On The Indistinguishability-Based Security Model of Key Agreement Protocols–Simple Cases[1]

Zhaohui Cheng, Manos Nistazakis, Richard Comley and Luminita Vasiu

School of Computing Science, Middlesex University
White Hart Lane, London N17 8HR, United Kingdom
{m.z.cheng,e.nistazakis,r.comley,l.vasiu}@mdx.ac.uk

Abstract. Since Bellare and Rogaway's work in 1994, the indistinguishability-based security models of authenticated key agreement protocols in simple cases have been evolving for more than ten years. In this paper, we review and organize the models under a unified framework with some new extensions. By providing a new ability (the *Coin* query) to adversaries and redefining two key security notions, the framework fully exploits an adversary's capacity and can be used to prove all the commonly required security attributes of key agreement protocols with key confirmation. At the same time, the *Coin* query is also used to define a model which can be used to heuristically evaluate the security of a large category of authenticated protocols without key confirmation. We use the models to analyze a few identity-based authenticated key agreement protocols with pairings.

1 Introduction

1.1 The Key Agreement Protocol and Its Security Properties

Key Agreement Protocols (KAP) are the mechanisms by which two or more parties can establish an agreed secret key over a network controlled by adversaries. Normally the established key varies on each execution (session) of the protocol. If in a protocol one party is assured that no other party aside from the specifically identified party (or parties) may gain access to the particular established secret key, then the key agreement protocol is said to provide key authentication. A key agreement protocol which provides mutual key authentication between (or among) parties is called an *Authenticated Key agreement* (AK). Although an AK provides key authentication, one party is not sure whether the other party (or parties) actually has possession of the established secret; otherwise, the protocol is said to provide key confirmation. If a key agreement protocol holds both key authentication and key confirmation, it is called an *Authenticated Key agreement with key Confirmation* (AKC) [40].

A number of security properties are generally believed to be necessary (or good) for an AK or AKC [16][17].

1. *Known session key security.* Each execution of the protocol should result in a unique secret session key. The compromise of one session key should not compromise the keys established in other sessions (e.g., parallel sessions, previous sessions and future sessions).
2. *Forward secrecy.* If the long-term private keys of one or more parties are compromised, the secrecy of previously established session keys should not be affected. We say that a

[1] The work first appeared on ACNS 2004. This is the significantly revised full version combined with an unpublished manuscript "Revisit The Indistinguishability-Based Model of Key Agreement Protocol" written in February 2004 [21].

protocol has *partial forward secrecy* if one or more but not all of the parties' long-term keys can be corrupted without compromising previously established session keys, and we say that a protocol has *perfect forward secrecy* (PFS) if the long-term keys of all the parties involved may be corrupted without compromising any session key previously established by these parties. In the identity-based cryptosystems, PFS becomes more complex. In the normal identity-based cryptosystems, the Key Generation Center's (KGC) long-term private key can be used to compute any party's long-term private key in the system. There is a (perhaps) stronger notion of forward secrecy in these identity-based systems, called *KGC forward secrecy*. Specifically, the KGC's long-term private key may be corrupted (and hence all entities' long-term private keys), but previously established session keys by any parties should not be compromised.

3. *Known session-specific temporary information security.* Many protocols use some random private information which is varying on each execution of the protocol as an input of the session key generation function. The exposure of this private temporary information should not compromise the secrecy of (other) generated session key.
4. *Key-compromise impersonation resilience.* The compromise of party A's long-term private key (keys) will allow an adversary to impersonate A, but it should not enable the adversary to impersonate other parties to A.
5. *Unknown key-share resilience.* Party A should not be able to be coerced into sharing a key with party C when in fact A thinks that he is sharing the key with some party B.

Designing key agreement protocols has a long history, however, many protocols were designed in *ad hoc* modes. Proposed protocols were often broken, sometimes years after they were first presented. To address this problem, some methodologies have been developed to check or prove the security of protocols.

1.2 The Modelling History of Key Agreement Protocol

In 1993, using the indistinguishability notion (see Definition 2) which requires that an adversary cannot differentiate the agreed key from a random sample generated from the agreed key's distribution in polynomial time, Bellare and Rogaway [14] for the first time formalized the model (from now on, we will refer to it as the B-R model) of symmetric key based two-party authenticated key agreement problem by borrowing the notion "matching protocol runs" from [28]. Notion "matching protocol runs" extends the idea of "matching histories" brought forth in [10]. Based on the B-R model, some extensions were made to formalize the other authenticated key agreement protocols, including the two-party asymmetric key based KAP [17], two-party KAP with a trusted server [15], KAP with extra security properties, e.g., forward secrecy [11], anti-dictionary attack [11], smart card model [50] and AK with confirmation [16]. Moreover, a fix [12] to the model was proposed in 1995, after Rackoff had found a security defect of the model [23].

It is also Bellare *et al.* who first proposed another KAP model [6] using the simulation paradigm in 1998. Later, Shoup [44] presented a different model based on simulation. Canetti and Krawczyk readdressed Bellare *et al.*'s simulation model and provided the security channel in [23]. A more general paradigm (the so called universally composable security) can be found in [24][31]. The basic rationale of the simulation-based model is first to design a simple protocol which is secure in an environment where the adversary's capability is restricted, then use an authenticator to compile the simple protocol to a complex one. It is required that a sound authenticator should guarantee that the view of any normal adversary attacking the compiled protocol is computationally indistinguishable from the view of the corresponding capability-restricted adversary (a simulator, which must exist) attacking the secure simple protocol (before compilation). This is a modular approach to design protocols as the simple protocol (which is much easy to design and prove its security) and an authenticator can be designed separately. The advantage of this methodology is its ease of designing and proving a protocol. However, not all the protocols are (can) be designed in this way, such as many AK's.

All these models treat the simple cases of KAP in which there are a few parties involved, usually two or three parties (some simulation-based models can deal with general protocols). Some group KAP modelling practices can be found in [7][8][43]. A different line of work, using a "logic-based" approach to "reason" that an authentication protocol is correct, was pioneered by Burrows, Abadi and Needham [3]. Although this idea is useful and easy to apply, it has a serious defect: a correctness proof does not guarantee that the protocol is secure. Abadi and Rogaway tried to combine the advantages of "logic reasoning" and "provable indistinguishability model" [2]. In this paper, we only discuss the indistinguishability-based models of simple key agreement protocols.

Definition 1 *Function $\epsilon(k)$ is negligible if for every constant $c \geq 0$, there exists k_0, such that for all $k > k_0$, it holds that $\epsilon(k) < k^{-c}$.*

Definition 2 *Let X_n, Y_n be two distributions on $\{0,1\}^n$. We say that X_n and Y_n are computationally indistinguishable if for all polynomial-time machine(PTM) A and polynomial Q, there exists n_0, such that, for all $n > n_0$, it holds that*

$$|Pr_{t \in X_n}(A(t) = 1) - Pr_{t \in Y_n}(A(t) = 1)| = \epsilon(n) < \frac{1}{Q(n)}.$$

Indistinguishability captures the notion "effective similarity", that is it is infeasible to differentiate two objects with tiny difference, in polynomial time.

2 The Indistinguishability-Based Models of Simple Key Agreement Protocol

2.1 The Adversary Model of Key Agreement Protocol

The General Key Agreement Protocol Model. Every party in a key agreement protocol can be described by an algorithm which given the required inputs produces outputs. The inputs could include the system parameters, incoming messages, long-term secrets, short-term random flips, etc. The outputs can be the outgoing messages, protocol conclusions, established secrets, etc. A basic requirement of a key agreement is to guarantee that if an adversary without knowing some (long-term or ephemeral) private information of inputs, based on some primitive assumptions, e.g., the hardness of Diffie-Hellman problem, the existence of collision-free hash function, etc., it is infeasible for the adversary to compute the established secret of a session. To simulate the ability of an adversary which fully controls the network, an oracle model [14] was built by Bellare and Rogaway to allow an adversary to control the inputs of the protocol algorithm. The B-R model originally was designed to formalize the authenticated two-party key agreement with entity authentication based on symmetric keys. Note that an authenticated key agreement with entity authentication is slightly different from an AKC in which each party is assured that the peer party has computed instead of being able to compute the agreed key. Even more, in the B-R model, an adversary can only control two inputs of the protocol algorithm, i.e., the long-term secrets and the incoming messages. We use Blake-Wilson *et al.*'s definitions of AK and AKC [16] as a framework and extend the B-R model to give an adversary the capability to fully control the inputs of a protocol algorithm except for the system parameters which are fixed in advance. By redefining two key notions: *fresh oracle* and *no-matching*, the framework not only can address the basic requirements of AKC and AK, but also covers all the desired security attributes.

Symmetric Key Based Two-Party KAP

• The B-R Protocol Model

In the model [14], each party involved in a session (run) of a protocol is treated as an oracle. An adversary can access the oracle by issuing the allowed queries. An oracle $\Pi^s_{i,j}$ denotes an instance s of party i involved with a partner party j in a session where the instance of party j is $\Pi^t_{j,i}$. The oracle $\Pi^s_{i,j}$ given an input message executes the prescribed protocol Π and produces the output as $\Pi(1^k, i, j, K_i, conv^s_{i,j}, r^s_{i,j}, x) = (m, \delta^s_{i,j}, \sigma^s_{i,j})$.

- 1^k – the security parameter, $k \in N$.
- i – the identity of the sender, $i \in I \subseteq \{0,1\}^k$.
- j – the identity of the intended partner, $j \in I \subseteq \{0,1\}^k$.
- K_i – the secret information of the sender i, $K_i \in \{0,1\}^k$.
- $conv^s_{i,j}$ – the conversation transcript of the session s so far. $conv^s_{i,j} \in \{0,1\}^+ \cup \{*\}$. $\{*\}$ denotes an empty string and $\{0,1\}^+$ denotes a string of polynomial length of k.
- $r^s_{i,j}$ – the random coin flips of the sender i in the session s with partner j. $r \in \{0,1\}^+$.
- x – the current input message in the session. $x \in \{0,1\}^+ \cup \{\lambda\}$. λ denotes an empty message.
- m – the next message to send out in the session. $m \in \{0,1\}^+ \cup \{*\}$.
- $\delta^s_{i,j}$ – the decision in the session s. $\delta \in \{A, R, *\}$, A for accept, R for reject, $*$ for no decision.
- $\sigma^s_{i,j}$ – the private output of the session s (the session key), $\sigma \in \{0,1\}^+ \cup \{*\}$.

And the conversation transcript $conv^s_{i,j}$ is updated as $conv^s_{i,j}.x.m$ ("$x.y$" denotes the result of the concatenation of x and y).

•Adversary Model

In the model, to simulate an adversary's manipulation of inputs to the protocol algorithm and possible knowledge of the established secrets in history, an adversary is allowed to issue the following queries:

1. Send a message query: $SendP(\Pi^s_{i,j}, x)$. $\Pi^s_{i,j}$ executes $\Pi(1^k, i, j, K_i, conv^s_{i,j}, r^s_{i,j}, x)$ and responds with m and $\delta^s_{i,j}$. If the oracle $\Pi^s_{i,j}$ does not exist, it will be created. Note that x can

be λ in the query which causes an oracle to be generated as an initiator (who sends the first message in a session); otherwise, $\Pi^s_{i,j}$ is a responder oracle. This query simulates the adversary's impersonation ability (we consider that the message modification, replay and injection are all impersonation attacks).

2. Reveal a session's agreed session key: $Reveal(\Pi^s_{i,j})$. $\Pi^s_{i,j}$ reveals the session's private output $\sigma^s_{i,j}$ if it exists. This query simulates the exposure of some session key.
3. Corrupt a party: $Corrupt(i, K)$. The party i responds with K_i and updates $K_i = K$ if $K \neq \lambda$. This query allows the adversary to be internal attackers.
4. Reveal a session's random coins: $Coin(\Pi^s_{i,j}, r)$. Oracle $\Pi^s_{i,j}$ replies $r^s_{i,j}$. If $r \neq \lambda$, the oracle will use r as the random flips $r^s_{i,j}$ and in this case the query should be issued before the oracle generates the messages for the session.

To define the security, an adversary can take another action as one of the following:

1. Test a session's agreed session key: $TestD(\Pi^s_{i,j})$. Oracle $\Pi^s_{i,j}$, as a challenger, flips a fair coin $b \stackrel{R}{\leftarrow} \{0, 1\}$ and answers with $\sigma^s_{i,j}$ if $b = 0$; otherwise it returns a random sample generated according to the distribution of the session secret $\sigma^s_{i,j}$.
2. Guess a session's agreed session key: $TestC(\Pi^s_{i,j})$. The adversary chooses oracle $\Pi^s_{i,j}$ as the challenger and guesses the oracle's private output $\sigma^s_{i,j}$.

Query $Coin(\Pi^s_{i,j}, r)$ in the above adversary model is a new capability introduced to an adversary here. With this extra query, an adversary can now fully control the inputs of a protocol algorithm except for the system parameters which are fixed values. We use this query to address the known session-specific temporary information security notion. Many protocols use some random private information which is varying on each session as an input to the session key generation function. The known session-specific temporary information security requires that the exposure of this private temporary information should not compromise the secrecy of the generated session key (or at least the other session keys). The *Coin* query helps us differentiate the protocols that are breakable by asking a *Coin* query to a party from those that are still secure even when the party's random flips of a session are leaked. Note that for some protocols even if the real private key is not exposed, the protocol is still broken, e.g., [33]. This differentiation is important in some scenarios. Many times, key agreement protocols are used in constraint devices. It has proven to be very difficult to generate "real" random strings. Thus in several situations the adversary may be able to predict the session random seeds. Moreover, in some environments, the devices may use out-side random source (such as temperature etc.) as the seed to generate the session random string. The adversary may control such kind of source in these environments. Though it is impossible to defeat this kind of attacks. But it is important to study to what extent the key agreement is still secure under the *Coin* query. Our work provides a first step in this direction. In the model, we do not restrict how many party's coin tosses can be queried in a session which could vary on different protocols.

Another advantage of this query is to enable us to address the security issue in the following scenario. Suppose that Alice's long-term private key has been compromised. However, Alice has not noticed that this has happened, so she continues to use this key to establish new sessions. If the adversary passively eavesdrops the conversations (sometimes, it is possible that the adversary cannot launch an active attack), obviously we prefer to a protocol that is still secure in this setting. We call this *backward secrecy* which is the counter-notion of *forward secrecy*. As a party normally has two pieces of private information at most, i.e., the long-term private key and the ephemeral random coins, if the session key is to be secure, the random coins of the party have to be kept secret when the long-term private key is exposed. In the formulations in the literature so far, there is no formulation to differentiate if the random coins are compromised or not. The *Coin* query addresses this issue. Furthermore, as the backward secrecy and the forward secrecy appear to imply each other, the query can be used to define the forward secrecy of a protocol.

In the simulation-based model [23], the authors introduced the "session-state reveal" query to treat the internal status exposure attack. The *Coin* query and the "session-state reveal" query share some similarity, but also have significant difference between them. Essentially the "session-state reveal" query is not an atomic query. The authors in [23] did not address how the query should be responded (what information should be disclosed). Instead they declared that the response should vary on each protocol. Obviously if the random flips are not erased or not protected, the "session-state reveal" query covers the query $Coin(\Pi^s_{i,j}, \lambda)$. However, there could be some other sensitive information computed during a session as well, such as the statically shared secret (e.g., K' in Protocol 1 in Section 3), a message to be signed, etc. If these types of information are leaked, some protocols could be totally broken. This differs the "session-state reveal" query from the *Coin* query. Another difference between these two queries is that the "session-state reveal" query can only be issued on an *incomplete* session, but we can restrict the time when an adversary can query $Coin(\Pi^s_{i,j}, \lambda)$, e.g., anytime in the middle of a session, or even after the completion of a session to define different security notions. On the other hand, similarly as in [23], how to respond to the $Coin(\Pi^s_{i,j}, \lambda)$ varies on different protocols and even on different implementations. For example, if a protocol uses the Diffie-Hellman scheme with a signature scheme. Both the Diffie-Hellman exchange and the signature scheme need random flips. If the random flips used in the signature and the message being signed are disclosed, the used long-term private key could be recovered in many signature schemes. Hence, if the *Coin* query is considered and the signed message is sent as plain-text over the network, the signature scheme should be implemented by some intrusion resilient device, such as a smart card.

Moreover, $Coin(\Pi^s_{i,j}, r)$ enables an adversary to control the generation of the random flips which cannot be done by the "session-state reveal" query. This feature also differentiates the *Coin* query from the *Access* query in the smart card model [50] in which the random flips are generated by the smart card. If a protocol is proved to be secure in the smart card model including query $Coin(\Pi^s_{i,j}, r)$, the random seeds generation function can be taken outside the smart card. Then the smart card becomes a truly stateless instrument. This query is also used in the AK model to simulate some general attacks and so to evaluate the security of a large category of AK's in Section 4. However, we want to stress that an adversary with the ability to issue this query is so powerful that we should be very cautious to use this formulation and should fully investigate the impact of such attack on the protocol. In fact, as analyzed above many protocols based on signature can be totally broken when this query is allowed, even if the signature scheme is not directly used in the exchanged message. For example, the MQV protocol [36] is broken when the *Coin* query is allowed [37].

