Identity-based Key Agreement Protocols From Pairings

L. Chen[1], Z. Cheng[2], and N.P. Smart[3]

[1] Hewlett-Packard Laboratories,
Filton Road,
Stoke Gifford,
Bristol, BS34 8QZ,
United Kingdom.
liqun.chen@hp.com
[2] School of Computing Science,
Middlesex University,
The Burroughs,
Hendon,
London, NW4 4BT,
United Kingdom.
m.z.cheng@mdx.ac.uk
[3] Department of Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom
nigel@cs.bris.ac.uk

Abstract. In recent years, a large number of identity-based key agreement protocols from pairings have been proposed. Some of them are elegant and practical. However, the security of this type of protocols has been surprisingly hard to prove. The main issue is that a simulator is not able to deal with reveal queries, because it requires solving either a computational problem or a decisional problem, both of which are generally believed to be hard (i.e., computationally infeasible). The best solution of security proof published so far uses the gap assumption, which means assuming that the existence of a decisional oracle does not change the hardness of the corresponding computational problem. The disadvantage of using this solution to prove the security for this type of protocols is that such decisional oracles, on which the security proof relies, cannot be performed by any polynomial time algorithm in the real world, because of the hardness of the decisional problem. In this paper we present a method incorporating a built-in decisional function in this type of protocols. The function transfers a hard decisional problem in the proof to an easy decisional problem. We then discuss the resulting efficiency of the schemes and the relevant security reductions in the context of different pairings one can use. We pay particular attention, unlike most other papers in the area, to the issues which arise when using asymmetric pairings.

1 Introduction

Key agreement is a cryptographic protocol, where two or more participants, who each have a long-term key, exchange ephemeral messages (alternatively called key tokens) over an open network with each other. Using the long-term keys and key tokens, these participants generate a session secret shared between them. This secret is used to establish a session key and to perform various security functions, for example, key confirmation, entity and data authentication and confidentiality. The open network is controlled by an adversary, who aims to infiltrate the protocol. A secure key agreement protocol guarantees that the adversary does not succeed.

In this paper we focus on a special type of key agreement protocols, which are two-party identity-based protocols and make use of pairings to compute session secrets. The concept of identity-based cryptography, in which a public key is the identity (an arbitrary string) of a user, and the corresponding private key is created by binding the identity string with a master secret of a trusted authority (called Key Generation Centre), was formulated by Shamir in 1984 [31]. In the same paper, Shamir provided the first identity-based key construction based on the RSA problem, and presented an identity-based signature scheme. By using varieties of the Shamir key construction, a number of identity-based key agreement schemes were proposed (e.g., [16, 25, 36]).

Pioneered by the work of Joux (the first key agreement scheme from pairings [17]), Sakai *et al.* (the first identity-based key construction from pairings [28]), and Boneh and Franklin (the first formally proved identity-based encryption scheme from pairings [4]), many identity-based key agreement protocols from pairings have recently been published, for example [7, 23, 32, 34, 37]. In the latter part of this paper, we will review all of the existing protocols, which are available to the authors. As shown in the review, some of these protocols are elegant and practical. However, the formal security analysis of this type of protocols has been extremely difficult.

The first formal security analysis of a protocol in this type was given by Chen and Kudla [7]. Their protocol (the CK scheme for short) is based on the first identity-based key agreement protocol from pairings by Smart [34]. In the first version of their proof, Chen and Kudla claimed that the CK scheme is secure in the Bellare and Rogaway model [2, 3]. Later on Cheng [8] pointed out a flaw in their proof in dealing with reveal queries. They then corrected this error by modifying the proof under a weaker variant of the Bellare and Rogaway model, where the adversary is not allowed to make reveal queries. A number of other protocols in this type were analysed in this weaker model as well, for example, the McCullagh and Barreto (MB) scheme [23].

Choo, Boyd, and Hitchcock in [12] revisited the CK scheme and MB scheme and demonstrated that after adding the participant identifiers and protocol transcripts into the computation of a session key, these two protocols can be proved secure with a looser restriction, where the adversary is not allowed to make reveal queries to a number of selected sessions, but allowed to other sessions. Their contribution improved the CK and MB scheme and their security analysis, and can also benefit other schemes. However, since a reveal query captures the known-key security property, neither the full nor partial restriction of disallowing reveal queries is really acceptable.

The reason that a simulator cannot answer reveal queries to certain sessions is that, without solving a hard computational problem, the simulator cannot compute the session secrets in these sessions. This property is required on purpose in this type of protocols, otherwise these protocols cannot hold the security property of key-compromise impersonation resilience. We will discuss the relationship between this property and reveal queries in Section 6. If the security proof is based on the random oracle model, the simulator can provide a random number as the session key. In that case, the computational problem is replaced by the corresponding decisional problem, which is believed to be hard (i.e., computationally infeasible) as well. The simulator has to solve this decisional problem in order to maintain the consistency of all random queries.

Some researchers have tried a number of various ways to solve the reveal query issue. For example, Cheng *et al.* in [10] introduced a coin query which can be used to force the adversary to reveal its ephemeral secret. Their approach can deal with some attacks, which have not been covered in the Bellare-Rogaway model without the reveal query. But, the problem of this approach is that the coin query cannot cover a special case that an adversary might break a protocol without knowing the ephemeral secret.

Kudla and Paterson in [19] proposed a modular proof approach, which makes use of a decisional oracle to help the simulator to maintain consistence of random oracle queries. This approach is the best solution published so far and it is a general solution suitable for different types of key agreement protocols. However, if using this approach in the type of protocols, which this paper is focused on, the disadvantage is that such decisional oracles, on which the security proof relies, cannot be performed by any polynomial time algorithm in the real world, because of the hardness of the decisional problem. For example this approach is used in [18] to show that Smart's ID-based key agreement protocol is secure assuming the Gap bilinear Diffie–Hellman (GBDH) problem is hard. In this paper we show how Chen and Kudla's modification of Smart's protocol allows one to prove security without the need for a decisional oracle. Hence, the security is reduced to the hardness of the standard bilinear Diffie–Hellman (BDH) problem.

Wang in [37] proposed another approach, which is opposite to the Kudla and Paterson one. By making use of a computational oracle, Wang analysed the security of his scheme under the decisional bilinear Diffie–Hellman (DBDH) problem. This proof relies on not only a computational oracle, which nobody knows how to perform using any polynomial algorithm in the real world, but also the requirement that the computational oracle cannot be abused by any entity.

In this paper, we propose a new approach to solve the reveal query issue; we incorporate a built-in decisional function in the key agreement protocol. This function makes an adversary release some necessary information, but still keeps the preciseness of a protocol. With the adversary's "help", the simulator can either compute a session secret or recognise the session secret when it is given by the adversary to a random oracle query. The built-in decisional function is designed to distinguish a Diffie–Hellman (DH) triple from a random input in the group where the decisional Diffie–Hellman (DDH) problem is not hard. Therefore, our security proof does not rely on any oracle which we do not known how to perform by using any polynomial algorithm in the real world.

We select four examples of the protocols in this type, and demonstrate that these protocols, with the built-in decisional function, can be proved secure in the Bellare-Rogaway (BR) model under either the computational bilinear Diffie–Hellman (BDH) assumption or the computational ℓ-bilinear Diffie–Hellman inverse (ℓ-BDHI) assumption.

As an extra fruit of this new approach, this built-in decisional function could make some protocols achieve the security property of master key forward secrecy (see examples in Section 7).

The paper is constructed as follows. In Section 2 we briefly review pairings and related assumptions. We then present the formal key agreement model in Section 3. In Section 4, we briefly review the existing identity-based key agreement protocols, which make use of pairings. Then in Section 5 we discuss various efficiency issues, focusing particularly on the case of asymmetric pairings. We then explain how our proof approach works in Section 6. In Sections 7 and 8, we take four examples of the protocols and demonstrate how to incorporating a built-in decisional function in the protocols and prove the security of them under the computational assumptions. Finally we conclude the paper.

1.1 Acknowledgements

The authors would like to thank Caroline Belrose, Yvonne Cliff, Alfred Menezes, Kenny Paterson and Mike Scott for comments on previous versions of this paper.

2 Preliminaries

2.1 Types of Pairings

Here we briefly recall some basic facts of pairings.

Definition 1 *A pairing is a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ between three groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of exponent q, which has the following properties:*

1. *Bilinear: $\forall (P_1, P_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ and $\forall (a, b) \in \mathbb{Z}_q \times \mathbb{Z}_q$, we have $\hat{e}(aP_1, bP_2) = \hat{e}(P_1, P_2)^{ab}$.*
2. *Non-degenerate: There exist non-trivial points $P_1 \in \mathbb{G}_1$ and $P_2 \in \mathbb{G}_2$ both of order q such that $\hat{e}(P_1, P_2) \neq 1$.*

3. Computable: $\forall(P_1,P_2)\in\mathbb{G}_1\times\mathbb{G}_2$, $\hat{e}(P_1,P_2)$ *is efficiently computable.*

It will be convenient to define four different types of pairing system, three of which are taken from [13] whilst the fourth is from [30]. However, before defining our four types of pairing it is convenient to explain how these pairings arise, since then the properties of our four pairing types become easier to explain. It turns out that the properties of the four different types of pairings provide subtle differences to our key agreement protocols and their proofs.

