pp. 308-322.

[15] C. Kudla, K. Paterson, Modular security proofs for key agreement protocols, in: Proceedings of the ASIACRYPT 2005, LNCS, vol. 3788, Springer-Verlag, 2005, pp. 549-565.

[16] M. Luo, Y. Wen, H. Zhao, An enhanced authentication and key agreement mechanism for SIP using certificateless public-key cryptography, in: Proceedings of the IEEE ICYCS 2008, IEEE, 2008, pp. 1577-1582.

[17] T. Mandt, C. Tan, Certificateless authenticated two-party key agreement protocols, in: Proceedings of the ASIAN 2006, LNCS, vol. 4435, Springer-Verlag, 2008, pp. 37-44.

[18] N. McCullagh, P. Barreto, A new two-Party identity-based authenticated key agreement, in: Proceedings of the CT-RSA 2005, LNCS, vol. 3376, Springer-Verlag, 2005, pp. 262-274.

[19] Y. Mu, W. Susilo, Identity-based instantaneous broadcast system in mobile ad-hoc networks, in: Proceedings of the 2004 International Workshop on Mobile Systems, E-commerce and Agent Technology, 2004, pp. 35-40.

[20] T. Okamoto, D. Pointcheval, The gap-problems: A new class of problems for the security of cryptographic schemes, in: Proceedings of the PKC 2001, LNCS, vol. 1992, Springer-Verlag, 2001, pp. 104-118.

[21] A. Shamir, Identity based cryptosystems and signature schemes, in: Proceedings of the CRYPTO 1984, LNCS, vol. 196, Springer-Verlag, 1984, pp. 47-53.

[22] Y. Shi, J. Li, Two-party authenticated key agreement in certificateless public key cryptography, Wuhan University Journal of Natural Sciences, 12(1) (2007) 71-74.

[23] K. Shim, Efficient ID-based authenticated key agreement protocol based on the Weil pairing, Electronics Letters, 39(8) (2003) 653-654.

[24] K. Shim, Breaking the short certificateless signature scheme, Information Sciences, 179(3) (2009) 303-306.

[25] N. Smart, An identity based authenticated key agreement protocol based on the Weil pairing, Electronics Letters, 38(13) (2002) 630-632.

[26] F. Wang, Y. Zhang, A new provably secure authentication and key agreement mechanism for SIP using certificateless public-key cryptography, Computer Communications, 31(10) (2008) 2142-2149.

[27] S. Wang, Z. Cao, X. Dong, Certificateless authenticated key agreement based on the MTI/CO protocol, Journal of Information and Computational Science, 3 (2006) 575-581.

[28] Q. Wu, Y. Mu, W. Susilo, B. Qin, J. Domingo-Ferrer, Asymmetric Group Key Agreement, in: Proceedings of the EUROCRYPT 2009, LNCS, vol. 5479, Springer-Verlag, 2009, pp. 153-170.

[29] Q. Yuan, S. Li, A new efficient ID-based authenticated key agreement protocol, Cryptology ePrint Archive, Report 2005/309. `http://eprint.iacr.org/2005/309`.

[30] L. Zhang, F. Zhang, A new certificateless aggregate signature scheme, Computer Communications, 32(6) (2009) 1079-1085.