We formally introduce the *TestC*[2] query to differentiate the possible variants of computing hardness assumptions on which a protocol can be based. Many protocols use a Diffie-Hellman agreement scheme, but some of them are based on the computing Diffie-Hellman problems (CDH) exactly, while others are based on the decisional Diffie-Hellman problems (DDH). Query *TestC* can be used in the CDH-based protocols, and query *TestD* can be used in the DDH-based protocols. It is necessary to model these two situations because in some scenarios a DDH problem can be solved with an (expected) polynomial-time algorithm, while the corresponding CDH problem is still hard. The problems of this type are the so called gap Diffie-Hellman problems (GDH). It seems that using query *TestD* can define a stronger security notion than *TestC*, although a DDH is a weaker computation complexity basis than the corresponding CDH or at most equal to the CDH. The model using query *TestD* can guarantee that no bit information of a session key generated by running a proved protocol will be leaked in the protocol execution. But a model using *TestC* cannot promise such strong security because it is possible that some bits of the session key can be computed easily while it is still difficult to compute the key as a whole. It is an interesting topic to strengthen a protocol's security which is proved using the *TestC* model. It seems that applying a cryptographic hash function (or a pseudo-random function (PRF)) on the established secret to derive a session key is a promising way. Furthermore, if using the random oracle model (see Remark 2) in a proof, it seems possible to prove a protocol's security using the *TestD* query based on a CDH assumption in some cases. In the literature, the *TestC* query has been used to prove the security of protocols, e.g., [27]. For more information of the CDH and the DDH problems, please refer to [42].

In some cases, an adversary does not tamper with the messages between oracles. We say such adversary *benign* to a session if it faithfully conveys each message from one engaged oracle to the other [14].

Remark 1 *In [11], the authors introduced another query Execute*($\Pi^s_{i,j}, \Pi^t_{j,i}$) *to simulate the benign adversary passively eavesdropping the conversations between* $\Pi^s_{i,j}$ *and* $\Pi^t_{j,i}$. *Although this new query makes the analysis of the protocol's ability of anti-dictionary attack easier, it is not an essential query and can be simulated by a series of SendP queries. Hence, we do not use it as a basic query type.*

Remark 2 Random Oracle Methodology *[13]. To design a scheme we assume that all parties (including the adversary) share a truly random function. The truly random function is interpreted as a program g with an initially empty vector v:*

```
Input: x Ouput: g(x)
    If x is in the vector v (v[x] is not empty),
    then
        return v[x];
    else
        begin
            v[x]=coin tosses;
            return v[x];
        end
```

[2] The model using TestC query is no longer indistinguishability-based.

To construct a real scheme we have to replace the random oracle with a "good cryptographic hashing function", i.e., providing all parties (including the adversary) the description of this function. This is different from the standard methodology. Using the random oracle model, we can design very succinct cryptographic schemes. However, the random oracle model can only provide heuristical evaluation of the security of the designed scheme. "Not only do there exist idealized protocols that have no secure implementations, practically any idealized protocol can be slightly 'tweaked' so that the tweaked protocol remains just as secure in the idealized model, but has no secure implementations." [22]; the latest result on the random oracle methodology can be found in [4]. However, this methodology has been widely used in the literature despite its shortcoming and in this paper, we prove the security of some protocols using the random oracle as well.

Asymmetric Key Based Two-Party KAP

• Protocol Model

The only difference between the symmetric and asymmetric key based two-party key agreements is in the latter, a party uses some asymmetric cryptography which needs a pair of private and public keys (S, P). So, a party should obtain the intended partner party's public key. Oracle $\Pi^s_{i,j}$ executes the prescribed protocol by

$$\Pi(1^k, i, j, SP_i, P_j, conv^s_{i,j}, r^s_{i,j}, x) \ =(m, \delta^s_{i,j}, \sigma^s_{i,j})\,.$$

SP_i – the private and public key pair of party i.

P_j – the public key of the intended party j. This value could vary on each session changed by the queries defined in the adversary model.

other inputs – identical to the corresponding ones in the model of symmetric key based two-party KAP.

•Adversary Model

Apart from being able to ask the same queries (for $Corrupt(i, K)$, the party i replies with S_i and updates SP_i (if $K \neq \lambda$)) as defined in the model of symmetric key based two-party KAP, an adversary E can issue another query:

$$Replace(\Pi^s_{i,j}, P). \quad \text{The oracle } \Pi^s_{i,j} \text{ updates } P_j = P \text{ when } i \neq j.$$

The newly introduced query provides a simulation that a certificate could be generated online and distributed in a message of a session and then accepted by the receiver. The condition $i \neq j$ disallows the adversary to send party i a new certificate issued to i (this attack should never succeed because party i should never accept a faked certificate of itself). By allowing this query, some interesting attacks in the literature can be addressed, especially the source substitution attack (Note 12.54 [40]). Many protocols suffer from this attack, for example the attacks on MTI A(0) in Note 12.54 [40]. Note that the unknown key-share attack on the MQV protocol [32] essentially can be simulated by the *Corrupt* query because the adversary knows the corresponding private key. As there is no way that the oracle could know whether the adversary knows the private key corresponding to the replaced public key, we assume that if $Replace(\Pi^s_{i,j}, P)$ is issued, then j is corrupted for simplicity, even if the adversary does not know the private key in some attacks (this does not affect the query to simulate the attack. One may check that the source substitution attacks on MT1 A(0) are detectable in the model defined in this paper). Hence there are two situations under which party i will be regarded as "*corrupted*". (1) when $Corrupt(i, \cdot)$ is issued; (2) when $Replace(\Pi^t_{j,i}, P)$ is issued for some j, t. (In fact, we can differentiate these two types of corruption to define an even more stringent security notion. It is more natural to treat $Replace(\Pi^t_{j,i}, P)$ as the corruption of i only in the session of $\Pi^t_{j,i}$, while $Corrupt(i, \cdot)$ as the corruption of i in all the following sessions). However in the identity-based protocols analyzed in this paper, this query will not do any harm to the security of the protocols because the public key of a party is exactly the party's identity.

Symmetric Key Based Two-Party KAP with an Online Trusted Server

•Protocol Model [15]

In every session of this type of protocol, apart from the two parties as in a two-party protocol, an online Trusted Server (TS) also engages in the session. The protocol consists of two algorithms (Π, Ψ). An oracle, denoted by $\Psi^u_{i,j}$, is an instance of the TS which is engaging in a session u to help party i and j establish a session secret. $\Psi^u_{i,j}$ executes the prescribed protocol as $\Psi(1^k, i, j, K_i, K_j, conv^u_{i,j}, r^u_{i,j}, x) = m = (m_i, m_j)$.

m_i – the message to party i. $m_i \in \{0,1\}^+ \cup \{*\}$.
m_j – the message to party j. $m_j \in \{0,1\}^+ \cup \{*\}$.
others – same as in the model of symmetric key based two-party KAP.

And the conversation $conv^u_{i,j}$ is updated as $conv^u_{i,j}.x.m$. Other oracles, such as $\Pi^s_{i,j}$, work in the same way as in the model of symmetric key based two-party KAP.

•Adversary Model

Apart from being able to issue the same queries as defined in the model of symmetric key based two-party KAP, an adversary E can ask another type of send query to the TS:

$SendS(\Psi^u_{i,j}, x)$. $\Psi^u_{i,j}$ executes $\Psi(1^k, i, j, K_i, K_j, conv^u_{i,j}, r^u_{i,j}, x)$ and answers with $m = (m_i, m_j)$.

2.2 The Matching Conversations

Every party involved in a session of a protocol has its own conversation transcript. A key agreement protocol requires that parties in the same session would establish the same session secret. To define the notion "same session", "*matching conversations*" is introduced in the model. Based on the relation between two oracles' transcripts, the matching conversations define the condition under which parties have engaged in the same session. Two ways[3] to define the matching conversations are widely used in the literature, i.e., the Turing machine time [14] and the session identity (ID) [11].

[3] Partner function as the third method was defined in [15] which may be more suitable for the two-party protocols with an online trusted server.

Turing Machine Time

Definition 3 *A conversation of oracle $\Pi^s_{i,j}$ is a message sequence $K = (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), \ldots, (\tau_m, \alpha_m, \beta_m)$. This sequence encodes that at time τ_i oracle $\Pi^s_{i,j}$ was asked α_i and responded with β_i. Time τ_{i+1} is later than time τ_i ($\tau_i < \tau_{i+1}$). α_1 can be λ, which means that $\Pi^s_{i,j}$ sends the first message of the session.*

Remark 3 *The time (τ_i) in the definition can be the real time information, or increasing or decreasing ordered sequence numbers included in the protocol messages. If there is no time-related information in the messages, the abstract Turing machine time is used, which reflects the message generating sequence on machines.*

Definition 4 *(**matching conversations**) For an R-move ($R = 2\rho - 1, \rho \geq 1$) protocol Π, two oracles, $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, engage in conversations K and K', respectively. We say*

1. *K' is a matching conversation to $K = (\tau_0, \lambda, \beta_1), (\tau_2, \alpha_1, \beta_2), \ldots, (\tau_{2\rho-4}, \alpha_{\rho-2}, \beta_{\rho-1})$, $(\tau_{2\rho-2}, \alpha_{\rho-1}, \beta_\rho)$, if $K' = (\tau_1, \beta_1, \alpha_1), (\tau_3, \beta_2, \alpha_2), (\tau_5, \beta_3, \alpha_3), \ldots, (\tau_{2\rho-3}, \beta_{\rho-1}, \alpha_{\rho-1})$ and $\tau_i < \tau_{i+1}$, for $0 \leq i < 2\rho - 2$.*
2. *K is a matching conversation to $K' = (\tau_1, \beta_1, \alpha_1), (\tau_3, \beta_2, \alpha_2), (\tau_5, \beta_3, \alpha_3), \ldots, (\tau_{2\rho-3}, \beta_{\rho-1}, \alpha_{\rho-1}), (\tau_{2\rho-1}, \beta_\rho, *)$, if $K = (\tau_0, \lambda, \beta_1), (\tau_2, \alpha_1, \beta_2), \ldots, (\tau_{2\rho-4}, \alpha_{\rho-2}, \beta_{\rho-1}), (\tau_{2\rho-2}, \alpha_{\rho-1}, \beta_\rho)$, and $\tau_i < \tau_{i+1}$, for $0 \leq i < 2\rho - 1$.*

Two oracles have matching conversations if K' is a matching conversation to K and vice versa.

Session ID

Definition 5 *A session ID is the information that can be used to uniquely identify a session by the involved parties. Two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have the matching conversations if both of them derive the same session ID from (each own) conversation transcript using the same method.*

Remark 4 *This definition is highly informal. The key issue of the definition is how to "uniquely" identify a session. As normally each session of a protocol uses some session-variant information, such as random flips implicitly or even a session identity explicitly (by some secure means). These types of information can be used to define a session ID. One suggestion is to use the concatenation of the messages exchanged [11]. However, we should be careful to use this method in case that it is so stringent that some secure protocols cannot meet this definition. For example, for all three-round protocols, the adversary simply intercepts the last message, then the initiator will accept the session but two parties do not have the same session ID. This consideration has been reflected in Definition 4 where that one oracle has a matching conversation to the another does not imply that two oracles have matching conversations.*

In an AKC, to provide key confirmation, a party must show the witness of the agreed secret[4] to convince the peer party to accept the session. Hence, the basic requirement of key confirmation is that when an uncorrupted party accepts that a session is running with an uncorrupted intended party, the intended party must have engaged in the session. Otherwise, it

[4] For AK's with entity authentication, the witness of the party's long-term private key could be shown instead.

indicates that an adversary can fake the witness of the (ephemeral or long-term) agreed secret that the adversary should not know. We use another notion "*no-matching*" to help us evaluate the soundness of this property of an AKC. A sound key confirmation scheme requires that the probability of the following no-matching event is negligible.

Definition 6 *No-matching$^E(k)$ is the event that there exist i, j, s such that an oracle $\Pi^s_{i,j}$ accepted and there is no oracle $\Pi^t_{j,i}$ which has engaged in a matching conversation to $\Pi^s_{i,j}$. Party j is uncorrupted and if party i is corrupted (this is allowed only when party i and j do not share the long-term secret) then it should not have been asked the* Coin *query.*

In [14][16], the definition of no-matching does not require that $\Pi^s_{i,j}$ has not been asked *Reveal* query. There are three possible reasons for this formulation. (1) An oracle can be defined to respond to the *Reveal* query only after it has accepted, instead of any time after the session key is available. (2) No-matching was first defined to address the entity authentication problem in [14]. By integrating an entity authentication scheme into a key exchange scheme, the combined protocol can achieve the commonly desired security properties. It is possible to design a protocol in which the agreed session key has no application in the used entity authentication scheme. Hence, the *Reveal* query would not affect a sound protocol to defeat an adversary with such ability. (Strictly speaking, this type of protocol should not be call an AKC.) (3) It is a prudent practice to derive different keys from the agreed secret for different uses. That is, the used key in the confirmation steps should be different from the generated session key (used in the later communication) and normally one (or more) cryptographic hash function (or PRF) is used to generate these keys. It is commonly assumed that the compromise of one derived key would not affect the security of another because of the security property of the used hash function (or PRF). Hence, revealing the session key would not help the adversary to deceit an oracle into accepting. Here, we keep this strict requirement.

Furthermore, the definition in [14][16] requires that both party i and j are uncorrupted (and the *Coin* query is not allowed), while Definition 6 allows i to be corrupted but not be asked the *Coin* query when i and j do not share the same long-term private key, e.g., in the asymmetric key based protocols. Hence, even when the adversary E knows party i's long-term private key, the only way for E to convince party i to accept a session with an uncorrupted party j is to faithfully convey messages in the session (being a benign adversary). The new definition enables the model to address the key-compromise impersonation resilience attribute which requires that the compromise of party i's long-term private key (keys) should not enable the adversary to impersonate other parties to i. Disallowing the *Coin* query is essential for the soundness of the formulation because an oracle has only two secrets (the long-term private key and the ephemeral random flips). If an adversary obtains both secrets of an oracle in a session, the adversary can simulate any circumstance under which the oracle would accept the session. Note that when j and i do share the same (long-term) private key and query *Coin* is disabled, the definition is identical to the one in [14][16].

2.3 The Security Definition of Key Agreement Protocol

The oracle model simulates the environment in which parties execute a protocol facing a powerful adversary who fully controls the network. The adversary can also be an internal attacker(s) which is a legitimate party with a long-term private key. Furthermore, the adversary can even control a party's random flips generation without completely corrupting the party. The question now is how to define the security notion of a protocol under attacks of an adversary in the model. Obviously, in the model, some oracles' private outputs (the session key) can be known by the adversary via some "trivial" means, e.g., through issuing the *Reveal* query directly or through getting the long-term and ephemeral private information by the *Corrupt* and *Coin* queries. Hence, we should differentiate the oracles that hold a session key about which the adversary should not be able to know from those whose session key is easy to acquire. This can be done by defining a new notion *fresh oracle*. The private output of a fresh oracle is called a fresh session key which should not be known by an adversary if a key agreement protocol is secure. The fresh oracle is defined as follow.

Definition 7 (fresh oracle 1) *An oracle $\Pi^s_{i,j}$ is fresh if (1) $\Pi^s_{i,j}$ has accepted; (2) $\Pi^s_{i,j}$ is unattacked; (3) j is not corrupted; (4) there is no opened oracle $\Pi^t_{j,i}$, which has had a matching conversation to $\Pi^s_{i,j}$.*

The definition of an "*unattacked*" oracle varies from one protocol to the other. It can be the combinations of item 1, 2 and 3 of the following conditions (item 1, 2 are always the mandatary ones):

1. The oracle was not asked $Reveal(\Pi^s_{i,j})$;
2. The oracle was not asked $Coin(\Pi^s_{i,j}, r)$;
3. The oracle was not asked $Corrupt(i, K)$;

And an oracle is "*opened*" if the oracle was asked the *Reveal* query.

A few points need to be stressed. Condition 3 in Definition 7 guarantees that the $Replace(\Pi^s_{i,j}, P)$ query (if allowed) was not issued in the certificate-based protocols. The definition does not require the existence of a partner oracle for a fresh oracle, but requires that the fresh oracle has not been asked the *Coin* query and there is no such requirement on the partner oracle if it exists. Note that the fresh oracle definition in [14][16] requires that i is not corrupted and $\Pi^s_{i,j}$ has not been revealed (and the *Coin* query is forbidden). However, we will show a few asymmetric key based protocols in Section 3 and 4 that i can be corrupted, so to achieve the key-compromise impersonation resilience property.