We let $\mathcal{G}=E[q]$, the points of order q on an elliptic curve. We assume the curve is defined over a finite field $\mathbb{F}_p$ and that the group $E[q]$ is contained in $E(\mathbb{F}_{p^k})$, where for simplicity (and efficiency) we assume that k is even. The group $\mathcal{G}$ is a product of two cyclic groups $\mathcal{G}_1$, $\mathcal{G}_2$ of order q. We can take a point $\mathcal{P}_1\in E(\mathbb{F}_p)$ as a generator of $\mathcal{G}_1$ and a point $\mathcal{P}_2\in E(\mathbb{F}_{p^k})$ as a generator of $\mathcal{G}_2$. We select $\mathcal{P}_2$ so that it is in the image of the quadratic twist of E over $\mathbb{F}_{p^{k/2}}$.

We can then define a pairing $\hat{e}$ from $\mathcal{G}\times\mathcal{G}$ to the subgroup $\mathcal{G}_T$ of order q of the finite field $\mathbb{F}_{p^k}$. This pairing is trivial if and only if the two input values are linearly dependent in the vector space $E[q]$. The trace map

$$\mathrm{Tr}:\begin{cases} E(\mathbb{F}_{p^k}) \longrightarrow & E(\mathbb{F}_p),\\ \quad P \quad\longmapsto & \sum_{\sigma\in Gal(\mathbb{F}_{p^k}/\mathbb{F}_p)} P^{\sigma},\end{cases}$$

defines a group homomorphism on $E[q]$ which has kernel $\mathcal{G}_2$.

An important point to note is that Tr and the pairing do not necessarily commute, indeed we have

$$\hat{e}(\mathrm{Tr}(A),B)=\hat{e}(\mathrm{Tr}(B),A)$$

if and only if A and B lie in the same order q subgroup of $\mathcal{G}$. In addition it is easy to produce a hash function which hashes onto $\mathcal{G}_1$, $\mathcal{G}_2$ or $\mathcal{G}$, but it is not easy to produce a function which hashes onto any other subgroup of order q of $\mathcal{G}$, bar $\mathcal{G}_1$ and $\mathcal{G}_2$.

Using the whole group $\mathcal{G}$ for both coordinate in a cryptographic system based on pairings is very inefficient. Thus in the literature one finds a number of specialisations of the above situation. We shall focus on the following four as they are of the most importance. In all cases we let $\mathbb{G}_T=\mathcal{G}_T$.

Definition 2 (Type 1) *In this situation, which corresponds to pairings over supersingular elliptic curves, we can define a pairing using so-called distortion maps, by taking* $\mathbb{G}_1=\mathbb{G}_2=\mathcal{G}_1$. *We let* $P_1=P_2=\mathcal{P}_1$. *There is an efficient algorithm to cryptographically hash arbitrary bit strings into* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *and (a trivial) group isomorphism* $\psi:\mathbb{G}_2\to\mathbb{G}_1$ *mapping* P_2 *to* P_1.

The following three types correspond to pairings over ordinary elliptic curves.

Definition 3 (Type 2) *In this situation, we take* $\mathbb{G}_1=\mathcal{G}_1$ *and* $\mathbb{G}_2$ *to be a subgroup of* $\mathcal{G}$ *which is not equal to either* $\mathcal{G}_1$ *or* $\mathcal{G}_2$. *We set* $P_1=\mathcal{P}_1$ *and for convenience we set* $P_2=\frac{1}{k}\mathcal{P}_1+\mathcal{P}_2$. *There is an efficient algorithm to cryptographically hash arbitrary bit strings into* $\mathbb{G}_1$, *but there is no way to hash bit strings into* $\mathbb{G}_2$ *(nor to generate random elements of* $\mathbb{G}_2$ *bar multiplying* P_2 *by an integer). However, there is an efficiently computable group isomorphism* $\psi:\mathbb{G}_2\to\mathbb{G}_1$ *mapping* P_2 *to* P_1, *which is simply the trace map restricted to* $\mathbb{G}_2$.

Definition 4 (Type 3) *In this situation, we take* $\mathbb{G}_1=\mathcal{G}_1$ *and* $\mathbb{G}_2=\mathcal{G}_2$, *with generators* $P_1=\mathcal{P}_1$ *and* $P_2=\mathcal{P}_2$. *There is an efficient algorithm to cryptographically hash arbitrary bit strings into* $\mathbb{G}_1$, *and a slightly less efficient algorithm to hash bit strings into* $\mathbb{G}_2$. *However, there is no known efficiently computable group isomorphism* $\psi:\mathbb{G}_2\to\mathbb{G}_1$ *mapping* P_2 *to* P_1.

Definition 5 (Type 4) *In this situation we take* $\mathbb{G}_1=\mathcal{G}_1$, *but we select* $\mathbb{G}_2$ *to be the whole group* $\mathcal{G}$ *which is a group of order* q^2. *As in the Type 2 situation we set* $P_1=\mathcal{P}_1$ *and* $P_2=\frac{1}{k}\mathcal{P}_1+\mathcal{P}_2$. *There is an efficiently computable homomorphism* ψ *from* $\mathbb{G}_2$ *to* $\mathbb{G}_1$ *such that* $\psi(P_2)=P_1$. *Hashing into* $\mathbb{G}_1$ *or* $\mathbb{G}_2$

can be performed, although maybe not very efficiently into $\mathbb{G}_2$. However, one cannot hash efficiently into the subgroup of $\mathbb{G}_2$ generated by P_2. Note, that the pairing of a non-zero element in $\mathbb{G}_1$ and a non-zero element in $\mathbb{G}_2$ may be trivial in this situation.

Hence, in all situations we have that P_1 is the generator of $\mathbb{G}_1$ and P_2 is a fixed element of $\mathbb{G}_2$ of prime order q, such that where there is a computable homomorphism ψ from $\mathbb{G}_2$ to $\mathbb{G}_1$ we have $\psi(P_2) = P_1$. In Type 3 curves, such an isomorphism exists one is just unable to compute it, we will still refer to ψ in this situation but it should be born in mind that one is unable to compute it.

We shall see that a number of efficient key agreement protocols can be implemented in the Type 3 setting, or less efficiently in the Type 2 setting. The rest are implementable only in the Type 1 and Type 4 setting. In the Type 1 setting we have problems due to efficiency as the security parameter increases as we are restricted to supersingular curves. In the Type 4 setting the security proofs become more cumbersome as the image of the hash function into $\mathbb{G}_2$ is not going to be into the group generated by P_2.

We shall refer to the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, the elements P_1 and P_2, the pairing $\hat{e}$, and possibly the homomorphism ψ, as a set of pairing parameters. We assume that given a security parameter one can generate a set of pairing parameters meeting the required security level.

2.2 Subgroup Membership Testing

In almost all key agreement schemes an assumption is made that all values passed from one party to another lie in the correct groups. Such assumptions are often implicit within security proofs. However, one needs to actually check that either given message flows lie within the correct group, or force the messages to lie in the group via additional computation, or by choosing parameters carefully so as the problem does not arise. Indeed some attacks on key agreement schemes, such as the small subgroup attack [20], are possible because implementors do not test for subgroup membership.

For pairing based systems one needs to be careful whether and how one implements these subgroup membership tests as it is not as clear as for standard discrete logarithm based protocols.

Subgroup membership testing in $\mathbb{G}_1 = \mathcal{G}_1$, $\mathbb{G}_T$, $\mathcal{G}$, and $\mathcal{G}_2$ is done in the standard way via multiplication and by inspection of the representation. If the cofactor is smaller than q one can test membership via cofactor multiplication, however in many pairing based situations the cofactor is larger than q in which case membership of the group of exponent q is tested by multiplication by q. Note, that depending on the security parameter this membership test may be quite expensive, as one may need to perform quite a large multiplication.

In the Type 2 and 4 situations there are other subgroup tests may need to be performed, which cannot be performed as above, namely testing whether a given point Q is a multiple of $P_2 = \frac{1}{k}\mathcal{P}_1 + \mathcal{P}_2$. In other words we wish to test whether $Q \in \langle P_2 \rangle$. We first test whether Q has order q by testing, via multiplication as above, whether it is in $\mathcal{G}$. Then we write $Q = a\mathcal{P}_1 + b\mathcal{P}_2$, for unknown a and b; one can compute $a\mathcal{P}_1$ and $b\mathcal{P}_2$ from Q via

$$a\mathcal{P}_1 = \frac{1}{k}\mathrm{Tr}(Q) \text{ and } b\mathcal{P}_2 = Q - a\mathcal{P}_1,$$

which requires one multiplication in $\mathbb{G}_1$. We need to test whether $a = b/k$, which can be done by performing the following test

$$\hat{e}(\mathrm{Tr}(Q), \mathcal{P}_2) = \hat{e}(ka\mathcal{P}_1, \mathcal{P}_2) = \hat{e}(\mathcal{P}_1, b\mathcal{P}_2) = \hat{e}(\mathcal{P}_1, Q - \frac{1}{k}\mathrm{Tr}(Q)).$$

In the Type 4 situation another situation occurs when we wish to test whether a point $Q = a\mathcal{P}_1 + b\mathcal{P}_2$ is a multiple of a point $P = c\mathcal{P}_1 + d\mathcal{P}_2$ without knowing a, b, c or d. We first test whether $P, Q \in \mathcal{G}$ as above. Then we test whether $a = tc$ and $b = td$ for some unknown t by testing whether

$$\hat{e}(\mathrm{Tr}(Q), P - \frac{1}{k}\mathrm{Tr}(P)) = \hat{e}(\mathrm{Tr}(P), Q - \frac{1}{k}\mathrm{Tr}(Q)).$$

In what follows we will implicitly assume within our security proofs that certain subgroup membership testing is performed. This is a common simplifying assumption in the literature but one which is often ignored.

2.3 Hard Problems Based on Pairings

The following bilinear Diffie–Hellman assumption has been used to construct many exciting cryptography schemes including many key agreement protocols. Each problem is assumed to be defined for a given set of pairing parameters.

Assumption 1 (Bilinear Diffie–Hellman (BDH)) *For $a, b, c \in_R \mathbb{Z}_q^*$, given (aP_i, bP_j, cP_k), for some values of $i, j, k \in \{1, 2\}$, computing $\hat{e}(P_1, P_2)^{abc}$ is hard.*

If we wish to make the values of i, j, k explicit we shall refer to this as the $\mathrm{BDH}_{i,j,k}$ problem. In the case of Type 1 pairings, i.e., when $\mathbb{G}_1 = \mathbb{G}_2$, this issue of needing to quantify the BDH problem does not arise. It is trivial to show that the $\mathrm{BDH}_{i,j,k}$ assumption implies the following Diffie–Hellman assumption $\mathrm{DH}_{i,j,k'}$ when $k \neq k'$.

Assumption 2 (Diffie–Hellman (DH)) *For $a, b \in_R \mathbb{Z}_q^*$ and some values of $i, j, k \in \{1, 2\}$, given (aP_i, bP_j), computing abP_k is hard.*

In some schemes, the following variants of the BDH assumption are used, again we can make them depend explicitly on i, j, k if required.

Assumption 3 (Decisional BDH (DBDH)) *For $a, b, c, r \in_R \mathbb{Z}_q^*$, differentiating*

$$(aP_i, bP_j, cP_k, \hat{e}(P_1, P_2)^{abc}) \text{ and } (aP_i, bP_j, cP_k, \hat{e}(P_1, P_2)^{r}),$$

for some values of $i, j, k \in \{1, 2\}$, is hard.

Assumption 4 (Bilinear DH Inversion (ℓ-BDHI)) *For an integer ℓ, and $\alpha \in_R \mathbb{Z}_q^*$, given $(\alpha P_i, \alpha^2 P_i, \ldots, \alpha^\ell P_i)$ for $i \in \{1, 2\}$, computing $\hat{e}(P_1, P_2)^{1/\alpha}$ is hard.*

Certainly, the DBDH and ℓ-BDHI assumptions are stronger than the BDH assumption. The existing security proofs of some protocols make use of some gap assumptions, which mean assuming the existence of an algorithm to resolve a decisional problem, the corresponding computational problem is still hard. In this paper, GBDH, ℓ-GBDHI and ℓ-GBCAA1 respectively stand for the gap BDH assumption, the gap ℓ-BDHI assumption and the gap ℓ-BCAA1 assumption. The ℓ-BCAA1 problem is a variant of ℓ-BDHI, as discussed in [6].

Assumption 5 (Bilinear Collision Attack Assumption (ℓ-BCAA1)) *For an integer ℓ, and $\alpha \in_R \mathbb{Z}_q^*$, given $(\alpha P_i, h_0, (h_1, \frac{1}{h_1+\alpha}P_j), \ldots, (h_\ell, \frac{1}{h_\ell+\alpha}P_j))$ for some values of $i, j \in \{1, 2\}$ where $h_i \in_R \mathbb{Z}_q^*$ and different from each other for $0 \le i \le \ell$, computing $\hat{e}(P_1, P_2)^{1/(\alpha+h_0)}$ is hard.*

In the case where one has a computable homomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$, the existence of the pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, implies that the DH assumption in $\mathbb{G}_2$ is in fact a gap assumption. We can use the pairing to construct an efficient decisional algorithm which given (aP_2, bP_2, cP_2) returns 1 if $\hat{e}(\psi(aP_2), bP_2) = \hat{e}(P_1, cP_2)$, or 0 otherwise.

Whilst a particular scheme may not require the computable homomorphism to implement it, the computable homomorphism may be required in the security proof. Thus for Type 3 curves, where no such isomorphism exists, we are creating a relativised security proof, namely relative to an oracle which can compute ψ. We denote the corresponding relativised hard problem by a superscript-ψ, as in $\mathrm{DH}^{\psi}_{2,2,1}$, $\mathrm{BDH}^{\psi}_{2,1,2}$, ℓ-BDHI^{ψ}_{2}, ℓ-$\mathrm{BCAA1}^{\psi}_{2,1}$ etc.

3 Security Model of Key Agreement

In this work we shall use a modified Bellare-Rogaway key exchange model [2] to analyse the protocol security. In the model, each party involved in a session is treated as an oracle, and an adversary can access the oracle by issuing some specified queries (defined below). An oracle $\Pi^s_{i,j}$ denotes the s-th instance of party i involved with a partner party j in a session

The security of a protocol is defined by a game with two phases. In the first phase, an adversary E is allowed to issue the following queries in any order.

1. *Send*$(\Pi^s_{i,j}, x)$. Upon receiving the message x, oracle $\Pi^s_{i,j}$ executes the protocol and responds with an outgoing message m or a decision to indicate accepting or rejecting the session. If the oracle $\Pi^s_{i,j}$ does not exist, it will be created as initiator if $x = \lambda$, or as a responder otherwise. In this work, we require $i \neq j$, i.e., a party will not run a session with itself. Such restriction is not unusual in practice.
2. *Reveal*$(\Pi^s_{i,j})$. If the oracle has not accepted, it returns $\perp$; otherwise, it reveals the session key.
3. *Corrupt*(i). The party i responds with its private key.

Once the adversary decides that the first phase is over, it starts the second phase by choosing a *fresh oracle* $\Pi^s_{i,j}$ and issuing a $Test(\Pi^s_{i,j})$ query, where the *fresh oracle* $\Pi^s_{i,j}$ and $Test(\Pi^s_{i,j})$ query are defined as follows.

Definition 6 (fresh oracle) *An oracle $\Pi^s_{i,j}$ is fresh if (1) $\Pi^s_{i,j}$ has accepted; (2) $\Pi^s_{i,j}$ is unopened (not being issued the* Reveal *query); (3) party $j \neq i$ is not corrupted (not being issued the* Corrupt *query); (4) there is no opened oracle $\Pi^t_{j,i}$, which has had a matching conversation to $\Pi^s_{i,j}$.*

The above fresh oracle definition is particularly defined to cover the key-compromise impersonation resilience property since it implies that the user i could have been issued a *Corrupt* query.

4. $Test(\Pi^s_{i,j})$. Oracle $\Pi^s_{i,j}$ which is fresh, as a challenger, randomly chooses $b \in \{0, 1\}$ and responds with the session key, if $b = 0$, or a random sample from the distribution of the session key otherwise.

After this point the adversary can continue querying the oracles except that it cannot reveal the test oracle $\Pi^s_{i,j}$ or its partner $\Pi^t_{j,i}$ (if it exists), and it cannot corrupt party j. Finally the adversary outputs a guess b' for b. If $b' = b$, we say that the adversary wins. The adversary's advantage is defined as

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

We use the session ID which can be the concatenation of the messages in a session to define matching conversations, i.e., two oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have *matching conversations* to each other if they have the same session ID.

A secure authenticated key (AK) agreement protocol is defined as follow.

Definition 7 *Protocol Π is a secure AK if:*

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

2. For any polynomial time adversary E, $Adv^E(k)$ is negligible.

It is straightforward to see that when a party will not run a session with itself, if a protocol is secure regarding Definition 6, 7, then it is secure in a weaker security model in which the fresh oracle $\Pi^t_{i,j}$ requires that both party i and j are uncorrupted (such a fresh oracle is used in [2, 3]). If a protocol is secure regarding the above formulation, it achieves implicit mutual key authentication and the following general security properties: known session key security, key-compromise impersonation resilience and unknown key-share resilience [3, 10].

We define another security property: forward secrecy as follow.

Definition 8 *An AK protocol is said to be forward secure if the adversary wins the game with negligible advantage when it chooses as the challenger (i.e., in place of the fresh oracle) an* unopened *oracle $\Pi^s_{i,j}$ which has a matching conversation to an* unopened *oracle $\Pi^t_{j,i}$ and both oracles accepted. If both i and j can be corrupted then the protocol achieves perfect forward secrecy. If in the game the master key can be disclosed, then the protocol achieves master key forward secrecy.*

4 Review on the Existing Schemes

In this section, we briefly review the existing identity-based key agreement protocols, which make use of pairings. We separate them into four categories, dependent on the various message flows needed between two parties A and B for them to establish a shared secret. Each category has one protocol, which indicates the message flows, and one or more schemes, which indicate what kind of secret A and B share at the end of the protocol. We pay particular attention to the difference between the asymmetric and the symmetric pairing setting, and the different types of asymmetric pairings. This is because the resulting efficiency and security reductions of each protocol depend heavily on which type of pairing one is using. When translating schemes into the asymmetric pairing setting we try to ensure that the key agreement scheme is role-symmetric, i.e. that the algorithm (resp. message flows) performed (resp. sent and received) by the initiator and the responder are the same. However, for Type 4 schemes this is often impossible since the trace map and the pairing do not commute. In such situation we present a non-role-symmetric version. In Section 5 we will compare all schemes in the various settings in more detail.