Now we come to the last step, to evaluate how well a fresh session key is hidden in a protocol. One possible method is to require that the adversary is not able to compute the whole session key. Another method is to apply the indistinguishability notion which was first used to define the security notion of encryption scheme [29]. The basic idea of indistinguishability is to require that given the fresh session key and a random sample generated according to the same distribution of session keys, the adversary cannot pick out the session key with probability non-negligibly greater than 1/2. The indistinguishability methodology defines a stronger security notion than the former one. As widely accepted, the first method is too weak to define the security of encryption, but here we still use it as an alternative definition to the indistinguishability method for some protocols, because the role of an agreed secret in a key agreement protocol is significantly different from the one of plaintexts in an encryption scheme. Normally we can apply a cryptographic hash function (or a PRF) on the agreed secret to derive a new key which is used as the real session key in the later communication. We define three types of adversary. Type-I and Type-II are based on the indistinguishability methodology and Type-III is based on the hardness to compute the whole session key. There are two types of definition using the indistinguishability method because of Rackoff's attack. And in some cases, when using the random oracle in a proof, it is possible to transform the Type-III security to the one of Type-I.

Type-I Adversary (adversary for TestD query): After having asked enough (polynomial times of k) queries, an adversary E chooses a fresh oracle $\Pi^s_{i,j}$ on which it wants to be challenged. It asks a single query $TestD(\Pi^s_{i,j})$. The adversary tries to guess the coin b flipped by the challenger. If the adversary guesses the correct b, we say that it wins.

Type-II Adversary (adaptive adversary for TestD query): As a Type-I adversary, an adaptive adversary E can ask the prescribed queries polynomial times, then it chooses one fresh oracle $\Pi^s_{i,j}$ as the challenger to issue $TestD(\Pi^s_{i,j})$ query. $\Pi^s_{i,j}$ responds in the same way as the challenger to the Type-I adversary. Moreover, the adaptive adversary can continue to ask queries (except *TestD*) polynomial times but keep $\Pi^s_{i,j}$ fresh. If the adversary does not want to ask any more queries, it guesses the bit b. If the adversary guesses the correct b, we say that it wins.

Define

$$Advantage^E(k) = \max\{0, \Pr[E \text{ wins}] - \tfrac{1}{2}\}.$$

Type-III Adversary (adversary for TestC query): After having asked enough (polynomial times of k) queries, an adversary E chooses a fresh oracle $\Pi^s_{i,j}$ on which it wants to be challenged. It asks a single query $TestC(\Pi^s_{i,j})$. The adversary tries to guess the established session key $\sigma^s_{i,j}$. If the adversary guesses the correct $\sigma^s_{i,j}$, we say that it wins.

Define

$$Advantage^E(k) = \Pr[E \text{ wins}].$$

Now we can define the security of AK's and AKC's. We use the definitions in [16] as a framework but define different *fresh oracle* and *no-matching*, so an adversary can fully exploit its capacity.

Definition 8 *Protocol Π is a secure AKC if:*

1. *In the presence of the benign adversary on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, both oracles always accept holding the same session key σ, and this key is distributed uniformly at random on $\{0,1\}^k$;*

and if for every adversary E:

2. *If two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have matching conversations and both i and j are uncorrupted, then both accept and hold the same session key σ;*
3. *The probability of the no-matching event is negligible;*
4. *$Advantage^E(k)$ is negligible.*

We can find that the model is a sound security definition of AKC's covering the desirable attributes. The *known session key security* is addressed by the *Reveal* query. The *known session-specific temporary information security* is covered by the new *Coin* query. As the model uses the *no-matching* of Definition 6, the *key-compromise impersonation resilience* property is automatically guaranteed. As explained in [16], the *unknown key-share resilience* is integrated in the model. The *(perfect) forward (backward) secrecy (PF(B)S)* notion can be defined as follow. First we define an *uncontrolled oracle* as an oracle which has not been asked the *Coin* query.

Definition 9 *A protocol is said to be forward (backward) secure if the adversary wins the game with negligible advantage if it chooses as the challenger an* unopened and uncontrolled oracle $\Pi^s_{i,j}$ *which has a matching conversation to an* unopened and uncontrolled oracle $\Pi^t_{j,i}$ *and both oracles accepted. If both* i *and* j *can be corrupted (only by* $Corrupt(\cdot, \lambda)$*) then the protocol achieves perfect forward (backward) secrecy.*

A point needs to be stressed that a protocol, proved to be an AKC of Definition 8, does not necessarily achieve the forward secrecy, because the partner oracle $\Pi^t_{j,i}$ of a fresh oracle in Definition 7, if existing, could have been issued the *Coin* query.

Note that the special security attributes including the dictionary attack resilience [11] and the smart card implementation [50] can also be integrated in the model. By allowing or forbidding different queries defined in the oracle model and redefining the two key notions, i.e., *fresh oracle* and *no-matching*, the framework is flexible enough to prove the security of AKC protocols with different security properties.

Similarly to the definition of AKC, the security of AK protocols is defined as follow.

Definition 10 *Protocol* Π *is a secure AK if:*

1. *In the presence of the benign adversary on* $\Pi^s_{i,j}$ *and* $\Pi^t_{j,i}$ *, both oracles always accept holding the same session key* σ*, and this key is distributed uniformly at random on* $\{0,1\}^k$*;*

and if for every adversary E*:*

2. *If two oracles* $\Pi^s_{i,j}$ *and* $\Pi^t_{j,i}$ *have matching conversations and both* i *and* j *are uncorrupted, then both accept and hold the same session key* σ*;*
3. $Advantage^E(k)$ *is negligible.*

3 On The Security of Some Paring-Based Key Agreement Protocols

In this section we show a few examples to demonstrate the flexibility of the framework defined in the last section. Before presenting some specific protocols, two primitives (*pairing* and *message authentication code*) employed in the protocols should be described first.

Definition 11 *A pairing is a bilinear map* $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ *with two cyclic groups* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *of prime order* q*, which has the following properties [9]:*

1. *Bilinear: For all* $P, Q \in \mathbb{G}_1$*,* $\hat{e}(sP, tQ) = \hat{e}(P, Q)^{st}$ *for all* $s, t \in \mathbb{Z}_q$*.*
2. *Non-degenerate: For a given point* $P \in \mathbb{G}_1$*,* $\hat{e}(P, Q) = 1_{\mathbb{G}_2}$ *for all* $Q \in \mathbb{G}_1$ *if and only if* $P = 1_{\mathbb{G}_1}$*.*
3. *Computable: There is an efficient algorithm to compute* $\hat{e}(P, Q)$ *for any* $P, Q \in \mathbb{G}_1$*.*

The security of many protocols using the pairing tool is based on the following assumption:

Assumption 1 Bilinear Diffie-Hellman Assumption (BDH) *[9] Let $\mathcal{G}$ be a parameter generator which with system parameters 1^k as input generates two cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q, a generator P of $\mathbb{G}_1$ and a bilinear map $\hat{e}$. We define the advantage of an algorithm $\mathcal{A}$ in solving the problem (given $\langle P, aP, bP, cP\rangle$, to compute $\hat{e}(P,P)^{abc}$) by:*

$$Adv_{\mathcal{G},\mathcal{A}}(k) = Pr[\ \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP) = \hat{e}(P,P)^{abc} | \\ \langle q, \mathbb{G}_1, \mathbb{G}_2, P, \hat{e}\rangle \leftarrow \mathcal{G}(1^k), a, b, c \xleftarrow{R} \mathbb{Z}_q^*].$$

For any randomized polynomial time (in k) algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{G},\mathcal{A}}(k)$ is negligible.

The above assumption implies the following computation Diffie-Hellman assumption.

Assumption 2 Computation Diffie-Hellman Assumption (CDH) *Let $\mathcal{G}$ be a parameter generator which with system parameters 1^k as input generates a cyclic group $\mathbb{G}_1$ of prime order q and a generator P of $\mathbb{G}_1$. We define the advantage of an algorithm $\mathcal{A}$ in solving the problem (given $\langle P, aP, bP\rangle$, to compute abP) by:*

$$Adv_{\mathcal{G},\mathcal{A}}(k) = Pr[\ \mathcal{A}(q, \mathbb{G}_1, P, aP, bP) = abP | \langle q, \mathbb{G}_1, P\rangle \leftarrow \mathcal{G}(1^k), a, b \xleftarrow{R} \mathbb{Z}_q^*].$$

For any randomized polynomial time (in k) algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{G},\mathcal{A}}(k)$ is negligible.

Instead of using the CDH assumption directly, we use the following assumption in the protocols presented in this paper.

Assumption 3 Gap Diffie-Hellman Assumption (GDH) *Let $\mathcal{G}$ be a parameter generator which with system parameters 1^k as input generates a cyclic group $\mathbb{G}_1$ of prime order q and a generator P of $\mathbb{G}_1$. We define the advantage of an algorithm $\mathcal{A}^{\mathcal{F}}$ in solving the problem (given $\langle P, aP, bP\rangle$, to compute abP with the help of $\mathcal{F}$ which can decide if $abP = cP$ given $\langle P, aP, bP, cP\rangle$):*

$$Adv_{\mathcal{G},\mathcal{A}^{\mathcal{F}}}(k) = Pr[\ \mathcal{A}^{\mathcal{F}}(q, \mathbb{G}_1, P, aP, bP) = abP | \langle q, \mathbb{G}_1, P\rangle \leftarrow \mathcal{G}(1^k), a, b \xleftarrow{R} \mathbb{Z}_q^*, \\ 1 \leftarrow \mathcal{F}(P, aP, bP, cP) \text{ if } abP = cP, 0 \leftarrow \mathcal{F}(P, aP, bP, cP) \text{ otherwise}].$$

For any randomized polynomial time (in k) algorithm $\mathcal{A}^{\mathcal{F}}$ with the access to $\mathcal{F}$, the advantage $Adv_{\mathcal{G},\mathcal{A}^{\mathcal{F}}}(k)$ is negligible.

Note that the above definition of GDH is different from the common one. Here, algorithm $\mathcal{F}$ is a deterministic algorithm, instead of a randomized one. Such algorithm can be constructed as follow: if $\hat{e}(P, cP) = \hat{e}(aP, bP)$, $\mathcal{F}$ returns 1; otherwise, $\mathcal{F}$ returns 0.

Definition 12 *([5]) A message authentication code (MAC) is a pair of polynomial algorithms (G, M):*

- *On input 1^k, algorithm G generates a bit string s.*
- *For every s in the range of $G(1^k)$ and for every $m \in \{0,1\}^*$, the deterministic algorithm M computes $\alpha = M(s, m)$. α is called the tag of message m under $M(s, \cdot)$. We shorthand $M(s, \cdot)$ as $M_s(\cdot)$.*

Definition 13 *([5] [30]) For a probabilistic oracle machine F, which has the access to the MAC oracle M_s, we denote by $Q_F^{M_s}(x)$ the set of the queries made by F. A MAC scheme is* secure *if for every probabilistic oracle machine F, the function $\epsilon(k)$ defined by:*

$$\epsilon(k) = Pr[M_s(m) = \alpha \wedge m \notin Q_F^{M_s}(1^k) \text{ where } (s) \leftarrow G(1^k) \text{ and } (m, \alpha) \leftarrow F^{M_s}(1^k)]$$

is negligible, where the probability is taken over the coin tosses of algorithms G and F.

In this paper, algorithm G of a MAC scheme always uniformly chooses s from its range and we shorthand a pair of message and tag $\{m, \mathrm{MAC}_s(m)\}$ as $\mathrm{MAC}_s(m)$. In the protocol description, we use $\mathrm{MAC}_s(m)$ directly for better format and it is easy to remove some redundancy of a message to save bandwidth.

Now we use the formulation to analyze a few pairing-based key agreement protocols. In this paper, all the pairing-based protocols use the same infrastructure. In the system there exists a Key Generation Center (KGC) which with the given security argument 1^k generates the system parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, s, P, P_{pub} = sP, H_1, l, H_2, H_3\rangle$. $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of prime order q. P is the generator of $\mathbb{G}_1$. s is randomly chosen from $\mathbb{Z}_q^*$ as the KGC's private key and sP acts as the public key (P_{pub}) of the center. In most protocols, $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$ and H_2 (and H_3): $\{0,1\}^* \rightarrow \{0,1\}^l$ are cryptographic hash functions. And in the system each party can apply the hash function H_1 on any party I's identity ID_I to find an element $Q_I = H_1(ID_I) \in \mathbb{G}_1^*$. Party I extracts its private key $d_I = sQ_I$ from the KGC. In one exceptional protocol (the MaCullagh-Barreto's protocols [38][39] discussed later), $H_1 : \{0,1\}^* \rightarrow \{0,1\}^q$ and H_2 and H_3 are as usual. In the scheme each party can apply the hash function H_1 on any party I's identity ID_I to find an element $\alpha = H_1(ID_I) \in \{0,1\}^q$. Party I extracts its private key $d_I = (\alpha + s)^{-1}P$ from the KGC. The authors claimed that the security of their protocols can be reduced to the Bilinear Inverse Diffie-Hellman Problem [52]: For $\alpha, \beta \in \{0,1\}^q$, given $\langle P, \alpha P, \beta P\rangle$, to compute $v = \hat{e}(P,P)^{\alpha^{-1}\beta}$ is hard[5].

The ability of Definition 6 can be demonstrated by a pairing-based version of Protocol 1 in [16]. The specification of the pairing-based Protocol 1 is presented in Figure 1 where r_A and r_B are random numbers from $\mathbb{Z}_q^*$ and $K' = H_2(\hat{e}(d_A, Q_B)) = H_2(\hat{e}(Q_A, d_B)) = H_2(\hat{e}(Q_A, Q_B)^s)$. The agreed session key is generated as $SK = H_3(r_A r_B P)$. The security of Protocol 1 in the model used by [16] follows from Theorem 8 of [16] by replacing the assumption of CDH with the assumption of BDH. However, Protocol 1 suffers from the key-compromise impersonation attack.

$$\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{r_A P} & \\
K' = H_2(\hat{e}(d_A, Q_B)) & \xleftarrow{MAC_{K'}(2,B,A,r_B P, r_A P)} & K' = H_2(\hat{e}(Q_A, d_B)) \\
SK = H_3(r_A r_B P) & \xrightarrow{MAC_{K'}(3,A,B,r_A P, r_B P)} & SK = H_3(r_B r_A P) \\
\hline
\end{array}$$

Fig. 1. Protocol 1

If party C knows A's long-term private key d_A, it can impersonate any party I to A by computing

[5] Later, it was pointed out in [20] that the reductions in [38][39] are invalid.

$K' = H_2(\hat{e}(d_A, Q_I))$ used in message 2. On the other hand, the protocol cannot be proved to be secure in the model using Definition 6. The adversary E participates in the game in this way. First, it issues $Corrupt(A, \lambda)$ query to obtain A's long-term private key d_A. Then E asks a query $SendP(\Pi^s_{A,B}, r_B P)$. Once it receives the second message $\mathrm{MAC}_{K'}(2, A, B, r_A P, r_B P)$ from the oracle $\Pi^s_{A,B}$, E asks the last query $SendP(\Pi^s_{A,B}, \mathrm{MAC}_{K'}(3, B, A, r_B P, r_A P))$ where $K' = H_2(\hat{e}(d_A, Q_B))$ is computable by E. Because the last message is valid, $\Pi^s_{A,B}$ will accept the session. However, there is no oracle $\Pi^t_{B,A}$ existing with a matching conversation. Recall that $\Pi^s_{A,B}$ is generated to respond to the first $SendP$ query and $\Pi^t_{B,A}$ as an initiator can only be generated by asking query $SendP(\Pi^t_{B,A}, \lambda)$. Apparently, the conditions of no-matching (Definition 6) are satisfied. However, the condition 3 of Definition 8 (a secure AKC) is breached.