In the reviewed schemes, the shared secret is used by A and B to compute their shared session key as well as in a key confirmation process. To do this, A and B make use of a couple of extra hash-functions. This part is standard and well-known, so we omit it for simplicity. Recent researches have shown that by including the party identifier and the protocol transcript in the computation of the shared session key, a matching conversation, which is used in the Bellare-Rogaway model, can be guaranteed. In the remaining part of this section, we assume that each reviewed scheme makes use of a hash-function, which takes as input a data string including the party identities, the protocol transcript and the shared secret and outputs a session key. Although some of these schemes might not have been originally defined in such a way, it can be modified straightforwardly. Therefore, we only list the shared secret, and do not address how a shared session key is computed in an individual scheme.

The schemes, their security properties and their computational performance in each category are listed in Table 1, 2, 3, and 4 respectively. In these tables, we use the symbols, ✓, $\times$ and -, to indicate respectively that the property holds in the scheme, that the property does not hold and that there is no an acceptable proof to support the judgement. In addition, the security properties are listed with their short names as follows:

- ksk: known session key security, i.e., that the compromise of one session key should not compromise other session keys.
- fs: forward secrecy, i.e., that if long-term private keys of one or more of the entities are compromised, the secrecy of previously established session keys should not be affected. We list the following three cases for different levels of this property:

  - s: the property holds if an adversary gets either A's or B's long-term private key.
  - d: the property holds if an adversary gets both A's and B's long-term private keys.
  - m: the property holds if an adversary gets the KGC master private key.
- kci: key-compromise impersonation resilience, i.e., that compromising an entity A's long-term private key will allow an adversary to impersonate A, but it should not enable the adversary to impersonate other entities to A.
- uks: unknown key-share resilience, i.e., that an entity A should not be able to be coerced into sharing a key with any entity C when in fact A thinks that she is sharing the key with another entity B.

We do not list the property of key control, i.e., that neither entity should be able to force the session key to be a preselected value, in the tables, because all the schemes discussed in this paper hold this property at the same level, as discussed in [24].

We also use the following symbols to explain the computational performance of each scheme (for simplicity, we only count these expensive operations):

- P: pairing.
- S_1: multiplication in $\mathbb{G}_1$.
- S_2: multiplication in $\mathbb{G}_2$.
- E: exponentiation in $\mathbb{G}_T$.

In the shared secret column of each table, the top line details the shared secret in terms of all secrets, and the second line refers to how the shared secret is computed by party A, in the case when the protocol is role-symmetric. In this situation, since parties A and B compute the shared secret in a symmetric way, readers can work out how B does this. In the case where the protocol is not role-symmetric we present the two ways that the key is obtained, first for party A and then for party B.

4.1 Setup and Extract Algorithms

Setup. All the schemes in these four categories use the same setup algorithm to create a KGC master key pair. Given the security parameter k, the algorithm first selects a set of pairing parameters of the correct form.

Two types of pairing-based key extract algorithms have been used in identity-based key agreement schemes. We call them Extract 1 and Extract 2 respectively. The schemes in the first three categories make use of Extract 1, and the schemes in the last category make use of Extract 2.

Extract 1. This algorithm was first proposed by Sakai *et al.* in [28]. It comes in two variants, which are identical in the symmetric pairing setting. We shall refer to the two variants as Extract 1 and Extract 1'. In Extract 1 given the pairing parameters, an identity string ID_A for a user A, a hash-function $H_1 : \{0,1\}^* \to \mathbb{G}_1$, the master private key $s \in \mathbb{Z}_q^*$, and the master public key, either $R = sP_1 \in \mathbb{G}_1$ or $R' = sP_2 \in \mathbb{G}_2$ or both, the algorithm computes $Q_A = H_1(ID_A) \in \mathbb{G}_1$ and $d_A = sQ_A \in \mathbb{G}_1$. Extract 1' is the same, except that H_1 is now a hash function with codomain $\mathbb{G}_2$, and hence Q_A and d_A lie in $\mathbb{G}_2$. In both cases, the values Q_A and d_A will be used as the public and private key pair corresponding to A's identity ID_A.

Extract 2. This algorithm was first proposed by Sakai and Kasahara in [27]. Given the pairing parameters, an identity string ID_A for a user A, a hash-function $H_1 : \{0,1\}^* \to \mathbb{Z}_q^*$, the master private key $s \in \mathbb{Z}_q^*$, and the master public key $R = sP_1 \in \mathbb{G}_1$, the algorithm computes $\alpha = H_1(ID_A) \in \mathbb{Z}_q^*$ and $d_A = \frac{1}{s+\alpha}P_2 \in \mathbb{G}_2$. The values $T_A = \alpha P_1 + R = (s+\alpha)P_1 \in \mathbb{G}_1$ and d_A will be used as the public and private key pair corresponding to A's identity ID_A. There is also a variant of this algorithm referred to Extract 2' in which $R = sP_2 \in \mathbb{G}_2$ and $T_A = \alpha P_2 + R = (s+\alpha)P_2 \in \mathbb{G}_2$ and $d_A = \frac{1}{s+\alpha}P_1 \in \mathbb{G}_1$. Algorithm Extract 2' is only used in a scheme presented in Section 8.

In all cases the value s is kept secret by the KGC, and the master public key R, or R' or (R, R') is made available to every user. Note, that Extract 1, Extract 2 and Extract 2' can be applied in all pairing types, however Extract 1' is unable to be run in the Type 2 situation, and that when run in the Type 4 situation it is not guaranteed that the output of the function H_1 is a multiple of the point P_2. This latter point can cause complications in security proofs in the Type 4 situation where the homomorphism ψ is used to derive the session key. In particular it means that one no longer has role-symmetric protocols mainly due to the fact that the homomorphism and the pairing do not commute. We note, that it is important that the KGC in the Extract 1' executes the expensive cofactor multiplication needed to make sure that Q_A (resp. d_A) lies in the correct subgroup.

In this paper we do not take into account versions of protocols in which the first party executes the Extract 1 protocol to determine the public key of the second party, whilst the second party executes Extract 1'. If such a situation is allowed then one can create Type 3 versions of the protocols below which cannot be otherwise implemented in the Type 3 situation. The adaptations required are left to the reader, as is the checking of the relevant security proofs.

In the protocol specifications which follow, $X \to Y : Z$ stands for that party X sends party Y a message Z.

4.2 Schemes in Category 1

This protocol family was first introduced by Smart in [34]. With their public and private key pairs, (Q_A, d_A) and (Q_B, d_B) computed by the algorithm Extract 1 or Extract 1' in Section 4.1, A and B randomly choose x and y from $\mathbb{Z}_q^*$ respectively and perform the protocol as follows:

$$A \to B : E_A = x\mathcal{P},$$
$$B \to A : E_B = y\mathcal{P},$$

where $\mathcal{P}$ is either P_1 or P_2 depending on the specific protocols, the exact value of $\mathcal{P}$ is listed in Table 1. On completion of the protocol, A and B use one of these schemes listed in Table 1 to compute a shared secret.

4.3 A Scheme in Category 2

This protocol was proposed by Scott in [29]. With their public and private key pairs, (Q_A, d_A) and (Q_B, d_B) computed by the algorithm Extract 1' in Section 4.1, entities A and B randomly choose x and y from $\mathbb{Z}_q^*$ respectively and perform the protocol as follows:

$$A \to B : E_A = \hat{e}(\psi(d_A), Q_B)^x,$$
$$B \to A : E_B = \hat{e}(\psi(Q_A), d_B)^y.$$

On completion of the protocol, A and B use the scheme listed in Table 2 to compute a shared secret. Note, this protocol family is not role-symmetric for Type 4 pairings, as in this situation we have that, in general,

$$\hat{e}(\psi(d_A), Q_B) \neq \hat{e}(\psi(d_B), Q_A).$$

4.4 Schemes in Category 3

This protocol family was first purposed by Chen and Kudla in [7]. With their public and private key pairs, (Q_A, d_A) and (Q_B, d_B) computed by the algorithm Extract 1' in Section 4.1, A and B randomly choose x and y from $\mathbb{Z}_q^*$ respectively and perform the protocol as follows.

$$A \to B : E_A = x\psi(Q_A),$$
$$B \to A : E_B = yQ_B.$$

Schemes	Pairing Type	Extract, Value $\mathcal{P}$	Shared Secret	Security Properties: ksk	fs: s	fs: d	fs: m	kci	uks	Reduction	Performance
Smart-1 [34]	1,2,3,4	1, P_2	$\hat{e}(xQ_B + yQ_A, P_2)^s$ $\hat{e}(xQ_B, R') \cdot \hat{e}(d_A, E_B)$	✓	✓	×	×	✓	✓	GBDH	$2P + S_1 + S_2$
Smart-2 [34]	1,3,4	1', P_1	$\hat{e}(P_1, xQ_B + yQ_A)^s$ $\hat{e}(xR, Q_B) \cdot \hat{e}(E_B, d_A)$	✓	✓	×	×	✓	✓	GBDH	$2P + 2S_1$
SCK-1 [7]*1	1,2,3,4	1, P_2	$xyP_2,\ \hat{e}(yQ_A + xQ_B, P_2)^s$ $xE_B,\ \hat{e}(xQ_B, R') \cdot \hat{e}(d_A, E_B)$	✓	✓	✓	✓	✓	✓	**BDH***2	$2P + S_1 + 2S_2$
SCK-2 [7]*1	1,3,4	1', P_1	$xyP_1,\ \hat{e}(P_1, yQ_A + xQ_B)^s$ $xE_B,\ \hat{e}(xR, Q_B) \cdot \hat{e}(E_B, d_A)$	✓	✓	✓	✓	✓	✓	**BDH***2	$2P + 3S_1$
CJL-1 [11]	1,2,3,4	1, P_2	$\hat{e}(h'yQ_A + h'xQ_B, P_2)^s$ *3 $\hat{e}(h'xQ_B, R') \cdot \hat{e}(h'd_A, E_B)$	✓	✓	✓	✓	✓	✓	-	$2P + 3S_1 + S_2$
CJL-2 [11]	1,3,4	1', P_1	$\hat{e}(P_1, h'yQ_A + h'xQ_B)^s$ *3 $\hat{e}(h'xR, Q_B) \cdot \hat{e}(h'E_B, d_A)$	✓	✓	✓	✓	✓	✓	-	$2P + 4S_1$
Shim [32]*4 *8	1, 4	1', P_2	$\hat{e}(yP_1 + \psi(Q_B), xP_2 + Q_A)^s$ $\hat{e}(\psi(E_B + Q_B), xR' + d_A)$ $\hat{e}(y\psi(R') + \psi(d_B), E_A + Q_A)$	b	r	o	k	e	n	-	 $P + 2S_2$ $P + S_1 + S_2$
SYL [39]*5 *8	1,4	1', P_2	$xyP_2,\ \hat{e}(yP_1 + \psi(Q_B), xP_2 + Q_A)^s$ $xE_B,\ \hat{e}(\psi(E_B + Q_B), xR' + d_A)$ $yE_A,\ \hat{e}(y\psi(R') + \psi(d_B), E_A + Q_A)$	✓	✓	✓	✓	✓	✓	**BDH***6	 $P + 3S_2$ $P + S_1 + 2S_2$
RYY [26] *8	1, 4	1', P_1	$xyP_1,\ \hat{e}(\psi(Q_A), Q_B)^s$ $xE_B,\ \hat{e}(\psi(d_A), Q_B)$ $yE_A,\ \hat{e}(\psi(Q_A), d_B)$	✓	✓	✓	✓	×	✓	-	 $P + 2S_1$ $P + 2S_1$
BMP [5] *8	1, 4	1', P_1	$xyP_1,\ \hat{e}(\psi(Q_A), Q_B)^s$ *7 $xE_B,\ \hat{e}(\psi(d_A), Q_B)$ $yE_A,\ \hat{e}(\psi(Q_A), d_B)$	✓	✓	✓	✓	×	✓	BDH	 $P + 2S_1$ $P + 2S_1$

*1 This scheme is a modification of the Smart scheme [34] by Chen and Kudla [7]. We call it the Smart-Chen-Kudla (SCK) scheme in this paper.
*2 The SCK-1 scheme and the SCK-2 scheme are proved in Section 7 of this paper.
*3 Where $h' = h(x\psi(E_B)) = h(y\psi(E_A))$ and h is a hash-function.
*4 This scheme was broken by Sun and Hsieh in [35].
*5 This scheme is a modification of the Shim scheme [32] by Yuan and Li [39]. We call it the Shim-Yuan-Li (SYL) scheme in this paper.
*6 This scheme is proved in Section 7 of this paper.
*7 As is different from RYY [26], in this scheme the value xyP_1 is used in the shared session key and the value $\hat{e}(\psi(Q_A), Q_B)^s$ is used in the key confirmation. The key confirmation process in this scheme is necessary.
*8 For Type 4 pairings these protocols are not role-symmetric, but they are for Type 1 pairings.

Table 1. The Existing Schemes in Category 1.

Schemes	Pairing Type	Extract Method	Shared Secret	Security Properties: ksk	fs: s	fs: d	fs: m	kci	uks	Reduction	Performance
Scott [29]	1,4	1'	$\hat{e}(\psi(Q_A), Q_B)^{sxy}$ E_B^x E_A^y	-	✓	✓	✓	×	-	-	 $P + 2E$ $P + 2E$

Table 2. The Existing Schemes in Category 2.

Note that the function ψ in E_A provides smaller bandwidth and better performance than the original protocol. In this paper, we call it the optimised version of the original protocol. On completion of the protocol, A and B use one of these schemes listed in Table 3 to compute a shared secret. Again for Type 4 pairings the protocols are not role-symmetric, but they are for Type 1 pairings.

Schemes	Pairing Type	Extract Method	Shared Secret	Security Properties: ksk	fs: s	fs: d	fs: m	kci	uks	Reduction	Performance
CK [7]	1,4	1'	$\hat{e}(\psi(Q_A), Q_B)^{s(x+y)}$	✓	✓	×	×	✓	✓	GBDH	
			$\hat{e}(\psi(d_A), xQ_B + E_B)$								$P + S_1 + S_2$
			$\hat{e}(E_A + y\psi(Q_A), d_B)$								$P + S_1 + S_2$
Wang [37]	1,4	1'	$\hat{e}(\psi(Q_A), Q_B)^{s(x+s_A)(y+s_B)}$ *	✓	✓	✓	×	✓	✓	DBDH	
			$\hat{e}((x + s_A)\psi(d_A), s_B Q_B + E_B)$								$P + 2S_1 + S_2$
			$\hat{e}(s_A\psi(Q_A) + E_A, (y + s_B)d_B)$								$P + S_1 + 2S_2$

$*$ Where $s_A = h(x\psi(Q_A), yQ_B)$ and $s_B = h(yQ_B, x\psi(Q_A))$ and h is a one-way function.

Table 3. The Existing Schemes in Category 3.

4.5 Schemes in Category 4

This protocol family was first proposed by McCullagh and Barreto in [23]. With their public and private key pairs, (T_A, d_A) and (T_B, d_B) computed by the algorithm Extract 2 in Section 4.1, A and B randomly choose x and y from $\mathbb{Z}_q^*$ respectively and perform the protocol as follows.

$$A \to B : E_A = xT_B,$$
$$B \to A : E_B = yT_A.$$

On completion of the protocol, A and B use one of these schemes listed in Table 4 to compute a shared secret. Note, we have assumed in the performance column that $\hat{e}(P_1, P_2)$ is precomputed.

5 Efficiency Considerations

In this section we do a more thorough comparison of the schemes above in terms of computational and bandwidth efficiency. The case of Type 1 pairings are suitable for 80 bit security levels, but for higher security levels their performance degrades quite considerably. On the positive side for Type 1 pairings we do not need to worry about the homomorphism ψ, nor the problem of hashing into $\mathbb{G}_2$, and one can obtain very efficient systems at the 80 bit security level.

We will not consider Type 1 systems further in this section, but will turn to the more complicated issue of comparing the efficiency of the key agreement protocols when using Type 2, 3 or 4 systems. We shall compare our systems at the 128-bit level of security, in which case the most efficient systems will have embedding degree $k = 12$. We assume that in Type 3 systems we use a sextic twist [15] which enables one to obtain greater efficiency for operations in $\mathbb{G}_2$. We also assume that in all cases one uses pairing friendly fields [14], which makes our estimates slightly better than those presented in [13].

We shall compare the protocols in terms of computational efficiency by reference to the cost of a multiplication in $\mathbb{G}_1$. Thus we need to know the relative efficiency of multiplication in $\mathbb{G}_2$ and exponentiation in $\mathbb{G}_T$, and the relative cost of pairings to this base measure. The relative cost of multiplication

Schemes	Pairing Type	Extract Method	Shared Secret	Security Properties						Reduction	Performance
				ksk	fs			kci	uks		
					s	d	m				
MB-1 [23]	1,2,3,4	2	$\hat{e}(P_1,P_2)^{xy}$ $\hat{e}(xE_B,d_A)$	-	✓	✓	×	×	-	-	$P+3S_1$
MB-2 [23]*1	1,2,3,4	2	$\hat{e}(P_1,P_2)^{x+y}$ $\hat{e}(P_1,P_2)^x \cdot \hat{e}(E_B,d_A)$	✓	✓	×	×	✓	✓	ℓ-GBCAA1	$P+E+2S_1$
Xie [38]*2	1,2,3,4	2	$\hat{e}(P_1,P_2)^{xy+x+y}$ $\hat{e}(P_1,P_2)^x \cdot \hat{e}((x+1)E_B,d_A)$	b	r	o	k	e	n	-	$P+E+3S_1$
LYL-1 [22]	1,2,3,4	2	$\hat{e}(P_1,P_2)^{xyh(\hat{e}(P_1,P_2)^x)h(\hat{e}(P_1,P_2)^y)}$ *3 $\hat{e}(E_B,d_A)^{xh(\hat{e}(P_1,P_2)^x)h(\hat{e}(E_B,d_A))}$	-	-	-	×	-	-	-	$P+2E+2S_1$
LYL-2 [22]	1,2,3,4	2	$\hat{e}(P_1,P_2)^{xy}+\hat{e}(P_1,P_2)^{x+y}$ $\hat{e}(E_B,d_A)^x+\hat{e}(P_1,P_2)^x \cdot \hat{e}(E_B,d_A)$	-	-	-	×	-	-	-	$P+2E+2S_1$
MB-1+2	1,2,3,4	2	$\hat{e}(P_1,P_2)^{x+y}, \hat{e}(P_1,P_2)^{xy}$ $\hat{e}(P_1,P_2)^x \cdot \hat{e}(E_B,d_A), \hat{e}(E_B,d_A)^x$	✓	✓	✓	×	✓	✓	ℓ-GBCAA1*4	$P+2E+2S_1$

∗1 The proof is given by Cheng and Chen in [9].
∗2 The scheme was broken by Shim in [33] and by Li, Yuan and Li in [22].
∗3 Where function h maps an element in $\mathbb{G}_T$ to an integer in a specified range.
∗4 The security of this scheme can be proved by following the same method used in [9]. But the perfect forward secrecy property relies on an unusual assumption, i.e., given $(sP_1, (h_1, \frac{1}{h_1+s}P_2), \ldots, (h_\ell, \frac{1}{h_\ell+s}P_2), aP_1, bP_1)$ for $h_i, a, b \in \mathbb{Z}_q^*$, computing $\hat{e}(P_1,P_2)^{\frac{ab}{(h_i+s)(h_j+s)}}$ is hard.