Similarly we can use the new model to analyze the security of AK protocols as well, such as MaCullagh and Barreto's pairing-based key agrement protocol [38] presented in Figure 2. In the

$$
\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{r_A(\beta P + sP)} & \\
K' = H_2(\hat{e}(d_A, r_B(\alpha P + sP))^{r_A}) & \xleftarrow{r_B(\alpha P + sP)} & K' = H_2(\hat{e}(r_A(\beta P + sP), d_B)^{r_B}) \\
\hline
\end{array}
$$

Fig. 2. MaCullagh-Barreto's AK

protocol, r_A and r_B are random numbers from $\mathbb{Z}^*_q$ and party A (resp. party B)'s private key is $d_A = (\alpha + s)^{-1}P$ (resp. $d_B = (\beta + s)^{-1}P$) where $\alpha = H_1(ID_A)$ (resp. $\beta = H_1(ID_B)$). The agreed session key is $K' = H_2(\hat{e}(d_A, r_B(aP + sP))^{r_A}) = H_2(\hat{e}(r_A(bP + sP), d_B)^{r_B}) = H_2(\hat{e}(P, P)^{r_A r_B})$. Again this protocol is vulnerable to the key-compromise impersonation attack [19]. The attack proceeds as in Figure 3. Assume that the adversary E compromised party A's long-term private

$$
\begin{array}{rcl}
A & & E_B \\
\hline
 & \xrightarrow{r_A(\beta P + sP)} & \\
K' = H_2(\hat{e}(d_A, r_B(\beta P + sP))^{r_A}) & \xleftarrow{r_B(\beta P + sP)} & K' = H_2(\hat{e}(r_A(\beta P + sP), d_A)^{r_B}) \\
\hline
\end{array}
$$

Fig. 3. Key-compromise Impersonation Attack on MaCullagh-Barreto's AK

key $d_A = (\alpha + s)^{-1}P$ by issuing the $Corrupt(A, \lambda)$ query. After that, when an oracle $\Pi^s_{A,B}$ sends out the first message, E can simulate the oracle $\Pi^t_{B,A}$ by using the query $SendP(\Pi^s_{A,B}, r_B(\beta P + sP))$ where r_B is a random number chosen by E. Now the adversary can compute the session key $K' = H_2(\hat{e}(d_A, r_B(\beta P + sP))^{r_A}) = H_2(\hat{e}(d_A, r_A(\beta P + sP))^{r_B})$. If E chooses this session as the challenge session, it wins the game with $Advantage^E(k)$ equal to 1/2 (i.e., with the probability equal to 1). The question left now is whether the challenge oracle $\Pi^s_{A,B}$ is fresh according to Definition 7. First, for AK's, every oracle is assumed to accept once it has received the required message in the message space following the protocol. Second, in the attack B is not corrupted (Query $Corrupt(B, \cdot)$ or $Replace(\Pi^s_{A,B}, P)$ has not been issued). Third, there is no oracle $\Pi^t_{B,A}$ (or existing with negligible probability) having a matching conversation with $\Pi^s_{A,B}$ because the message from oracle $\Pi^s_{A,B}$ was generated randomly (by using random r_A) and intercepted by E.

Last, if we define "unattacked oracle" by applying the first two conditions (i.e., not being asked queries *Reveal* and *Coin*), then $\Pi^s_{A,B}$ is "unattacked". So, oracle $\Pi^s_{A,B}$ is a legitimate choice in the challenge phase and the MaCullagh-Barreto's protocol cannot be proved to be secure in the new model. An updated version [39] fixed the identified vulnerability.

We present a few examples to demonstrate the ability of *Coin* query. Scott proposed a pairing-based authenticated key agreement protocol without confirmation [46]. We extend the protocol (in Figure 4) to an AKC version by applying the commonly used MAC technique. In the protocol, the agreed secret is $K = (\hat{e}(Q_A, d_B)^{r_B})^{r_A} = (\hat{e}(d_A, Q_B)^{r_A})^{r_B}$ and $K' = H_2(K)$ and the generated session key is $SK = H_3(K)$. We find that the protocol can be easily broken

$$
\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{\hat{e}(d_A,Q_B)^{r_A}} & \\
K = (\hat{e}(Q_A, d_B)^{r_B})^{r_A} & \xleftarrow{MAC_{K'}(2,B,A,\hat{e}(Q_A,d_B)^{r_B},\hat{e}(d_A,Q_B)^{r_A})} & K = (\hat{e}(d_A, Q_B)^{r_A})^{r_B} \\
K' = H_2(K) & \xrightarrow{MAC_{K'}(3,A,B,\hat{e}(d_A,Q_B)^{r_A},\hat{e}(Q_A,d_B)^{r_B})} & K' = H_2(K) \\
SK = H_3(K) & & SK = H_3(K) \\
\hline
\end{array}
$$

Fig. 4. Protocol 2

if the *Coin* query is allowed. If the adversary E intercepts message (1) and asks *Coin* query to get r_A, it can compute K, so as to obtain the session key SK. Even worse, E can use the obtained information to compute $\hat{e}(d_A, Q_B)$ and so to impersonate A to B or B to A in other sessions. Some other protocols also suffer from this attack, e.g., [33]. Scott's protocol can be broken by a single *Coin* query; while some protocols seem more secure and even if the random flips of both parties are leaked, the session key is still safe, e.g., Protocol 2 in [16] and its pairing-based version. In the pairing-based version (Protocol 3 in Figure 5), the agreed secret is $K = \{r_A r_B P, \hat{e}(Q_A, Q_B)^s\}$ and the derived key used in MAC is $K' = H_2(K)$ and the session key is $SK = H_3(K)$.

$$
\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{r_A P} & \\
K = \{r_A r_B P, \hat{e}(d_A, Q_B)\} & \xleftarrow{MAC_{K'}(2,B,A,r_B P,r_A P)} & K = \{r_B r_A P, \hat{e}(Q_A, d_B)\} \\
K' = H_2(K) & \xrightarrow{MAC_{K'}(3,A,B,r_A P,r_B P)} & K' = H_2(K) \\
SK = H_3(K) & & SK = H_3(K) \\
\hline
\end{array}
$$

Fig. 5. Protocol 3

However, all the previously mentioned protocols cannot achieve the key-compromise impersonation resilience. Now, we show a protocol with many good security properties. Protocol 4 (in Figure 6) is a variant of the AKC proposed by Smart [49] (Protocol 4 using one pairing in each party is faster than the one proposed by Smart which needs two pairings). The agreed secret is $K = \hat{e}(r_A Q_B + r_B Q_A, sP) = \hat{e}(Q_A, sP)^{r_B} \cdot \hat{e}(Q_B, sP)^{r_A}$ and the key used in MAC is $K' = H_2(K)$ and the session key is $SK = H_3(K)$. This protocol has an interesting property:

$$
\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{r_A P} & \\
K = \hat{e}(d_A, r_B P) \cdot \hat{e}(r_A Q_B, sP) & \xleftarrow{MAC_{K'}(2,B,A,r_B P,r_A P)} & K = \hat{e}(r_B Q_A, sP) \cdot \hat{e}(d_B, r_A P) \\
K' = H_2(K) & \xrightarrow{MAC_{K'}(3,A,B,r_A P,r_B P)} & K' = H_2(K) \\
SK = H_3(K) & & SK = H_3(K) \\
\hline
\end{array}
$$

Fig. 6. Protocol 4

even if the random flips of two parties and the agreed secret K are leaked, this compromise will not affect the security of other sessions between the two parties. The adversary gets nothing more than the given two random flips from the attack and K can be recovered from r_A, r_B and the publicly-known system parameters. This protocol also achieves the key-compromise impersonation resilience property. The security of Protocol 6 can be defined by the following theorem (Theorem 1) in which an "unattacked" oracle was not asked the *Reveal* and *Coin* queries.

Theorem 1 *Protocol 4 is a secure AKC protocol against Type-I adversaries, in the model using Definition 6 of no-matching and Definition 7 of fresh oracle and the* $Coin(\Pi^s_{i,j}, r)$ *query is allowed, provided that the BDH assumption is sound and the MAC scheme is secure and* H_1, H_2 *and* H_3 *are random oracles.*

The proof is presented in Appendix. So, Protocol 4 is proved to achieve many desired security properties: known session key security, known session-specific temporary information security, key-compromise impersonation resilience and unknown key-share resilience. As pointed in [47], the protocol does not achieve perfect forward secrecy (recall that a proof of an AKC satisfying Definition 8 does not guarantee that the protocol obtains PFS). However, we can simply modify the computation of K' and SK to achieve this property as follow (Fig. 7). Note that using the

$$
\begin{array}{l}
\hline
K' = H_2(K \| r_A r_B P \| r_A P \| r_B P) \\
SK = H_3(K \| r_A r_B P \| MAC_{K'}(2, B, A, r_B P, r_A P) \| MAC_{K'}(3, A, B, r_A P, r_B P)) \\
\hline
\end{array}
$$

Fig. 7. New Key Generation Function of Protocol 4 with Perfect Forward Secrecy

new key generation method, even when the master key of KGC is exposed, the session key is still secure, i.e., the perfect forward secrecy (including forward secrecy of KGC's master key) is achieved.

Theorem 2 *Protocol 4 with the new key generation function achieves PFS against Type-I adversaries provided that the GDH assumption in* $\mathbb{G}_1$ *is sound and* H_1, H_2 *and* H_3 *are random oracles.*

The proof is presented in Appendix.

4 The Security Definition of AK, Revisited

If the AK definition (Definition 10) uses the fresh oracle defined in Definition 7, as commented in [16], "many protocols do not contain asymmetry in the formation of the agreed key to distinguish which party involved is initiator, and which is the protocol's responder. Such protocols certainly will not meet the security requirement". This means, there exist a large category of AK protocols whose security cannot be proved using the above definition. Put this in another way, the definition is over-rigorous in many cases. In the following part, we elaborate this problem in detail.

4.1 The Security Implications of Fresh Oracle

In the definition of AKC (Definition 8), the conditions 1 and 2 are not related with the *fresh oracle* notion. Condition 3 implies that if an oracle $\Pi^s_{i,j}$ has accepted a conversation with an uncorrupted j, then there must exist (the probability of nonexistence is negligible) an oracle $\Pi^t_{j,i}$ which has had a matching conversation with $\Pi^s_{i,j}$. (The assertion in [16] is even stronger, i.e., "The third says that essentially the only way for any adversary to get an uncorrupted party to accept in a run of the protocol with any other uncorrupted party is by relaying communications like a wire". Note that there is no requirement on whether $\Pi^t_{j,i}$ is unopened). Condition 4 in the model using the *fresh oracle* of Definition 7 requires that at least (1) $\Pi^s_{i,j}$ has accepted; (2) $\Pi^s_{i,j}$ is unopened; (3) $\Pi^s_{i,j}$ is uncontrolled (not being asked the *Coin* query); (4) j is not corrupted (in [14] [16], i is required not to be corrupted as well); (5) $Advantage^E(k)$ is negligible. From the requirements 1, 3 and 4 of *fresh oracle* and condition 3 of AKC ($No\text{-}matching^E(k)$ is negligible), we have that there must exist an oracle $\Pi^t_{j,i}$ which has a matching conversation to a fresh oracle $\Pi^s_{i,j}$. If $\Pi^t_{j,i}$ is opened, then $Advantage^E(k)$ is 1/2. Hence in the model there must exist an unopened $\Pi^t_{j,i}$ which has a matching conversation to $\Pi^s_{i,j}$. Note that proving condition 3 does not need the *fresh oracle* notion, hence if an AKC is proved satisfying condition 3, the prerequisite of *TestD* (or *TestC*) query (the chosen *fresh oracle*) can be redefined as follow:

Definition 14 (fresh oracle 2) *An oracle $\Pi^s_{i,j}$ is fresh if (1) $\Pi^s_{i,j}$ has accepted; (2) $\Pi^s_{i,j}$ is unattacked; (3) j is not corrupted; (4) there is an* unopened *oracle $\Pi^t_{j,i}$, which engaged in a matching conversation to $\Pi^s_{i,j}$.*

Note that condition 3 of AKC (Definition 8) does not necessarily imply condition 4. A secure entity authentication protocol must satisfy condition 3, but not necessarily meet condition 4. We can tweak a security-proved AKC in the model by forcing the initiator of the session to release the generated session key in the last message of a session (after authenticating the responder), which satisfies that the event no-matching happens with negligible probability. However the session key is exposed and the tweaked protocol cannot be regarded as secure.

Obviously, the two fresh oracle definitions are different in the AK model, because in AK's there is no conformation step and hence there is no requirement on $No\text{-}matching^E(k)$. In the model using Definition 7, there is no requirement on whether there exists a peer oracle. But Definition 14 requires that there exits a fresh oracle with a matching conversation.

Definition 7 implies a much stronger security notion than Definition 14 and many protocols fail to meet its requirement. Let us first elaborate what attacks are covered by the AK model using fresh oracle 1 (we refer to it as model 1) but not covered by the AK model using fresh oracle 2 (we refer to it as model 2). This elaboration will help us understand the difficulty in defining the security of AK protocols. We analyze a general protocol (Figure 8) with only two messages (basically the normal AK protocols without confirmation can be transformed into this general protocol). In the protocol, each party has a message generation function (f or g) which takes the ephemeral random flips, the system parameters and possibly the long-term private key as input to generate the outgoing message. After completing the message exchanges, a party uses an agreed key generation function to form the agreed secret key. The agreed key is generated based on the system parameters, the received message, the party's long-term private key and the random flips of the session. The notation used in the following part is summarised in Table 1.

1^k :	the system parameters. The identifers of parties are part of them
$random_I$:	party I's random flips
sk_I :	party I's secret, could be pre-shared secret or private portion of a key pair.
s_I :	the agreed key generation function of party I.
f, g :	the message generation functions.
$[x]$:	the input x is an option.
A, B, E :	party A, B and an adversary E.
$msg_1 := msg_2$:	replacing msg_1 with msg_2.
$A \xrightarrow{msg} B$:	party A sends message msg to party B.

Table 1. Notation

A	B
$\xrightarrow{msg_1 := f(random_A, [sk_A], 1^k)}$	
	$\xleftarrow{msg_2 := g(random_B, [sk_B], 1^k)}$
$s_A(msg_2, random_A, sk_A, 1^k)$	$s_B(msg_1, random_B, sk_B, 1^k)$

Fig. 8. Authenticated Key Agreement

Remark 5 *We use $random_I$ as an input to function s_I instead of msg_I because we want to define a more general abstraction of AK protocols and msg_I can be regenerated in s_I based on the inputs. A responder generates its message not based on the input message. A protocol in which the responder uses some security scheme to bind the received message with an outgoing message is treated as an AK with semi-confirmation whose security can be defined by a variant of the definition of AKC (Definition 8).*

The following are the attacks covered by model 1, but not in model 2.

1. Case 1. An adversary E generates msg_1 or msg_2 (this is one type of general impersonation attack, although maybe E cannot recover the agreed key) or replays a previous valid message.

Note that in some protocols, it is possible that E can generate a valid message. For the replayed message, it is also possible that the random flips for that message have been exposed to E. Two attacks are presented in Figure 9. In this case, if the protocol is proved to be secure in model 1, it means that even if an adversary E knows (only) one party's random flips (if the *Coin* query is allowed), it still cannot break the protocol.

$$
\begin{array}{ll}
E & B \\
\hline
\xrightarrow{msg_1:=f(random_A,1^k)} & \\
 & \xleftarrow{msg_2:=g(random_B,1^k)} \\
s_A(msg_2, random_A, sk_A, 1^k) & s_B(msg_2, random_B, sk_B, 1^k) \\
\hline
\end{array}
$$

$$
\begin{array}{ll}
A & E \\
\hline
\xrightarrow{msg_1:=f(random_A,1^k)} & \\
 & \xleftarrow{msg_2:=g(random_B,1^k)} \\
s_A(msg_2, random_A, sk_A, 1^k) & s_B(msg_2, random_B, sk_B, 1^k) \\
\hline
\end{array}
$$

Fig. 9. Impersonation

2. Case 2. An adversary E modifies one or two messages to attack the protocol which is presented in Figure 10. In this case, E may change a message (or messages) to the one in a particular value set or having some particular property, such as the subgroup attack in [35].

$$
\begin{array}{lccr}
A & & E & B \\
\hline
\left\{\begin{array}{l} \xrightarrow{msg_1:=f(random_A,[sk_A],1^k)} \\ \\ s_A(msg_2, random_A, sk_A, 1^k) \end{array}\right. & & \begin{array}{c} \xrightarrow{msg_1:=msg_1'} \\ \xleftarrow{msg_2} \\ \\ \end{array} & \begin{array}{r} \\ \xleftarrow{msg_2:=g(random_B,[sk_B],1^k)} \\ s_B(msg_1', random_B, sk_B, 1^k) \end{array} \\
\left\{\begin{array}{l} \xrightarrow{msg_1:=f(random_A,[sk_A],1^k)} \\ \\ s_A(msg_2', random_A, sk_A, 1^k) \end{array}\right. & & \begin{array}{c} \xrightarrow{msg_1} \\ \xleftarrow{msg_2:=msg_2'} \\ \\ \end{array} & \begin{array}{r} \\ \xleftarrow{msg_2:=g(random_B,[sk_B],1^k)} \\ s_B(msg_1, random_B, sk_B, 1^k) \end{array} \\
\left\{\begin{array}{l} \xrightarrow{msg_1:=f(random_A,[sk_A],1^k)} \\ \\ s_A(msg_2', random_A, sk_A, 1^k) \end{array}\right. & & \begin{array}{c} \xrightarrow{msg_1:=msg_1'} \\ \xleftarrow{msg_2:=msg_2'} \\ \\ \end{array} & \begin{array}{r} \\ \xleftarrow{msg_2:=g(random_B,[sk_B],1^k)} \\ s_B(msg_1', random_B, sk_B, 1^k) \end{array} \\
\hline
\end{array}
$$