Table 4. The Existing Schemes in Category 4.

in $\mathbb{G}_2$ and exponentiation in $\mathbb{G}_T$ is relatively easy to measure [13], but the relative cost of pairings is harder. We shall assume, following [14], that a pairing in the Type 3 case at the 128-bit security level requires around 20 times the cost of a multiplication in $\mathbb{G}_1$. Using the map ψ and a projecting onto the q-division points of the underlying elliptic curve which are orthogonal to the group $\mathbb{G}_1$, one can perform a pairing in the Type 2 and 4 situations at the same cost as that in the Type 3 situation, bar an extra multiplication in $\mathbb{G}_1$. Thus we are assuming that a Tate pairing is used, as opposed to an Ate-pairing [15].

The fact that Type 2 and Type 4 have similar performance is because the only real difference is in what we consider to be the group $\mathbb{G}_2$. In the Type 2 case it is a general subgroup of the group used in the Type 4 case. In Table 5 we summarise the estimates we will use in what follows.

We now turn to each protocol above in turn and give the computational cost, bandwidth of the message flows and the known security reduction for each in both the Type 2, 3 and 4 settings, at the 128-bit security level. This is summarised in Table 6. An absence in the cost/bandwidth setting of Table 6 implies that that the scheme cannot be implemented in this case. Where a security reduction is unproved in the asymmetric pairing setting, but one is proved in the symmetric setting we signal this by a $\perp$. The reason for marking these differently is that we have not checked these proofs in the asymmetric pairing setting. If a security reduction is marked as relative to an oracle which computes ψ then this reduction is not relative to such an oracle if, as in the Type 1, 2 and 4 situations, there is an efficient algorithm to compute ψ.

The relative speed in Table 6 is the cost of a full execution by one party relative to the cost of a multiplication in $\mathbb{G}_1$. Also in Table 6 we compare the cost of the enhanced CK and MB-2 schemes which are considered later in Section 8. We label these e-CK and e-MB-2 respectively. Note, the security proofs for these enhanced schemes no longer depend on a gap assumption.

We now summarise the table. In the Type 2 setting, if one requires a proof related to a standard hard problem then the most efficient scheme is the SCK-1 scheme. In the Type 3 setting one always requires

	Type 2	Type 3	Type 4
Size of Elements			
$\in \mathbb{G}_1$	256 bits	256 bits	256 bits
$\in \mathbb{G}_2$	3072 bits	512 bits	3072 bits
$\in \mathbb{G}_T$	3072 bits	3072 bits	3072 bits
Relative Cost of Basic Operations			
Multiplication in $\mathbb{G}_1$	1	1	1
Multiplication in $\mathbb{G}_2$	45	3	45
Exponentiation in $\mathbb{G}_T$	3	3	3
Pairing	21	20	21
Hash into $\mathbb{G}_1$	free	free	free
Hash into $\mathbb{G}_2$	n/a	3	540
Relative Cost of Subgroup Membership Test			
$\mathbb{G}_1$	1	1	1
$\mathbb{G}_2$	88	3	45
$\langle P_2 \rangle$	88	3	88
$\langle Q \rangle < \mathcal{G}$	89	3	89
$\mathbb{G}_T$	4	4	4

Table 5. Relative cost of operations and bandwidth at the 128-bit security level, with $k = 12$. These are theoretical relative costs based on multiplication counts only, an actual implementation will have different relative costs.

a reduction to a relativised problem, either a gap assumption or an assumption related to an ψ-oracle. If one prefers a gap assumption, albeit a non-standard one, and does not require having the perfect forward secrecy and/or master key forward secrecy property, then one should use the MB-2 scheme. However, if one prefers an assumption related to an ψ-oracle, then one should adopt the SCK-2 scheme. We note that most security proofs in the literature in the asymmetric pairing setting for Type 3 curves are relative to an ψ-oracle. However, this is often not explicitly stated. In the Type 4 setting the most efficient scheme with a security proof relative to a standard problem is the SCK-2 scheme, if one is willing to accept non-standard problems again no perfect and master key forward secrecy then the MB-2 scheme is the most efficient one.

6 The Proposed Approach

In this section, we discuss the relationship between reveal queries and the security property of key-compromise impersonation resilience. We come to the conclusion that a simulator of any of the key agreement protocols listed in Section 4, which has a goal of solving the computational BDH (or ℓ-BDHI) problem or its computational or decisional variants, cannot deal with reveal queries to certain sessions; otherwise the key-compromise impersonation resilience property does not hold in this protocol. One example is the BMP scheme in [5], which does not hold the key-compromise impersonation resilience property, but can be proved to hold a number of other security properties with reveal queries.

In general, three types of secrets are used in this type of key agreement protocols: the KGC master private key, each party's identity-based private key, which is computed using the master private key and the party's identifier, and each party's ephemeral secret, which is used to compute the party key token. In a protocol, after exchanging the key token with the other party, each party takes as input the master public key, its own identity-based private key and ephemeral secret along with the other party's identifier and the key token, and computes a pairing (or a few pairings) as a session secret shared with the other party.

Scheme	Reduction	Type 2			Type 3			Type 4		
		$\#m$	$m \in$	Relative Speed[*2]	$\#m$	$m \in$	Relative Speed[*2]	$\#m$	$m \in$	Relative Speed[*2]
Smart-1	$\mathrm{GBDH}^{\perp}$	3072	88	88	512	3	44	3072	88	88
Smart-2	$\mathrm{GBDH}^{\perp}$	-	-	-	256	1	42	256	1	44
SCK-1	$\mathrm{BDH}^{\psi}_{2,1,2}$[*1]	3072	88	133	512	3	47	3072	88	133
SCK-2	$\mathrm{BDH}^{\psi}_{2,2,1}$[*1]	-	-	-	256	1	43	256	1	45
CJL-1	-	3072	88	90	512	3	48	3072	88	90
CJL-2	-	-	-	-	256	1	44	256	1	46
SYL	$\mathrm{BDH}^{\perp}$[*1]	-	-	-	-	-	-	3072	89	156/112[*4]
RYY	-	-	-	-	-	-	-	256	1	23
BMP	$\mathrm{BDH}^{\perp}$	-	-	-	-	-	-	256	1	23
Scott	-	-	-	-	-	-	-	1024[*3]	4	27
CK[*5]	$\mathrm{GBDH}^{\perp}$	-	-	-	-	-	-	256/3072[*4]	1/89	67
e-CK	$\mathrm{BDH}^{\perp}$[*1]	-	-	-	-	-	-	3328	90	154/110[*4]
Wang[*5]	$\mathrm{DBDH}^{\perp}$	-	-	-	-	-	-	256/3072[*4]	1/89	68/112[*4]
MB-1	-	256	1	24	256	1	23	256	1	24
MB-2	ℓ-$\mathrm{GBCAA1}^{\perp}$	256	1	26	256	1	25	256	1	26
MB-1+2	ℓ-$\mathrm{GBCAA1}^{\perp}$	256	1	29	256	1	28	256	1	29
e-MB-2	ℓ-$\mathrm{BCCA1}^{\psi}_{2,1}$[*1]	3328	90	157	768	4	70	3328	90	157
LYL-1	-	256	1	29	256	1	28	256	1	29
LYL-2	-	256	1	29	256	1	28	256	1	29

∗1 Proved in this paper.
∗2 The schemes Smart-1, SCK-1 and CJL-1 use the Extract 1 method, whereas Smart-2, SCK-2 and CJL-2, SYL, RYY, BMP, Scott, CK and Wang use the Extract 1' method. However, the cofactor multiplication needed to implement the hash function H_1 in the Extract 1' method can be simplified in all these schemes, by combining the cofactor into the final powering step of the Tate pairing calculation.
∗3 One can apply a form of pairing compression to reduce the bandwidth requirements from 3072 down to about 1024.
∗4 Recall for Type 4 curves these protocols are not role-symmetric either with respect to the message flows or the computation of the shared secret.
∗5 Here we counted the optimised version of CK and Wang protocol.