Fig. 10. Modification Attack

3. Case 3. One attack presented in [16] is shown in Figure 11. It is similar to the reflection attack on the entity authentication protocols. We call it the concatenation attack. When party A launches two concurrent sessions s and t with party B, adversary E intercepts the two outgoing messages from A and impersonates B by reflecting back the two intercepted messages cross the sessions (session s's message is sent to session t and vice versa). The at-

$$\begin{array}{lll}
 & A & E \\
\hline
Session_s = & \left\{ \begin{array}{l} \xrightarrow{msg_1^s:=f(random_A^s,[sk_A],1^k)} \\ \\ s_A(msg_1^t, random_A^s, sk_A, 1^k) \end{array} \right. & \xleftarrow{msg_2^s:=msg_1^t} \\
Session_t = & \left\{ \begin{array}{l} \xrightarrow{msg_1^t:=f(random_A^t,[sk_A],1^k)} \\ \\ s_A(msg_1^s, random_A^t, sk_A, 1^k) \end{array} \right. & \xleftarrow{msg_2^t:=msg_1^s} \\
\hline
\end{array}$$

Fig. 11. Concatenation Attack [16]

tack is only feasible when two messages of the protocol are generated by the same function or the initiator's messages can be modified to a responder's valid message without being detected or a party accepts a session with itself. We generalize this attack to remove the above constraint as a general concatenation attack presented in Figure 12. If the two random flips (one is introduced by the received message) are only used in the *general commutative*[6] computations, e.g., for two random flips r_1, r_2, the computations $g^{r_1 r_2}$, $g^{r_1+r_2}$ are commutative on the flips, in the agreed key generation function, then the attack is feasible. Model 1

$$\begin{array}{llll}
 & A & E & B \\
\hline
Session_s = & \left\{ \begin{array}{l} \xrightarrow{msg_1^s:=f(random_A^s,[sk_A],1^k)} \\ \\ s_A(msg_1^t, random_A^s, sk_A, 1^k) \end{array} \right. & \begin{array}{c} \xrightarrow{msg_1^s} \\ \xleftarrow{msg_2^s:=msg_1^t} \\ \\ \end{array} & \begin{array}{r} \\ \xleftarrow{msg_2^s:=g(random_B^s,[sk_B],1^k)} \\ s_B(msg_1^s, random_B^s, sk_B, 1^k) \end{array} \\
Session_t = & \left\{ \begin{array}{l} \\ \xrightarrow{msg_2^t:=f(random_A^t,[sk_A],1^k)} \\ s_A(msg_1^t, random_A^t, sk_A, 1^k) \end{array} \right. & \begin{array}{c} \xleftarrow{msg_1^t} \\ \xrightarrow{msg_2^t:=msg_1^s} \\ \\ \end{array} & \begin{array}{r} \xleftarrow{msg_1^t:=g(random_B^t,[sk_B],1^k)} \\ \\ s_B(msg_1^s, random_B^t, sk_B, 1^k) \end{array} \\
\hline
\end{array}$$

Fig. 12. General Concatenation Attack

is sensitive to this type of attack. For example, in the concatenation attack, an adversary E

[6] A *general-commutative* computation is such an operation that does not "distinguish which party involved is initiator, and which is the protocol's responder" [16].

first asks the *Reveal* query to A for session t and then chooses session s as the challenge. E wins the game with advantage 1/2 (probability 1) because the two sessions have the same agreed key. This attack is feasible in many AK protocols, including all the AK protocols analyzed in this paper and other protocols such as MTI A(0) [41] in Figure 13. In MTI A(0), party I has a private key $x_I \in \mathbb{Z}_q^*$ and a public key $y_I = g^{x_I} \mod p$ where q is a prime factor of $p-1$ for a large prime p and g is a generator of a cyclic subgroup of $\mathbb{Z}_p^*$ with order q. B's agreed key generation function is $s_B(t_A, r_B, x_B, 1^k) = g^{(x_A r_B + x_B r_A)} \mod p$, which is general-commutative on random flips r_A and r_B. Note that the model allows party A to engage in two concurrent sessions with party B and $K_A^1 = K_B^2 = g^{(x_B r_A^1 + x_A r_B^2)} \mod p$ in the attack. Hence the attack is feasible.

$$
\begin{array}{llccl}
 & (x_A \in_R \mathbb{Z}_q^*, y_A = g^{x_A} \, mod \, p)A & & E & B(x_B \in_R \mathbb{Z}_q^*, y_B = g^{x_B} \, mod \, p) \\
\hline
S_1 = \left\{ \begin{array}{l} \\ \\ \\ \end{array} \right. & \begin{array}{l} r_A^1 \in \mathbb{Z}_q^*, t_A^1 = g^{r_A^1} \, mod \, p \\ \\ K_A^1 = y_B^{r_A^1} (t_B^2)^{x_A} \, mod \, p \end{array} & \begin{array}{c} \xrightarrow{t_A^1} \\ \xleftarrow{t_B^2} \\ \\ \end{array} & \begin{array}{c} \xrightarrow{t_A^1} \\ \xleftarrow{t_B^1} \\ \\ \end{array} & \begin{array}{l} \\ r_B^1 \in \mathbb{Z}_q^*, t_B^1 = g^{r_B^1} \, mod \, p, \\ K_B^1 = y_A^{r_B^1} (t_A^1)^{x_B} \, mod \, p \end{array} \\
S_2 = \left\{ \begin{array}{l} \\ \\ \\ \end{array} \right. & \begin{array}{l} \\ r_A^2 \in \mathbb{Z}_q^*, t_A^2 = g^{r_A^2} \, mod \, p \\ K_A^2 = y_B^{r_A^2} (t_B^2)^{x_A} \, mod \, p \end{array} & \begin{array}{c} \xleftarrow{t_B^2} \\ \xrightarrow{t_A^2} \\ \\ \end{array} & \begin{array}{c} \xleftarrow{t_B^2} \\ \xrightarrow{t_A^1} \\ \\ \end{array} & \begin{array}{l} r_B^2 \in \mathbb{Z}_q^*, t_B^2 = g^{r_B^2} \, mod \, p, \\ \\ K_B^2 = y_A^{r_B^2} (t_A^1)^{x_B} \, mod \, p \end{array} \\
\hline
\end{array}
$$

Fig. 13. Attack on MTI A(0)

4.2 Redefining The Security of AK

As commented in [16], the protocols which only use the general-commutative computation on the random flips in the agreed key generation function will not meet the security requirement of model 1 because the attacks in case 3 are detectable in model 1. And as we showed in the above attacks, model 2 certainly is not a sound security definition of AK's because the attacks in case 1 and some attacks in case 2 should be covered in a rigorous model although the attacks in case 3 are not of particular interest in practice.

In [16], the authors did not present an alternative security definition for the AK protocols which use general-commutative computation on random flips although they stated that their definition only needs slight modification to address the issue. We carefully investigate the problem and find that it seems that a slight modification does not work.

The rationale behind formalizing and defining the security of protocols is to test the security strength of a protocol by ignoring (excluding) the possible weaknesses integrated in all protocols. An (over-rigorous) model in which no protocol is secure has no meaning in practice. Hence we first try to remedy model 1 by removing the attacks of case 3 which are feasible in all AK protocols using only general-commutative computation on random flips. We define a new fresh oracle to remove this type of attack.

Definition 15 *Provided the session ID is defined as the concatenation of messages (each unique message is denoted by a single symbol from an infinite symbol set), two oracles have "reverse matching conversations" if one oracle's session ID is the reverse symbol string of the other.*

Definition 16 (fresh oracle 3) *An oracle $\Pi^s_{i,j}$ is fresh if (1) $\Pi^s_{i,j}$ has accepted; (2) $\Pi^s_{i,j}$ is unattacked and j is not corrupted; (3) there is no opened oracle $\Pi^t_{j,i}$, which has had a matching conversation to $\Pi^s_{i,j}$; (4) there is no opened oracle $\Pi^t_{j,i}$, which has had a reverse matching conversation with $\Pi^s_{i,j}$; (5) there is no opened oracle $\Pi^u_{i,j}$, which has had a reverse matching-session with $\Pi^s_{i,j}$.*

Remark 6 *In the above definition, not all the criteria are required for an AK protocol. For example, if an AK protocol's two message generation functions are different then the concatenation attack in Figure 11 is not feasible; hence there is no need of item (5).*

The definition successfully removes the unnecessary consideration of attacks of case 3. However, this definition does not consider some attacks of case 2 which are also of no interest in practice. Let us investigate two specific examples to demonstrate the deficiency of the new definition.

The first example is the modified AK version of Protocol 4 which is slightly different from

$$
\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{r_AP} & \\
K = \hat{e}(d_A, r_BP) \cdot \hat{e}(r_AQ_B, sP) & \xleftarrow{r_BP} & K = \hat{e}(r_BQ_A, sP) \cdot \hat{e}(d_B, r_AP) \\
\hline
\end{array}
$$

Fig. 14. Smart's Protocol [49]

Smart' AK protocol in [49] (the hash function is removed). At the end of the protocol, A computes $K_{AB} = \hat{e}(d_A, r_BP) \cdot \hat{e}(r_AQ_B, sP) = \hat{e}(r_BQ_A + r_AQ_B, P_s)$ and B computes $K_{BA} = \hat{e}(r_BQ_A, sP) \cdot \hat{e}(d_B, r_AP) = \hat{e}(r_BQ_A + r_AQ_B, P_s)$. So the session key is $K = K_{AB} = K_{BA}$ (the computation of the established session key by $SK = H_2(K)$ is removed). This protocol can be demonstrated insecure in the model with the new *fresh oracle* (Definition 16). An adversary E which has the private key d_E can launch the attack presented in Figure 15. After tampering

$$
\begin{array}{ccc}
A & E & B \\
\hline
\xrightarrow{aP} & \xrightarrow{aP+cQ_E} & \\
 & \xleftarrow{bP+cQ_E} & \xleftarrow{bP} \\
K = \hat{e}(bQ_A + aQ_B, P_s) \cdot \hat{e}(sQ_A, cQ_E) & & K = \hat{e}(aQ_B + bQ_A, P_s) \cdot \hat{e}(sQ_B, cQ_E) \\
\hline
\end{array}
$$

Fig. 15. Attack on Smart's Protocol

with the messages, E issues a *Reveal* query to the session in A. Upon getting the result K_{AB}, E chooses the session in B as the challenge oracle. Obviously, the chosen oracle satisfies the conditions of Definition 16, because the session ID in A is $(aP, bP+cQ_E)$, on the other hand the session ID in B is $(bP, aP+cQ_E)$. These two session ID's are neither matching session ID's nor reverse matching session ID's. Hence the challenge oracle is a legitimate choice in the model. E computes $K_{AB} = \hat{e}(Q_A, d_E)^{-c} \cdot \hat{e}(Q_B, d_E)^c$ and responds to the challenge with the result K_{AB}. E wins the game with advantage 1/2 (probability 1). Note that an adversary E without a valid private key can launch a similar attack as well.

One may argue that removing the hash operation affects the security of the protocol. Let us investigate another example in Figure 16, the AK protocol proposed in [25]. At the end of

$$
\begin{array}{rcl}
A & & B \\
\hline
 & \xrightarrow{r_A Q_A} & \\
K = \hat{e}(d_A, r_A Q_B + r_B Q_B) & \xleftarrow{r_B Q_B} & K = \hat{e}(r_A Q_A + r_B Q_A, d_B) \\
\hline
\end{array}
$$

Fig. 16. Chen-Kudla's AK [25]

the protocol, A computes $K_{AB} = \hat{e}(d_A, r_A Q_B + r_B Q_B) = \hat{e}(Q_A, Q_B)^{s(r_A+r_B)}$ and B computes $K_{BA} = \hat{e}(r_A Q_A + r_B Q_A, d_B) = \hat{e}(Q_A, Q_B)^{s(r_A+r_B)}$. Hence, the agreed secret is $K = K_{AB} = K_{BA}$ and the established session key is $SK = H_2(K)$. A similar attack shown in Figure 17 is also feasible to the protocol. Following the same strategy, even with the hash operation H_2, E still can win the game with advantage 1/2 (probability 1). That these attacks are feasible in the model is still because the positions of the random flips are general-commutative.

$$
\begin{array}{ccccc}
A & & E & & B \\
\hline
 & \xrightarrow{aQ_A} & & \xrightarrow{aQ_A+cQ_A} & \\
 & \xleftarrow{bQ_B+cQ_B} & & \xleftarrow{bQ_B} & \\
K = \hat{e}(Q_A, Q_B)^{s(a+b+c)} & & & & K = \hat{e}(Q_A, Q_B)^{s(a+b+c)} \\
\hline
\end{array}
$$

Fig. 17. Attack on Chen-Kudla's AK [34]

Note that many other protocols including MTI A(0) in Figure 13 suffer from this attack. In practice, this attack has no particular interest. We think that a sound model should be able to differentiate serious attacks from this type of attack. Moveover, we do not think that simply assuming that an adversary would not launch this type of attack in the model is a serious formulation because this attack is undetectable in the model without some extra notion defined. It seems that there is no alternative modification on the *fresh oracle* definition to remove the sensitivity to the aforementioned attack.

In [16], the authors proved the security of a protocol by disallowing the adversary to use the *Reveal* query to get around the attack. As the authors suggested in [16], the model without the *Reveal* query cannot guarantee the security of a protocol, especially to counter the known session key attacks. A dumm version of the password-based protocol EKE [11] in which two parties share a password *pwd* shows this point. In the dumm version, instead of applying a hash operation, the protocol proceeds as follow (Figure 18). It appears that the protocol is secure if

$Alice^{pwd}$		Bob^{pwd}
$x \xleftarrow{R} \{1, \ldots, \mid \mathcal{G} \mid\}$	$\xrightarrow{A \| \mathcal{E}_{pwd}(g^x)}$	
	$\xleftarrow{B \| \mathcal{E}_{pwd}(g^y)}$	$y \xleftarrow{R} \{1, \ldots, \mid \mathcal{G} \mid\}$
$K_A = A \| B \| g^x \| g^y \| g^{xy} \| pwd$		$K_B = A \| B \| g^x \| g^y \| g^{xy} \| pwd$

Fig. 18. EKE2

the *Reveal* query is disallowed. However, obviously the protocol is not secure if an established session key is compromised, because the password *pwd* is exposed as well. A more convincing example would be the known session key attack on the tripartite protocol ATK-2 [1] presented by Shim [48].

Here we present a partial solution. Note that the attacks of case 2 can be simulated by one or two attacks of case 1 *in practice*. The two attacks presented above are the third type attack in case 2 which only happens *in theory*. Basically, if adversary E can break the protocol, E does not need to modify both messages in a session, because a party computes the established secret only based on the input message, its random flips and the long-term private secret. Hence even E modifies a party's output message, it will not affect the security of the agreed key computed by that party. And if E only modifies one message, the attack can be simulated by the attacks of case 1. For the attacks of case 1, adversary E can launch a session by impersonating an oracle in an AK even without knowing the private key. What E knows is the random flips at most (if the message is generated by E following the protocol or a replayed message but with the *Coin* query being issued). So we use the newly introduced *Coin* query to enable E to reveal one oracle's random flips of a session. E is even able to control the generation of a party's random flips. We redefine the security of an AK by requiring that in a session, even if one party is corrupted (being asked a *Corrupt* query) and the other party's random flips are generated by the adversary, the adversary still can not differentiate the generated session key from a random number chosen according to the session key distribution. This challenge session can be defined as a *fresh oracle* as in Definition 14. Note that the definition only restricts the behavior of the adversary in the challenge phase to choose a fresh session. Here, we do not restrict how many parties can be asked the *Coin* query in other sessions. This depends on each protocol's security strength, for example, in Scott's AK protocol [46], no *Coin* query should be allowed, while in Smart's AK protocol [49], both parties of a session can be asked the *Coin* query. However, we do regard the situation that the adversary generates a message without going through an oracle (i.e., a message to an oracle is not generated by another oracle[7]) as one *Coin* query has been issued.

If the new simulation is the same as the possible attacks of case 1, then we can regard the adversary of case 1 and 2 *in practice* as an unopened and uncorrupted oracle but with the random flip exposed. Unfortunately for many protocols, E does not need to know the random flips to launch an attack of case 1. For example, in the Chen-Kudla's AK, E can randomly choose

[7] Note that this situation is detectable in the model.

i and send iP, but it does not know the value r such that $rQ_B = iP$. Even more it is likely that such random r does not exist in protocols such as MTI A(0)[8] because g is only the generator of a proper cyclic subgroup of $\mathbb{Z}_p^*$. An attack on another pairing-based protocol vividly demonstrates this point. Shim presented a two-party protocol in [47] (specified in Figure 19). After exchanging

$$\begin{array}{ccc}
A & & B \\
\hline
 & \xrightarrow{T_A = r_A P} & \\
K = \hat{e}(r_A P_{pub} + d_A, T_B + Q_B) & \xleftarrow{T_B = r_B P} & K = \hat{e}(r_B P_{pub} + d_B, T_A + Q_A) \\
\hline
\end{array}$$

Fig. 19. Shim's Protocol [47]

the messages, A computes $K_A = \hat{e}(r_A P_{pub} + d_A, T_B + Q_B)$ and B computes $K_B = \hat{e}(r_B P_{pub} + d_B, T_A + Q_A)$. The agreed secret is $K = K_A = K_B = \hat{e}(P,P)^{r_A r_B s} \cdot \hat{e}(Q_A, P)^{r_B s} \cdot \hat{e}(P, Q_B)^{r_A s} \cdot \hat{e}(Q_A, Q_B)^s$. Each party sets the session key as $SK = H(K \| ID_A \| ID_B)$. The protocol was demonstrated to be vulnerable to the man-in-the-middle attack in [45] as in Figure 20. In the

$$\begin{array}{ccccc}
A & & E & & B \\
\hline
 & \xrightarrow{T_A = aP} & \xrightarrow{T'_A = a'P - Q_A} & & \\
 & & \xleftarrow{T'_B = b'P - Q_B} & \xleftarrow{T_B = bP} & \\
K = \hat{e}(aP_{pub} + d_A, T'_B + Q_B) & & & & K = \hat{e}(T'_A + Q_A, bP_{pub} + d_B) \\
\hline
\end{array}$$

Fig. 20. Attack on Shim's Protocol [45]

attack $K_{AB} = \hat{e}(aP_{pub} + d_A, T'_B + Q_B) = \hat{e}(P,P)^{asb'} \cdot \hat{e}(Q_A, P)^{sb'} = \hat{e}(aP, P_{pub})^{b'} \cdot \hat{e}(Q_A, P_{pub})^{b'}$, and $K_{BA} = \hat{e}(T'_A + Q_A, bP_{pub} + d_B) = \hat{e}(P,P)^{a'sb} \cdot \hat{e}(Q_B, P)^{a's} = \hat{e}(P_{pub}, sP)^{a'} \cdot \hat{e}(Q_B, P_{pub})^{a'}$. Both keys are computable by the adversary. In this attack, the adversary cannot recover the corresponding random flips r_1 (resp. r_2) such that $r_1 P = a'P - Q_A$ (resp. $r_2 P = b'P - Q_B$). This means that the newly formalized model cannot cover all the possible attacks of case 1. On the other hand, the model disallowing the *Reveal* query appears to be able to check this type of attack.

Hence, it is more reasonable to check the security of a protocol using both methods.[9]

Definition 17 *Protocol Π is a secure AK which uses only general-commutative operations on the random flips if:*

1. *In the presence of the benign adversary on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, both oracles always accept holding the same session key σ, and this key is distributed uniformly at random on $\{0,1\}^k$;*

and if for every adversary E:

---

[8] Note that the subgroup attack in [35] is detectable in the model.

[9] Note that the other models of AK protocols in the literature are neither suitable for AK's with general-commutative computation, such as the model used in [11].

2. *If two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have matching conversations and both i and j are uncorrupted, then both accept and hold the same session key σ;*
3. *$Advantage^E(k)$ is negligible if the fresh oracle is defined as Definition 7 and the* Reveal *query is disallowed;*
4. *$Advantage^E(k)$ is negligible if the fresh oracle is defined as Definition 14.*

We show an example of using the new model. Note that Shim's attack [48] on ATK-2 [2] are detectable if the new model is extended to the tripartite case. So, ATK-2 cannot be proved to be secure in the new model. In [25], the proof of the security of the Chen-Kudla's AK is problematic [18]. In the revised version [26] of the paper the authors use the model without *Reveal* query (so, the condition 3 of Definition 17 is satisfied). We present another proof (of condition 4) in the new model as a complement to demonstrate the security strength of the protocol. The security of the Chen-Kudla's AK can be defined by the following theorem (Theorem 3) where an "unattacked" oracle was not asked the *Reveal, Coin* and *Corrupt* queries.

Theorem 3 *The Chen-Kudla's AK is a secure AK protocol against Type-I adversaries under Definition 17, provided that H_1 and H_2 are random oracles and the BDH is hard.*

The proof is presented in Appendix.

As how to formalize a general indistinguishability-based model for AK protocol remains an open problem, before such a general model is presented, it is recommended to include the asymmetric computation on random flips in the agreed key generation functions, which enable us to prove the security of protocols. A simple way to introduce the asymmetric computation is to apply a hash operation on the agreed secret and the exchanged messages of a session (or the identity of involved parties) to generate the session key[10].

5 Conclusion

In this paper, we review the development of indistinguishability-based security models of key agreement protocols of simple cases and organize the models in a unified framework with a few new extensions. By introducing a new *Coin* query and defining different *fresh oracle* and *no-matching* notions, the model enables an adversary to fully exploit its capacity, so to define a strong security notion for AKC's and some AK's. By allowing different queries and using different definition variants of *fresh oracle* and *no-matching*, the framework is flexible enough to prove the security of AKC protocols with different properties. For AK's, the model with the *Coin* query provides a heuristic method, which is stronger than the model without the *Reveal* query, to evaluate the security of a large category of AK protocols. We use these models to analyze the security of a few pairing-based authenticated key agreement protocols.

6 Acknowledgements

We would like to thank Caroline Kudla and Liqun Chen's comments on the early version of the paper. Specially Caroline Kudla explained the attack in Figure 17. We also want to thank Yangge Wang's comments on the latest version of the paper.

[10] We should be careful to use the asymmetric operation properly, because it is possible that a protocol still suffers from the general concatenation attack even with an asymmetric operation applied on the exchange messages, such as the protocol in [51].

References

1. S. S. Al-Riyami and K. G. Paterson. Tripartite Authenticated Key Agreement Protocols from Pairings. IMA Conference on Cryptography and Coding, LNCS 2898, pp. 332-359. See also Cryptology ePrint Archive, Report 2002/035.
2. M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). IFIP TCS 2000, Sendai, Japan, August 2000.
3. M. Burrows, M. Abadi and R. Needham. A logic for authentication. DEC Systems Research Center Technical Report 39, February 1990.
4. M. Bellare, A. Boldyreva and A. Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. Advances in Cryptology - Eurocrypt 2004, LNCS 3027, 2004.
5. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. Advances in Cryptology - Crypto 96, LNCS 1109, 1996.
6. M. Bellare, R. Canetti and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. Proc. of the 30th STOC. ACM Press, New York, 1998.
7. E. Bresson, O. Chevassut and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange - the dynamic case. Advances in Cryptology - Proceedings of AsiaCrypt 2001, LNCS 2248 pp. 290-309, 2001.
8. E. Bresson, O. Chevassut and D. Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. Advances in Cryptology - Proceedings of Eurocrypt 2002, LNCS 2332 pp. 321-336, 2002.
9. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. Advances in Cryptology - Crypto'2001, LNCS 2139, pp.213-229, 2001.
10. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung. Systematic design of two-party authentication protocols. Advances in Cryptology-Proceedings of CRYPTO 91, 1991.
11. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. Advances in Cryptology - Eurocrypt 2000 Proceedings, LNCS 1807, 2000.
12. M. Bellare, E. Petrank, C. Rackoff and P. Rogaway. Authenticated key exchange in the public key model. manuscript, 1995-1996
13. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Desiging Efficient Protocols. Proc. of First ACM Conference on Computer and Communication Security, November 1993.
14. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. CRYPTO '93, LNCS 773, pp. 232-249, 1994.
15. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: the Three Party Case. Proc. of the 27th STOC. ACM Press, New York, 1995.
16. S. Blake-Wilson, D. Johnson and A. Menezes. Key Agreement Protocols and their Security Analysis. the Sixth IMA International Conference on Cryptography and Coding, Cirencester, England, 1997.
17. S. Blake-Wilson and A. Menezes. Entity Authentication and Authenticated Key Transport Protocols Employing Asymmetric Techniques. Security Protocols Workshop '97, 1997.
18. Z. Cheng. Private communication. 2003.
19. Z. Cheng. Private communication. 2004.
20. Z. Cheng and L. Chen. On Security Proofs of McCullagh-Barreto's Authenticated Key Agreements. manuscript, 2005.
21. Z. Cheng, R. Comley and L. Vasiu. Revisit The Indistinguishability-Based Model of Key Agreement Protocol. manuscript, Feburary 2004.
22. R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. the Proc. of the 30th ACM Symp. on Theory of Computing (STOC), 1998.
23. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. Eurocrypt 2001, LNCS 2045, 2001.
24. R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. Eurocrypt 2002, LNCS 2332, 2002
25. L. Chen and C. Kudla. Identity Based Authenticated Key Agreement from Pairings. The 16th IEEE Computer Security Foundations Workshop(CSFW'03) 2003.
26. L. Chen and C. Kudla. Identity Based Authenticated Key Agreement from Pairings. Cryptology ePrint Archive, Report 2002/184, Revised version.

27. R. Dupont and A. Enge. Practical Non-Interactive Key Distribution Based on Pairings. Proceedings of the International Workshop on Coding and Cryptography (WCC), Versailles, 2003. See also Cryptology ePrint Archive, Report 2002/136.
28. W. Diffie, P. van Oorschot and M. Wiener. Authentication and authenticated key exchanges. Designs, Codes and Cryptography, 2, 107-125, 1992.
29. S. Goldwasser and S. Micali. Probabilistic encryption. Journal of Computer and System Sciences Vol. 28, 270-299, 1984.
30. O. Goldreich. Foundations of cryptography: volume II basic applications. Cambridge University Press, 2004.
31. D. Hofheinz, J. Müller-Quade and R. Steinwandt. Initiator-Resilient Universally Composable Key Exchange. ESORICS 2003, LNCS 2808, 2003.
32. B. S. Kaliski. An unknown key-share attack on the MQV key agreement protocol. ACM Transactions on Information and System Security, vol. 4, no. 3, pp. 275-288, 2001.
33. K. Kobara and H. Imai. Pretty-Simple Password-Authenticated Key-Exchange Protocol Proven to be Secure in the Standard Model. IEICE Trans. Fundamentals, Vol. E85-A, pp. 2229-2237, Oct. 2002.
34. C. Kudla. Private communication. 2004.
35. C. H. Lim, and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In Advances in Cryptology - CRYPTO'97, 1997.
36. L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. Designs, Codes and Cryptography, 28, pp. 119-134, 2003.
37. P. J. Leadbitter and N. P. Smart. Cryptanalysis of MQV with partially known nonces. Cryptology ePrint Archive: Report 2002/145, 2002
38. N. McCullagh and P. S. L. M. Barreto. A New Two-Party Identity-Based Authenticated Key Agreement. CT-RSA 2005, LNCS 3376, 2005.
39. N. McCullagh and P. S. L. M. Barreto. A New Two-Party Identity-Based Authenticated Key Agreement. Updated version, Cryptology ePrint Archive, Report 2004/122.
40. A. Menezes, P. van Oorschot and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.
41. T. Matsumoto, Y. Takashima and H. Imai. On seeking smart public-key distribution systems. The Transactions of the IEICE of Japan, Vol. E69, pp.99-106, 1986.
42. T. Okamoto and D. Pointcheval. The gap-problems: a new class of problems for the security of cryptographic schemes. in Practice and Theory in Public Key Cryptography-PKC'2001, LNCS 1992, 2001.
43. O. Pereira. Modelling and Security Analysis of Authenticated Group Key Agreement Protocols. Dissertation, 2003
44. V. Shoup. On Formal Models for Security Key Exchange. Theory of Cryptography Library, 1999.
45. H.-M. Sun and B.-T. Hsieh. Security Analysis of Shim's Authenticated Key Agreement Protocols from Pairings. Cryptology ePrint Archive, Report 2003/113.
46. M. Scott. Authenticated ID-based Key Exchange and remote log-in with insecure token and PIN number. Cryptology ePrint Archive, Report 2002/164.
47. K. Shim. Efficient ID-based authenticated key agreement protocol from the Weil pairing. Electronics Letters 39, pp. 653-654, 2003.
48. K. Shim. Cryptanalysis of Al-Riyami-Paterson's Authenticated Three Party Key Agreement Protocols. Cryptology ePrint Archive, Report 2003/122.
49. N. P. Smart. An Identity Based Authenticated Key Agreement Protocol Based on the Weil Pairing. Electronics Letters 38, pp.630-632, 2002. See also Cryptology ePrint Archive, Report 2001/111.
50. V. Shoup and A. Rubin. Session Key Distribution using Smart Cards. Advances in Cryptology-Eurocrypt 96, LNCS 1070, 1996.
51. Y. Wang. Efficient Identity-Based and Authenticated Key Agreement Protocol. Cryptology ePrint Archive, Report 2005/108.
52. F. Zhang, R. Safavi-Naini and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In International Workshop on Practice and Theory in Public Key Cryptography-PKC 2004, 2004.

Appendix

Proof of Theorem 1.

Proof: In the proof, the session ID is used to define the matching conversations. However, we use the complete transcript of an oracle as the session ID to prove conditions 1 and 2 in Definition 8 of AKC which follow immediately from the description of the protocol. We use the concatenation of the first two messages of a transcript as the session ID to prove conditions 3 and 4 (see Remark 4 for the reason).

The following part treats the condition 3. If the advantage of an adversary $\mathcal{A}$ against the protocol is non-negligible, i.e., Pr[No-matching$^{\mathcal{A}}(k)$] is non-negligible, we can construct an adversary $\mathcal{B}$ using $\mathcal{A}$ as a subroutine against either a BDH challenger or the used MAC scheme with non-negligible advantage. We say that $\mathcal{A}$ wins if at the end of the game, there exists an oracle $\Pi^s_{i,j}$ (j is uncorrupted and i has not been asked the $Coin(\Pi^s_{i,j}, r)$ query) which has accepted but no oracle $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$. Let us assume Pr[$\mathcal{A}$ wins]=$n(k)$ for some non-negligible function $n(k)$. Suppose, in the game, there are $T_1(k)$ oracles created by the engaged parties and $\mathcal{A}$, and the random oracle H_2 has been queried for $q_2(k)$ times and the *Corrupt* query has been asked for $q_C(k)$ times.

In the sequel (in the proofs of Theorem 1 and 2), we slightly abuse the notation $\Pi^s_{i,j}$ as the s-th oracle among all the oracles initiated by all the party or the adversary, instead of the s-th instance of i. The session is still between party i and j. This abuse does not affect the soundness of the model because s originally is just used to uniquely identify an instance of party i. Now, we assume that there is a global counter and every time when an oracle is created, the counter is increased. The global counter is used to uniquely identity the instance of a party in all the sessions.

Now let us construct the adversary $\mathcal{B}$. $\mathcal{B}$ is provided with two challengers i.e., a BDH challenger and a MAC$_{k''}$ challenger where $k'' \in_R \{0,1\}^l$. $\mathcal{B}$ needs to either answer a BDH challenge or construct a pair of message and tag $\{msg, \alpha\}$ such that α =MAC$_{k''}(msg)$ and the message msg has not been queried to its MAC$_{k''}$ challenger. When given the BDH challenge $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, sP, bP, rP\rangle$, $\mathcal{B}$ simulates the system parameter generator $\mathcal{G}$ of the KGC by choosing system params as $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, sP, H_1, l, H_2, H_3\rangle$, where $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$ and H_2 (resp. H_3): $\mathbb{G}_2^* \to \{0,1\}^l$ are three random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ randomly chooses $u \in_R \{1, \dots, T_1(k)\}$ and interacts with $\mathcal{A}$ in the following way:

1. $H_1(ID_i)$ query. $\mathcal{B}$ maintains an initially empty list of tuples $(ID_i, Q_i, h_i, d_i, coina_i)$ sorted according to ID_i, denoted as H_1^{list}. When $\mathcal{A}$ queries oracle H_1 on ID_i, $\mathcal{B}$ responds as follows:
   (a) If one tuple corresponding to ID_i is on the list, $\mathcal{B}$ returns Q_i of the tuple.
   (b) Otherwise, $\mathcal{B}$ generates a random $coina \in \{0,1\}$ so that $\Pr[coina = 0] = \delta$ for some δ that will be determined later.
   (c) $\mathcal{B}$ picks a random $h_i \in \mathbb{Z}_q^*$. If $coina = 0$, $\mathcal{B}$ computes $Q_i = h_iP \in \mathbb{G}_1^*$ and $d_i = h_isP$; otherwise $\mathcal{B}$ sets $Q_i = h_ibP$ and $d_i = \perp$.
   (d) $\mathcal{B}$ inserts the tuple $(ID_i, Q_i, h_i, d_i, coina)$ into the list H_1^{list} and returns $H_1(ID_i) = Q_i$.
2. $H_2(p)$ query. $\mathcal{B}$ maintains an initially empty list of pairs $(p, value)$, denoted as H_2^{list}. If one pair corresponding to p is on the list, $\mathcal{B}$ returns the $value$ of the pair. Otherwise, $\mathcal{B}$ randomly selects an integer r from $\{0,1\}^l$ and after inserting (p, r) in the list H_2^{list} returns r.
3. $H_3(p)$ query. $\mathcal{B}$ maintains an initially empty list of pairs $(p, value)$, denoted as H_3^{list}. If one pair corresponding to p is on the list, $\mathcal{B}$ returns the $value$ of the pair. Otherwise, $\mathcal{B}$ randomly selects an integer r from $\{0,1\}^l$ and after inserting (p, r) in the list H_3^{list} returns r.