Table 6. Comparison of schemes in the Type 2, 3 and 4 Settings: The $\#m$ column denotes the size of the message flow in bits. The $m \in$ column denotes the relative cost of testing whether the message received is in the correct subgroup. Relative speed denotes the cost relative to a multiplication in $\mathbb{G}_1$ of the protocol, excluding the subgroup check.

In the security proof of this type of protocols, as defined in Section 3, a simulator of a real protocol has a goal to solve a pairing related hard problem, as defined in Section 2. The security of such a protocol is defined as a game between the simulator (say S) and an adversary (say E), who has a goal to break the protocol.

For example, suppose S's goal is to solve the BDH problem: given (xP, yP, zP), compute $\hat{e}(P,P)^{xyz}$. Algorithm S arranges these three secrets as follows: the master private key is x; the identity-based public key of the attacked party (say I) is related to y; the ephemeral secret of the challenge party (say J) is z.

To answer reveal queries from E, S has to deal with the following different sessions:

– Challenge session($\Pi^s_{I,J}$, $\Pi^t_{J,I}$): E impersonates oracle $\Pi^s_{I,J}$ and challenges oracle $\Pi^t_{J,I}$. S does not need to answer any reveal query to this session, based on the definition of the security model in Section 3.

– Session ($\Pi^s_{I,J}$, $\Pi^t_{J,I}$): E impersonates oracle $\Pi^t_{J,I}$ and asks a reveal query to oracle $\Pi^s_{I,J}$. S is not able to compute the session secret; otherwise the session secret can be computed by using the party ephemeral secret and the partner long-term key, and therefore the key-compromise impersonation resilience property is not held in this protocol.
– Session ($\Pi^s_{I,C}$, $\Pi^t_{C,I}$): E impersonates oracle $\Pi^t_{C,I}$ ($C \notin \{I, J\}$) and asks a reveal query to oracle $\Pi^s_{I,C}$. Again, S is not able to compute the session secret for the same reason as the above session.
– Session ($\Pi^s_{I,C}$, $\Pi^t_{C,I}$): E impersonates oracle $\Pi^s_{I,C}$ and asks a reveal query to oracle $\Pi^t_{C,I}$. S can compute the session secret by following the protocol correctly.
– Session ($\Pi^s_{C,D}$, $\Pi^t_{D,C}$): E impersonates either oracle $\Pi^s_{C,D}$ or $\Pi^t_{D,C}$ and asks a reveal query to the other oracle, where C and D is not I. S can compute the session secret by following the protocol correctly.

As mentioned before, computing the session secret guarantees that S can answer the reveal queries, but is not necessary when the security proof is in the random oracle model. In this case, S controls the random oracle, which takes as input the session secret and outputs a random number as the session key. Although S cannot compute the session secret, S can choose a random number as the answer to a reveal query. The problem is that for some sessions, as Sessions ($\Pi^s_{I,J}$, $\Pi^t_{J,I}$) and ($\Pi^s_{I,C}$, $\Pi^t_{C,I}$) discussed above, S cannot compute a session secret, but E may be able to do so. Therefore, in order to check whether or not S acts as a real protocol, E can query the random oracle with the correct session secret after the reveal query. Since S cannot recognize this correct value, S cannot make the output of the random oracle consistent with the responses to the reveal query.

This tells us that the computational problem can be replaced with the corresponding decisional problem in the random oracle model. As long as S is able to make a right decision to E's random oracle query, the behave of S, from E's point of view, is indistinguishable to the real world. As mentioned in Section 1, researches have tried a number of various ways to solve this problem. Let us take a closer look at them:

Cheng *et al.* in [10] used a coin query to force the adversary to register the ephemeral secret used to generate the key token with the simulator. They heuristically demonstrated that the model with the coin query can address certain attacks which are not covered in the model with the reveal query completely disallowed. On the other hand, they also showed that for some attacks, the adversary may not know the ephemeral value corresponding to the key token used in the attacks. Hence, their model requires a protocol to be analysed in two separate reductions, one with the reveal query disallowed and the other with both the coin query and the reveal query allowed. However, this approach does not guarantee the security of a protocol even if both valid reductions in the model can be constructed.

Kudla and Paterson in [19] proposed a modular proof approach, which introduces a decisional oracle in the security proof. By resorting to this decisional oracle, the simulator can choose a random number to answer a reveal query and maintain all random answers consistent to each other. This approach has been used to prove secure a number of protocols under a gap assumption. However, this approach has to rely on the assumption that such a decisional oracle exists and the gap problem is sound. The former may not be true in this type of protocols, as discussed before, and the later is not as strong as the computational problem.

Wang in [37] proposed an approach, which is opposite to the Kudla and Paterson one. This approach has to resort to a computational oracle and is used to analyse the Wang scheme [37] based on a decisional assumption instead. As in the Kudla and Paterson approach, the problem of relying on an oracle, which nobody knows how to construct using any polynomial algorithm in the real world, also happens in this approach. In addition, while using this approach the simulator has to guarantee that the computational oracle would not be bullied by the adversary to compute the underlying hard problem challenge.

To improve the above solutions, we propose a new approach, which incorporates a built-in decisional function. With this function, the simulator can now take the advantage of the "help" of the adversary either for computing the session secret or for maintaining the consistency of random oracle answers.

Such a built-in decisional function can be constructed by introducing the DH key computation in the computation of the session secret or by introducing the DH exchange in the key tokens in certain ways and verifying the consistence of key tokens via a decisional DH algorithm.

Based on the fact that the DDH problem is not hard because a pairing exists, the simulator in our approach does not need to rely on an outside computational oracle in order to generate the session key to be revealed (as required in the Wang approach), or an outside decisional oracle to keep the consistency between the random oracle queries and the reveal queries (as required in the Kudla and Paterson method), or the knowledge of the adversary ephemeral secret (as required in the Cheng *et al.* approach). As a result of incorporating the built-in decisional function, the security reduction can be constructed on the assumptions that are weakest possible.

Implementation of such a built-in decisional function depends on the individual key agreement protocol. The following are two examples:

1. In some protocols of Category 1 specified in Section 4.2, a DH key exchange has already been performed. Hence, a built-in decisional function is obtained by simply adding the DH key computation into the session secret. For example, Smart's scheme in [34] does not have such a decisional function, so it can only be proved secure under the GBDH assumption. The Smart-Chen-Kudla (SCK) scheme in [7] enhances the Smart scheme by including the DH key value in the session secret. Similarly, the Shim scheme in [32] does not have such a decisional function. This scheme is even vulnerable to the man-in-the-middle attack. As suggested by Yuan and Li in [39], the Shim scheme can be enhanced by adding the DH key value in the session secret. The DH key value along with the key tokens provides a decisional DH function to the simulator. In Section 7, we will give a formal proof of these two schemes. The built-in decisional function in this case costs only one scalar multiplication.
2. In the protocols of Category 3, 4 specified in Section 4.4, 4.5 respectively, we do not have a straightforward method to create such a decisional function, because the key tokens provided by the two players are computed with different bases. We suggest adding an extra DH key exchange in the key tokens. The base of this DH key involves a unique session identifier. The session uniqueness allows the simulator to choose the base individually in each session. This helps the simulator using the adversary input to perform the decisional function. In Section 8, we will give an example of enhancing the CK scheme (called e-CK for short) and another example of enhancing the MB scheme (called e-MB-2 for short). We formally prove the former under the computational BDH assumption and the later under the computational ℓ-BDHI assumption. Note that this built-in decisional function is more expensive than the previous one.

7 Security Proof of Two Schemes in Category 1

In this section we formally analyse the security of two schemes listed in Category 1 of Section 4.2 in the model defined in Section 3. The first one is the Smart-Chen-Kudla (SCK-1) scheme in [7]. The second one is the Shim-Yuan-Li (SYL) scheme in [39]. For completeness, we reprint these two schemes below.

The KGC executes the following **Setup** algorithm:

1. Generates a set of pairing parameters of the required size.
2. Pick a random $s \in \mathbb{Z}_q^*$ as the master key and compute $R = sP_2$.
3. Pick two cryptographic hash functions as follows:

$$H_1 : \{0,1\}^* \to \mathbb{G}_1(\text{resp. } \mathbb{G}_2),$$
$$H_2 : \{0,1\}^* \times \{0,1\}^* \times \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{G}_T \to \{0,1\}^n$$

for some integer $n > 0$. The codomain of H_1 is $\mathbb{G}_1$ for the SCK-1 protocol and $\mathbb{G}_2$ for the SYL protocol.

The KGC keeps the master key as a secret and publishes other parameters. For any user with an identity $ID \in \{0,1\}^*$, the KGC executes the Extract algorithm to compute $Q_{ID} = H_1(ID)$, $d_{ID} = sQ_{ID}$ and passes d_{ID} as the private key to this user via some secure channel.

In the protocols party A and B randomly choose x and y from $\mathbb{Z}_q^*$ respectively and perform the protocol as follows:

$$A \to B : E_A = xP_2,$$
$$B \to A : E_B = yP_2.$$

On completion of the protocol, A and B use one of following schemes to compute a session secret shared between them.

The SCK-1 scheme: A computes $K = \hat{e}(xQ_B, R) \cdot \hat{e}(d_A, E_B)$ and $xE_B = xyP_2$, and B computes $K = \hat{e}(yQ_A, R) \cdot \hat{e}(d_B, E_A)$ and $yE_A = xyP_2$. The session key is computed by $SK = H_2(A, B, E_A, E_B, xyP_2, K)$.