4. Corrupt query $Corrupt(i, \lambda)$. $\mathcal{B}$ looks through H_1^{list} to find the corresponding tuple with i equal to ID. If on the tuple $coina = 0$, $\mathcal{B}$ returns d_i; otherwise it reports a failure and aborts (**Event 1**). Because in the IBC systems, we can regard the ID as the party's public key, the adversary cannot update the public and private key pair without changing the party's ID. Hence, we assume that the adversary will not try to replace a party's public/private keys.
5. Replace query $Replace(\Pi_{i,j}^s, P)$. As this protocol is an identity-based scheme, the public key of a party is then related to the party's identity. Hence, no adversary will try this attack, because it will never succeed.
6. Coin query $Coin(\Pi_{i,j}^s, r)$. If $s = u$ and $coina_i$ of the tuple corresponding to $\Pi_{i,j}^s$ on H_1^{list} is 0, $\mathcal{B}$ then reports a failure and aborts (**Event 2**). Otherwise, if $r = \lambda$, $\mathcal{B}$ finds r_i^s corresponding to $\Pi_{i,j}^s$ from $Coin^{list}$ and returns it as the response. If $r \neq \lambda$, $(\Pi_{i,j}^s, r)$ is inserted in the $Coin^{list}$.
7. Send query $SendP(\Pi_{i,j}^s, m)$. $\mathcal{B}$ maintains an initially empty list $Coin^{list}$ to store the pairs in the form of $(\Pi_{i,j}^s, r_i^s)$ and another initially empty list $Conv^{list}$ to store the conversation transcripts in the form of $(\Pi_{i,j}^s, conv, K_{i,j}^s, K_{i,j}'^s)$. $\mathcal{B}$ first finds the corresponding $coina_i, coina_j, d_i, d_j, Q_i, Q_j, h_i$ and h_j from H_1^{list}. In the following cases, $\mathcal{B}$ responds differently.
   - If $s = u$ and ($coina_i \neq 0$ or $coina_j \neq 1$), $\mathcal{B}$ reports a failure and aborts (**Event 3**).
   - Otherwise, if $s = u$ (then $coina_i = 0$ and $coina_j = 1$)
     - If $m = \lambda$ (so i is the initiator), $\mathcal{B}$ responds with rP as the first message of the session.
     - If m is the first message of the session (so i is the responder) in the form of R, $\mathcal{B}$ accesses the oracle $\mathrm{MAC}_{k''}$ with message $msg_2 = (2, i, j, rP, R)$ to get the tag α, then it responds with $\{msg_2, \alpha\}$. If this message is rejected by $\mathcal{A}$, $\mathcal{B}$ computes $U^{-1} = \hat{e}(sP, R)^{-h_i}$ and randomly chooses a tuple from the list H_2^{list}. Suppose the value of p of the chosen tuple is V, $\mathcal{B}$ computes $W = V \cdot U^{-1}$. Then $\mathcal{B}$ responds to its BDH challenger with W^{1/h_j} and terminates the game.
     - If m is the second message of the session (so i is the initiator) in the form of $\{msg_2, \alpha\}$= $\mathrm{MAC}_X(2, j, i, R, rP)$, then $\mathcal{B}$ takes the following steps. Note that the valid agreed secret should be $K = \hat{e}(h_i sP, R) \cdot (rh_j bP, sP)$ if the session is completed. $\mathcal{B}$ computes $U^{-1} = \hat{e}(sP, R)^{-h_i}$ and randomly chooses a tuple from the list H_2^{list}. Suppose the value of p of the chosen tuple is V. $\mathcal{B}$ computes $W = V \cdot U^{-1}$. Then $\mathcal{B}$ responds to its BDH challenger with W^{1/h_j} and terminates the game.
     - If m is the third message of the session (so i is the responder) in the form of $\{msg_3, \beta\}$= $\mathrm{MAC}_Y(3, j, i, R, rP)$, $\mathcal{B}$ computes $U^{-1} = \hat{e}(sP, R)^{-h_i}$ and randomly chooses a tuple from H_2^{list}. Suppose the value of p of the chosen tuple is V. $\mathcal{B}$ computes $W = V \cdot U^{-1}$. Finally $\mathcal{B}$ responds to its BDH challenger with W^{1/h_j} and responds to its $\mathrm{MAC}_{k''}$ challenger with $\{msg_3, \beta\}$ and terminates the game.
   - Otherwise, if $coina_i = 1$, $\mathcal{B}$ reports a failure and aborts (**Event 4**).
   - Otherwise, $\mathcal{B}$ answers the query as specified by Π.
     - $\mathcal{B}$ tries to find the random flips r_i^s corresponding to $\Pi_{i,j}^s$ from $Coin^{list}$ and if it fails, $\mathcal{B}$ randomly selects an integer $r_i^s \in \mathbb{Z}_q^*$ and inserts $(\Pi_{i,j}^s, r_i^s)$ into the $Coin^{list}$.
     - If $m = \lambda$ (so i is the initiator), $\mathcal{B}$ responds with $r_i^s P$ as the first message of the session, at the same time sets $K_{i,j}^s = \perp$, $K_{i,j}'^s = \perp$ and updates $conv$.
     - If m is the first message of the session (so i is the responder) in the form of R, $\mathcal{B}$ computes $K = \hat{e}(d_i, R) \cdot \hat{e}(r_i^s Q_j, sP)$. Then it computes $K' = H_2(K)$ and responds

with $\mathrm{MAC}_{K'}(2, i, j, r_i^s P, R)$, at the same time sets $K_{i,j}^s = K$, $K'^s_{i,j} = K'$ and updates *conv*.

- If m is the second message of the session (so i is the initiator) in the form of $\{msg_2, \alpha\}=$ $\mathrm{MAC}_X(2, j, i, R, r_i^s P)$, then $\mathcal{B}$ takes the following actions. First it computes $K = \hat{e}(d_i, R) \cdot \hat{e}(r_i^s Q_j, sP)$ and $K' = H(K)$. Second, $\mathcal{B}$ checks the tag of the input message. If α is a valid tag, then it responds with $\mathrm{MAC}_{K'}(3, i, j, r_i^s P, R)$, at the same time, sets $K_{i,j}^s = K$, $K'^s_{i,j} = K'$ and updates *conv* and accepts. Otherwise, $\mathcal{B}$ rejects the input message.
- If m is the third message of the session (so i is the initiator) in the form of $\{msg_3, \beta\}=$ $\mathrm{MAC}_Y(3, j, i, R, r_i^s P)$, $\mathcal{B}$ checks the tag and if it is valid, then it accepts the session. Otherwise, $\mathcal{B}$ rejects the input message. Note that here, msg_3 is used only for ease of interpretation. When $\mathcal{B}$ checks the tag, it should generate msg_3 by itself (in fact, msg_3 will not be exchanged in the real protocol).

8. Reveal query $Reveal(\Pi_{i,j}^s)$. If the session is accepted, then $\mathcal{B}$ returns $H_3(K_{i,j}^s)$; otherwise, $\mathcal{B}$ outputs $\perp$.

If the adversary $\mathcal{B}$ does not abort during the simulation, then $\mathcal{A}$'s view is identical to its view in the real attack. To complete the proof, firstly let us check the probability that $\mathcal{B}$ does not abort during the simulation. Let $\mathcal{F}$ be the event that $\mathcal{B}$ does not abort the game and $\mathcal{H}_i$ be **Event** i for $i \in \{1, 2, 3, 4\}$ and let $\mathcal{H}_0$ be the event that $\mathcal{A}$ wins the game (making the oracle $\Pi_{i,j}^s$ accept) in the u-th session. Let $\mathcal{E}$ be the event that $\mathcal{B}$ finds the correct response to the BDH challenger or a valid pair of message and tag $\{msg_3, \beta\}$ where msg_3 has not been queried to the MAC oracle. Hence

$$Pr[\mathcal{B} \text{ wins}] = Pr[\mathcal{F} \wedge \mathcal{H}_0 \wedge \mathcal{E}]$$

and

$$Pr[\mathcal{F}] = Pr[\neg\mathcal{H}_1 \wedge \neg\mathcal{H}_2 \wedge \neg\mathcal{H}_3 \wedge \neg\mathcal{H}_4].$$

We have

$$\begin{aligned} Pr[\neg\mathcal{H}_1] &= \delta^{q_C} \\ Pr[\neg\mathcal{H}_2] &= 1 - \delta \\ Pr[\neg\mathcal{H}_3] &= \delta(1 - \delta) \\ Pr[\neg\mathcal{H}_4] &= \delta^{T_1} \end{aligned}$$

Therefore the overall probability of non-abortion is at least $\delta^{(q_C+T_1+1)}(1-\delta)^2$. This value is maximized at $\delta_{opt} = \frac{q_C+T_1+1}{q_C+T_1+3}$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort the game is at least $\frac{4}{e(3+q_C+T_1)^2}$.

Suppose that $\mathcal{A}$ did make party i accept the u-the session with party j. If i is the initiator of the session, $\mathcal{A}$ has to query the random oracle to find the random value of the corresponding agreed secret because H_2 is supposed to be fully random. Otherwise, $\mathcal{A}$ can only guess the correct value with negligible probability, so to convince i to accept the session after receiving the second message of the session. Because $\mathcal{B}$ randomly chooses a pair from H_1^{list}, we have $Pr[\mathcal{E}|\mathcal{H}_0 \wedge \mathcal{F}] \geq \frac{1}{q_2}$. If i is the responder of the session, because $\mathcal{B}$ uses the random key k'' held by the MAC oracle to generate the pair of the second message $\{msg_2, \alpha\}$, $\mathcal{A}$ can force i to accept the session either by generating the valid pair of $\{msg_3, \beta\}$ under key k'' or by finding the valid agreed secret K first and querying the oracle $K' = H_2(K)$ and generating the valid third message under K'. Hence, if $\mathcal{A}$ is in the former case, $\mathcal{B}$ wins the game with the MAC challenger with probability 1; otherwise, $\mathcal{B}$ wins the game with the BDH challenger with probability at least $\frac{1}{q_2}$. Note that in the latter case, $\mathcal{A}$ could possibly reject the session (the second message created by $\mathcal{B}$), and $\mathcal{B}$ correctly answers the BDH challenge with probability at least $\frac{1}{q_2}$ as well because of the behavior of $\mathcal{B}$ after the message was rejected. Overall $Pr[\mathcal{E}|\mathcal{H}_0 \wedge \mathcal{F}] \geq \frac{1}{q_2}$. So, $\mathcal{B}$ wins the whole game with probability:

$$Pr[\mathcal{B} \text{ wins}] = Pr[\mathcal{F} \wedge \mathcal{H}_0 \wedge \mathcal{E}] \geq n(k)\frac{1}{q_2(k)}\frac{1}{T_1(k)}\frac{4}{e(3+q_C(k)+T_1(k))^2}$$

Using a very similar strategy we can prove that the condition 4 is also satisfied. In fact, it appears that in this protocol (and many protocols using the similar structure), the condition 3 implies the condition 4. This can be interpreted in this way. As shown in the proof, the strength of the protocol against the adversary to deceit a party into accepting a session relies on the hardness to compute K. The proof above has shown that the adversary working as a responder cannot succeed. Due to the security property of the used MAC scheme, given the third message (a MAC tag of a new message), the adversary cannot generate new message/tag pairs which implies that the adversary still cannot recovery K' and (so cannot compute) K (after the completion of the protocol). As H_3 is a random oracle, so the adversary cannot differentiate $SK = H_3(K)$ from a random sample (without knowing K). However the condition 3 does not necessarily imply the condition 4 in all protocols (see the discussion in Section 4.1). $\square$

Remark 7 *In the proof, there is an assumption that the upper bound of $q_C(k) + T_1(k)$ on the number of* Corrupt *queries and* SendP *queries of $\mathcal{A}$ is known by $\mathcal{B}$ in advance. Furthermore, the construction cannot answer some* SendP *queries. This certainly shows the limitation of the proof. However, the strategy seems to be commonly used in the literature, e.g., [9] [27].*

Proof of Theorem 2.

Proof: Suppose that there exits an adversary $\mathcal{A}$ against the PFS of Protocol 4 with the new key generation function, we can construct an algorithm $\mathcal{B}$ using $\mathcal{A}$ as a subroutine to solve the GDH problem.

When given the GDH challenge $\langle q, \mathbb{G}_1, P, aP, bP\rangle$, $\mathcal{B}$ simulates the system parameter generator $\mathcal{G}$ of the KGC by randomly choosing $s \in \mathbb{Z}_q^*$ and setting system params as $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, sP, H_1, l, H_2, H_3\rangle$, where $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$ and H_2 (resp. H_3): $\mathbb{G}_2^* \rightarrow \{0,1\}^l$ are three random oracles controlled by $\mathcal{B}$. Suppose, in the game, there are $T_1(k)$ oracles created by the engaged parties and $\mathcal{A}$, and the random oracle H_3 has been query for $q_3(k)$ times. $\mathcal{B}$ randomly chooses $u \in_R \{1, \ldots, T_1(k)\}$ and interacts with $\mathcal{A}$ in the following way.

1. $H_1(ID_i)$ query. $\mathcal{B}$ maintains an initially empty list of tuples (ID_i, Q_i, h_i) sorted according to ID_i, denoted as H_1^{list}. When $\mathcal{A}$ queries oracle H_1 on ID_i, $\mathcal{B}$ responds as follows:
   (a) If one tuple corresponding to ID_i is on the list, $\mathcal{B}$ returns Q_i of the tuple.
   (b) Otherwise, $\mathcal{B}$ picks a random $h_i \in \mathbb{Z}_q^*$ and sets $Q_i = h_iP$. $\mathcal{B}$ inserts the tuple (ID_i, Q_i, h_i) into the list H_1^{list} and returns $H_1(ID_i) = Q_i$.

2. $H_2(p_1, \ldots, p_4)$ query (recall that H_2 in Figure 7 takes four inputs). $\mathcal{B}$ maintains an initially empty list of tuples $(p_1, \ldots, p_4, value)$, denoted as H_2^{list}. If one tuple corresponding to $(p_1, \ldots, p_4)$ is on the list, $\mathcal{B}$ returns the *value* of the tuple. Otherwise, if a special tuple with the form $(K, \star, aP, bP, r)$ (which is generated in the following part of simulation) is on the list, then $\mathcal{B}$ checks if $p_1 = K, p_3 = aP, p_4 = bP$ and $\hat{e}(aP, bP) = \hat{e}(p_2, P)$. If the conditions are satisfied, $\mathcal{B}$ replaces $\star$ with p_2 and returns r as the result. Otherwise, $\mathcal{B}$ randomly selects an integer r from $\{0,1\}^l$ and after inserting $(p_1, \ldots, p_4, r)$ in the list H_2^{list} returns r.
3. $H_3(p_1, \ldots, p_4)$ query (recall that H_3 in Figure 7 takes four inputs). $\mathcal{B}$ maintains an initially empty list of tuples $(p_1, \ldots, p_4, value)$, denoted as H_3^{list}. If one tuple corresponding to $(p_1, \ldots, p_4)$ is on the list, $\mathcal{B}$ returns the *value* of the tuple. Otherwise, $\mathcal{B}$ randomly selects an integer r from $\{0,1\}^l$ and after inserting $(p_1, \ldots, p_4, r)$ in the list H_3^{list} returns r.
4. Corrupt query $Corrupt(i, \lambda)$. $\mathcal{B}$ looks through H_1^{list} to find the corresponding tuple with i equal to ID. $\mathcal{B}$ returns $d_i = sh_iP$. Again we assume that adversary will not try to replace a party's public/private key pair.
5. Replace query $Replace(\Pi_{i,j}^s, P)$. No adversary will try this attack, because it will never succeed.
6. Coin query $Coin(\Pi_{i,j}^s, r)$. If $s = u$, $\mathcal{B}$ then reports a failure and aborts (**Event 1**). Otherwise, if $r = \lambda$, $\mathcal{B}$ finds r_i^s corresponding to $\Pi_{i,j}^s$ from $Coin^{list}$ and returns it as the response; otherwise, $(\Pi_{i,j}^s, r)$ is inserted in the $Coin^{list}$.
7. Send query $SendP(\Pi_{i,j}^s, m)$. $\mathcal{B}$ maintains an initially empty list $Coin^{list}$ to store the pairs in the form of $(\Pi_{i,j}^s, r_i^s)$ and another initially empty list $Conv^{list}$ to store the conversation transcripts in the form of $(\Pi_{i,j}^s, conv, K_{i,j}^s, K_{i,j}'^s)$. $\mathcal{B}$ first finds the corresponding Q_i, Q_j from H_1^{list} and computes $d_i = sQ_i, d_j = sQ_j$. In the following cases, $\mathcal{B}$ responds differently.
   - If $s \neq u$, $\mathcal{B}$ responds by following the protocol, i.e., randomly choosing r_i^s, computing r_i^sR (R is the component in the input message) and K and $K_{i,j}'^s$ using d_i, d_j, input message and r_i^s and checking the tag of the input message, so to accept or reject the session. Finally, if the session is accepted, $\mathcal{B}$ computes $SK_{i,j}^s$.
   - Otherwise,
     - If $m = \lambda$ (i is the initiator), then $\mathcal{B}$ responds with aP.
     - If the input message equals to aP (i is the responder), then $\mathcal{B}$ responds in the following way. First, $\mathcal{B}$ computes $K = \hat{e}(d_i, aP) \cdot \hat{e}(Q_j, bP)^s$. Then $\mathcal{B}$ looks through H_2^{list} to find all the tuples with $p_1 = K, p_3 = aP, p_4 = bP$. In the found tuples set, $\mathcal{B}$ tests each p_2 to find whether such a p_2 satisfying $\hat{e}(P, p_2) = \hat{e}(aP, bP)$ exists. If such a p_2 is found, then $\mathcal{B}$ responds to the GDH challenger with p_2 and wins the game. (However, this should happen with only negligible probability because of the randomness of aP and bP). Otherwise, $\mathcal{B}$ randomly chooses $k \in \{0,1\}^l$ and inserts a tuple $(K, \star, aP, bP, k)$ to H_2^{list} (note that there is only one such tuple generated in the whole simulation). Finally, $\mathcal{B}$ generates message $MAC_k(2, i, j, bP, R)$ with the message tag 1.
     - If R in the input message ($MAC_k(2, j, i, R, aP)$) equals to bP (i is the initiator), then $\mathcal{B}$ uses k again to generate the message tag 2 ($MAC_k(3, i, j, aP, R)$).
     - For other situations, $\mathcal{B}$ reports an error and aborts (**Event 2**. $\mathcal{A}$ tampered with the messages).

8. Reveal query $Reveal(\Pi^s_{i,j})$. If $s = u$, $\mathcal{B}$ reports an error and aborts (**Event 3**). Otherwise, $\mathcal{B}$ outputs $SK^s_{i,j}$ by following the protocol.
9. Test query $TestD(\Pi^s_{i,j})$. If $s \neq u$, $\mathcal{B}$ aborts the game (**Event 4**). Otherwise, $\mathcal{B}$ randomly chooses a number $k' \in \{0,1\}^l$ and gives it to $\mathcal{A}$. When $\mathcal{A}$ responds, $\mathcal{B}$ randomly chooses a tuple from H_3^{list} with p_3 equal to tag 1 and p_4 equal to tag 2 generated in session u. $\mathcal{B}$ responds to the GDH challenger with the value of p_2 of the chosen tuple.

If the adversary $\mathcal{B}$ does not abort during the simulation, then $\mathcal{A}$'s view is identical to its view in the real attack. $\mathcal{B}$ aborts the game only when **Event** 1, 2, 3 and 4 happen. While these events only happened when $\mathcal{A}$ did not choose session u as the challenge session (in which both oracles were not asked the *Coin* query and the messages were faithfully conveyed). Hence $\mathcal{B}$ does not abort the game with probability at least $1/T_1(k)$. Let $\mathcal{H}$ be the event that abP as p_2 has been queried to H_3. Since H_3 is a random oracle, and the challenged oracles $\Pi^s_{i,j}$ and $\Pi^s_{j,i}$ are *unopened*, $\Pr[\mathcal{A} \text{ wins}|\neg\mathcal{H}] = \frac{1}{2} + \epsilon(k)$ for some negligible function $\epsilon(k)$. Suppose, $\mathcal{A}$ wins the game with non-negligible advantage $n(k)$.

$$n(k) + \tfrac{1}{2} = Pr[\mathcal{A} \text{ wins}] \leq Pr[\mathcal{A} \text{ wins}|\mathcal{H}]Pr[\mathcal{H}] + \tfrac{1}{2} + \epsilon(k) \leq Pr[\mathcal{H}] + \tfrac{1}{2} + \epsilon(k).$$

So $Pr[\mathcal{H}] \geq n(k) - \epsilon(k) > n'(k)$ which is non-negligible. Overall, $\mathcal{B}$ can solve the GDH problem with advantage $\frac{n'(k)}{q_3(k)\cdot T_1(k)}$. Obviously, if aP, bP are included as the inputs to H_3, the reduction can be tighter (with advantage $\frac{n'(k)}{T_1(k)}$) by exploiting the fact that the decisional DH problem is easy in the protocol's setting. □

Remark 8 *In this protocol, it appears that it is possible to prove the security based on the GDHP in the model using the* TestD *query against the Type-I adversary because the cryptographic hash function H_3 is used in the protocol (and treated as a random oracle in the reduction). $\mathcal{B}$ responds to the* TestD *query with a random element from $\{0,1\}^l$, and randomly chooses a value p_2 from H_3^{list} as the response to the GDHP challenger. For a modified protocol without H_3, if in the model with the* TestD *query algorithm $\mathcal{B}$ responds with a random element from $\mathbb{G}_1$ and $\mathcal{A}$ cannot differentiate the simulation from the real world, then at least the decisional DHP is hard. Otherwise, we can simply modify each adversary in the real attack by adding a procedure to check whether the response is the result of the challenge and the modified adversary can easily win the game in the simulation.*

Proof of Theorem 3.

Proof: In the proof, the session ID is used to define the matching conversation. The session ID here is the concatenation of two messages. The proofs of the conditions 1 and 2 in Definition 17 of AK are straightforward. The proof in [26] shows that the protocol satisfies the condition 3. The following part treats the condition 4.

If the advantage of an adversary $\mathcal{A}$ against the protocol is non-negligible, we can construct an adversary $\mathcal{B}$ against a BDH problem with non-negligible advantage. After given the BDH problem $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, sP, aP, bP\rangle$, $\mathcal{B}$ simulates the system parameter generator of the KGC by choosing system params as $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, sP, H_1, l, H_2\rangle$, where $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$ and $H_2 : \mathbb{G}_2^* \rightarrow \{0,1\}^l$ are two random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ interacts with $\mathcal{A}$ in the following way.

1. $H_1(ID_i)$ query. $\mathcal{B}$ maintains an initially empty list of tuples $(ID_i, Q_i, h_i, d_i, coina_i, coinb_i)$ indexed by ID_i, denoted as H_1^{list}. When $\mathcal{A}$ queries oracle H_1 on ID_i, $\mathcal{B}$ responds as follows:
   (a) If one tuple corresponding to ID_i is on the list, $\mathcal{B}$ returns Q_i of the tuple.
   (b) Otherwise, $\mathcal{B}$ generates a random $coina \in \{0,1\}$ so that Pr[coina=0]=δ for some δ that will be determined later.
   (c) $\mathcal{B}$ picks a random $h_i \in \mathbb{Z}_q^*$. If $coina = 0$, $\mathcal{B}$ computes $Q_i = h_iP \in \mathbb{G}_1^*$ and $d_i = h_isP$ and sets $coinb = \perp$. Otherwise, $\mathcal{B}$ uniformly picks $coinb \in \{0,1\}$. If $coinb = 0$, compute $Q_i = h_iaP$, else compute $Q_i = h_ibP$. If $coina = 1$, $\mathcal{B}$ sets $d_i = \perp$.
   (d) $\mathcal{B}$ inserts the tuple $(ID_i, Q_i, h, d_i, coina, coinb)$ into the list H_1^{list} and returns $H_1(ID_i) = Q_i$.
2. $H_2(p)$ query. $\mathcal{B}$ maintains an initially empty list of pairs $(p, value)$, denoted as H_2^{list}. If one pair corresponding to p is on the list, $\mathcal{B}$ returns the $value$ of the pair. Otherwise, $\mathcal{B}$ randomly selects an integer r from $\{0,1\}^l$ and after inserting (p, r) in the list H_2^{list} returns r.
3. Corrupt query $Corrupt(i, K)$. $\mathcal{B}$ looks through H_1^{list} to find the corresponding tuple with i equal to ID. If on the tuple $coina = 0$, $\mathcal{B}$ returns d_i; otherwise it reports a failure and aborts (**Event 1**). Again we assume that adversary will not try to replace a party's public/private key pair.
4. Replace query $Replace(\Pi_{i,j}^s, P)$. No adversary will try this attack, because it will never succeed.
5. Send query $SendP(\Pi_{i,j}^s, m)$. $\mathcal{B}$ maintains an initially empty list $Coin^{list}$ to store the pairs in the form of $(\Pi_{i,j}^s, r_i^s)$ and another initially empty list $Conv^{list}$ to store the conversation transcripts in the form of $(\Pi_{i,j}^s, conv)$. First $\mathcal{B}$ tries to find the random flips r_i^s corresponding to $\Pi_{i,j}^s$ from $Coin^{list}$ and if it fails, $\mathcal{B}$ randomly selects an integer $r_i^s \in \mathbb{Z}_q^*$ and inserts $(\Pi_{i,j}^s, r_i^s)$ into the $Coin^{list}$. $\mathcal{B}$ responds with $r_i^sQ_i$, where Q_i is found from the list H_1^{list} with i equal to the ID field, and $conv$ of $\Pi_{i,j}^s$ is updated as $conv.m.r_i^sQ_i$.
6. Coin query $Coin(\Pi_{i,j}^s, r)$. If $r = \lambda$, $\mathcal{B}$ finds r_i^s corresponding to $\Pi_{i,j}^s$ from $Coin^{list}$ and returns it as the response. Otherwise, $(\Pi_{i,j}^s, r)$ is inserted in the $Coin^{list}$.
7. Reveal query $Reveal(\Pi_{i,j}^s)$. $\mathcal{B}$ responds to the query in the following ways:
   (a) $\mathcal{B}$ uses i as the ID to find d_i and uses j as the ID to find Q_j in the list H_1^{list} and finds r_i^s and $conv$ corresponding to $\Pi_{i,j}^s$ from $Coin^{list}$ and $Conv^{list}$ respectively (assuming the received message is m on the conversation transcript of $\Pi_{i,j}^s$).
   (b) If $d_i \neq \perp$ on the tuple with i as the ID, $\mathcal{B}$ returns $H_2(\hat{e}(d_i, r_i^sQ_j + m))$.
   (c) Otherwise, it reports failure and aborts (**Event 2**).
8. Test query $TestD(\Pi_{i,j}^s)$. $\mathcal{B}$ takes the following actions:
   (a) $\mathcal{B}$ responds to the $TestD$ query with a random element from $\{0,1\}^l$. And after receiving the decision from $\mathcal{A}$,
   (b) $\mathcal{B}$ uses i as the ID to find $coina_i, coinb_i$ and uses j as the ID to find $coina_j, coinb_j$ from the list H_1^{list}
   (c) If $coina_i \neq 1$ or $coina_j \neq 1$, it reports failure and aborts (**Event 3**).
   (d) Otherwise ($coina_i = 1$ and $coina_j = 1$), if $coinb_i = coinb_j$, it reports failure and aborts (**Event 4**).
   (e) Otherwise ($coinb_i \neq coinb_j$), $\mathcal{B}$ randomly chooses a pair from H_2^{list}. Suppose that the value of the chosen pair is R. $\mathcal{B}$ then searches j's conversation transcripts to find the

matching conversation $\Pi^t_{j,i}$ to $\Pi^s_{i,j}$. As the challenged oracles should be fresh (according to Definition 14), the matching conversation can be found. $\mathcal{B}$ uses two oracles to find r^s_i and r^t_j from the $Coin^{list}$ and uses i and j to find h_i and h_j from H_1^{list}. $\mathcal{B}$ computes $R^{[h_i h_j (r^s_i + r^t_j)]^{-1}}$ and returns it as the answer of the BDH problem.

Note that the model requires that, in the challenge session, only one party's random flips can be queried. This rule can be forced to be applied in the proof. For example, if oracle $\Pi^s_{i,j}$ (as the initiator) that has been asked the *Coin* query has the conversation transcript $r^s_i Q_i.m$ where r^s_i is the random flips of $\Pi^s_{i,j}$ and m is the incoming message, then there must be an oracle $\Pi^t_{j,i}$ which with the same conversation has sent message m and has not been issued the *Coin* query. Otherwise, the oracle regards that the adversary did not follow the rule. Note that there are two circumstances where $\Pi^s_{i,j}$ would have such conversation in the model. First, the adversary sends the response message m in the session without going through some oracle $\Pi^t_{j,i}$. This behavior is treated as one *Coin* query has been issued in the session. The second circumstance is that some oracle $\Pi^t_{j,i}$ sends m. Note that in the model, the only way that the adversary can force oracle $\Pi^t_{j,i}$ to send a specific message m is to issue $Coin(\Pi^t_{j,i}, r^t_j)$ query such that $m = r^t_j Q_j$ because otherwise, $\Pi^t_{j,i}$ will randomly choose r (which equals to r^t_j, so to generate m, with only negligible probability) to send rQ_j. It is also required that for the Test query $TestD(\Pi^s_{i,j})$, both i and j are uncorrupted. This can also be checked (in fact, if two parties were corrupted, the simulation would abort before the *TestD* query).

If the adversary $\mathcal{B}$ does not abort during the simulation, then $\mathcal{A}$'s view is identical to its view in the real attack. To complete the proof, firstly let us check the probability that $\mathcal{B}$ does not abort during the simulation. Let $\mathcal{F}$ be the event that $\mathcal{B}$ does not abort the game. Let $q_R(k)$ be the number of *Reveal* queries issued by $\mathcal{A}$. Then the probability that $\mathcal{B}$ does not abort in the *Reveal* query is at least δ^{q_R} (**Event** 2 does not happens at least if $coina_i = 0$, so that the *Reveal* query can be answered) and the probability of non-abortion in the *TestD* query is $\frac{1}{2}(1-\delta)^2$ (the probability that neither **Event** 3 nor 4 happens). Let $q_C(k)$ be the number of *Corrupt* queries issued by $\mathcal{A}$. Then the probability of non-abortion in the *Corrupt* query is δ^{q_C} (the probability that **Event** 1 does not happen). Therefore the overall probability of non-abortion is at least $\frac{1}{2}\delta^{(q_R+q_C)}(1-\delta)^2$. This value is maximized at $\delta_{opt} = \frac{q_R+q_C}{q_R+q_C+2}$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort the game is at least $\frac{2}{e(2+q_R+q_C)^2}$.

Let $\mathcal{H}$ be the event that $\hat{e}(P,P)^{sabh_i h_j (r^s_i + r^t_j)}$ has been queried to H_2. Since H_2 is a random oracle, and the challenged oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ are *unopened*, $\Pr[\mathcal{A} \text{ wins}|\neg\mathcal{H}] = \frac{1}{2} + \epsilon(k)$ for some negligible function $\epsilon(k)$. Suppose, $\mathcal{A}$ wins the game with non-negligible advantage $n(k)$.

$$n(k) + \tfrac{1}{2} = Pr[\mathcal{A} \text{ wins}] \le Pr[\mathcal{A} \text{ wins}|\mathcal{H}]Pr[\mathcal{H}] + \tfrac{1}{2} + \epsilon(k) \le Pr[\mathcal{H}] + \tfrac{1}{2} + \epsilon(k).$$

So $Pr[\mathcal{H}] \ge n(k) - \epsilon(k) > n'(k)$ which is non-negligible. Let $\mathcal{E}$ be the event that $\mathcal{B}$ finds the correct $\hat{e}(P,P)^{sabh_i h_j (r^s_i + r^t_j)}$ on the list H_2^{list}, so $\mathcal{B}$ computes the correct $\hat{e}(P,P)^{sab}$. Let $q_2(k)$ be the number of distinct H_2 hash queries issued during the simulation. We have

$$Pr[\mathcal{B} \text{ wins}] = Pr[\mathcal{F} \wedge \mathcal{H} \wedge \mathcal{E}] \ge n'(k)\frac{1}{q_2(k)}\frac{2}{e(2+q_R(k)+q_C(k))^2}$$

□