The SYL scheme: A computes $K = \hat{e}(\psi(E_B + Q_B), xR + d_A)$ and $xE_B = xyP_2$, and B computes $K = \hat{e}(\psi(yR + d_B), E_A + Q_A)$ and $yE_A = xyP_2$. The session key is computed as $SK = H_2(A, B, E_A, E_B, xyP_2, K)$.

The security of the SCK-1 scheme can be summarised by Theorem 1, 2.

Theorem 1 *The SCK-1 scheme is a secure AK, provided the* $BDH^{\psi}_{2,1,2}$ *assumption holds and the hash functions are modelled as random oracles. Specifically, suppose in the attack, an adversary* B *which makes* q_i *queries to* H_i *for* $i = 1, 2$ *and creates* q_o *oracles, wins the game with advantage* $\epsilon(k)$*. Then there exists an algorithm* A *to solve the* $BDH^{\psi}_{2,1,2}$ *problem with advantage*

$$Adv_A^{BDH^{\psi}_{2,1,2}}(k) \geq \frac{1}{q_1 \cdot q_o \cdot q_2}\epsilon(k).$$

Notice, that the proof is relative to an oracle which computes the isomorphism. In pairing parameter instances where such an isomorphism exists this is equivalent to the $\mathrm{BDH}_{2,1,2}$ problem. However, one can implement the SCK-1 scheme for pairing parameters which do not have an explicitly computable isomorphism.

Proof: We define the session ID as the concatenation of $xP_2 \| yP_2$. The first condition in Definition 7 is trivial to prove. Now we prove that the protocol meets the second condition.

Given a $\mathrm{BDH}^{\psi}_{2,1,2}$ problem instance (aP_2, bP_1, cP_2), we construct an algorithm A using the adversary B against the protocol to solve the $\mathrm{BDH}^{\psi}_{2,1,2}$ problem.

A simulates the system setup to adversary B as follows. The system public parameters are defined to be the pairing parameters of the input problem. The master public key is set to be $R = aP_2$, hence A does not know the master secret key. The functions H_1 and H_2 are instantiated as random oracles under the control of A.

Algorithm A randomly chooses $1 \leq I \leq q_1$ and $1 \leq J \leq q_o$ and starts simulating the real world where the adversary B launches the attack. Algorithm A answers the following queries, which are asked by adversary B in an arbitrary order. We shall slightly abuse the notation $\Pi^t_{i,j}$ to refer to the t-th party instance among all the party instances created in the attack, instead of the t-th instance of party i. This would not affect the soundness of the security model.

– $H_1(ID_i)$: Algorithm A maintains an initially empty list H_1^{list} with entries of the form (ID_i, Q_i, ℓ_i). The algorithm A responds to the query in the following way.

  - If ID_i already appears on H_1^{list} in a tuple (ID_i, Q_i, ℓ_i), then A responds with $H_1(ID_i) = Q_i$.
  - Otherwise, if ID_i is the I-th unique identifier query, then A inserts $(ID_i, bP_1, \perp)$ into the list and returns bP_1 (hence the private key of ID_I should be abP_1 which is not known by the algorithm A).
  - Otherwise, A randomly chooses $\ell_i \in \mathbb{Z}_q^*$, inserts $(ID_i, \ell_i P_1, \ell_i)$ into the list and returns $\ell_i P_1$.
- $H_2(ID_u^a, ID_u^b, X_u, Y_u, Z_u, K_u)$: Algorithm A maintains an initially empty list H_2^{list} with entries of the form $(ID_u^a, ID_u^b, X_u, Y_u, Z_u, K_u, h_u)$. The algorithm A responds to the query in the following way (for easily following the reduction, we suggest one reading the Send and Reveal query first).
  - If a tuple indexed by $(ID_u^a, ID_u^b, X_u, Y_u, Z_u, K_u)$ is on the list, then A responds with h_u.
  - Otherwise, A goes through the list $\mathcal{L}$ (maintained in the **Reveal** query) to find a tuple with values $(ID_u^a, ID_u^b, X_u, Y_u, \Pi_{i,j}^t)$ and proceeds as follows (in the following, without loosing generality, we assume Y_u is the message generated by oracle $\Pi_{i,j}^t$, so X_u is the incoming message to $\Pi_{i,j}^t$. By the behavior of Send query, $Y_u = f_{i,j}^t aP_2$).
    - Test if $\hat{e}(\psi(Y_u), X_u) = \hat{e}(P_1, Z_u)$ holds and for Type 2 or 4 pairings test if $\hat{e}(\psi(X_u), Y_u) = \hat{e}(\psi(Z_u), P_2)$ holds as well. If the equations hold, A then does the following:
      - Find the values $f_{i,j}^t$ and $SK_{i,j}^t$ corresponding to oracle $\Pi_{i,j}^t$ from the list Ω maintained in the **Send** query.
      - Find the value ℓ_j from H_1^{list} for party with ID_j.
      - Compute the shared secret via the following equation

$$\begin{aligned} K_{i,j}^t &= \hat{e}(x_i Q_j, R) \cdot \hat{e}(d_i, X_u) \\ &= \hat{e}(f_{i,j}^t a\ell_j P_1, aP_2) \cdot \hat{e}(abP_1, X_u), \text{ since } x_i = f_{i,j}^t a, Q_j = \ell_j P_1, d_i = abP_1 \\ &= \hat{e}(f_{i,j}^t \ell_j \psi(aP_2), aP_2) \cdot \hat{e}(bP_1, \frac{1}{f_{i,j}^t} Z_u), \text{ since } f_{i,j}^t aX_u = Z_u. \end{aligned}$$

        Note that $\Pi_{i,j}^t$ is put on the list $\mathcal{L}$ in the **Reveal** query only when $\Pi_{i,j}^t$ has been revealed and $d_i = abP_1$, but $H_2(ID_u^a,\ ID_u^b,\ X_u,\ Y_u,\ Z_u,\ K_{i,j}^t)$ had not been queried before the reveal query. So, $SK_{i,j}^t$ has been randomly sampled.
      - Set $h_u = SK_{i,j}^t$.
      - Remove $(ID_u^a, ID_u^b,\ X_u, Y_u, \Pi_{i,j}^t)$ from the list $\mathcal{L}$. Put $(ID_u^a, ID_u^b,\ X_u, Y_u, Z_u, K_{i,j}^t, h_u)$ in the list H_2^{list}.
      - Check if $K_{i,j}^t = K_u$. If it is not true, A randomly chooses new $h_u \in \{0,1\}^n$, inserts $(ID_u^a, ID_u^b,\ X_u,\ Y_u,\ Z_u,\ K_u, h_u)$ into the list H_2^{list}.
      - Return h_u.
    - Otherwise (no tuple on $\mathcal{L}$ meets the test), algorithm A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID_u^a, ID_u^b,\ X_u,\ Y_u,\ Z_u,\ K_u, h_u)$ into the list and returns h_u
  - Otherwise (no relative tuple on $\mathcal{L}$ is found), A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID_u^a, ID_u^b,$ $X_u,\ Y_u,\ Z_u,\ K_u, h_u)$ into the list and returns h_u.
- **Corrupt(ID_i)**: A looks through list H_1^{list}. If ID_i is not on the list, A queries $H_1(ID_i)$. A checks the value of ℓ_i: if $\ell_i \neq \perp$, then A responds with $\ell_i \psi(aP_2) = \ell_i aP_1$; otherwise, A aborts the game (**Event 1**).
- **Send($\Pi_{i,j}^t$, M)**: A maintains a list Ω for each oracle of the form $(\Pi_{i,j}^t, tran_{i,j}^t, r_{i,j}^t, K_{i,j}^t, SK_{i,j}^t, f_{i,j}^t)$ where $tran_{i,j}^t$ is the transcript of the oracle so far; $r_{i,j}^t$ is the random integer used by the oracle to generate message, $f_{i,j}^t$ is used for special purpose explained below, and $K_{i,j}^t$ and $SK_{i,j}^t$ are set $\perp$ initially. Note that the Ω list can be updated in other queries as well, such as the reveal query and the H_2 query. A proceeds in the following way:
  - If M is the second message on the transcript, do nothing but simply accept the session. Otherwise,
  - Query $H_1(ID_i)$ and $H_1(ID_j)$.
  - If $t = J$,

  - $*$ If $\ell_j \neq \perp$, then abort the game (**Event 2**).
  - $*$ Otherwise, respond with cP_2 and set $r^t_{i,j} = \perp$ (if $M = \lambda$, then party ID_i is an initiator, otherwise a responder as M is the first message of the session).
- Otherwise,
  - $*$ If $\ell_i = \perp$, randomly choose $f^t_{i,j} \in \mathbb{Z}^*_q$ and respond with $f^t_{i,j}aP_2$ and set $r^t_{i,j} = \perp$.
  - $*$ Otherwise, randomly choose $r^t_{i,j} \in \mathbb{Z}^*_q$ and respond with $r^t_{i,j}P_2$.

– **Reveal($\Pi^t_{i,j}$)**: A maintains a list $\mathcal{L}$ with tuples of the form $(ID_i, ID_j, X_i, Y_j, \Pi^t_{i,j})$, A proceeds in the following way to respond:
- Get the tuple of oracle $\Pi^t_{i,j}$ from Ω.
- If oracle $\Pi^t_{i,j}$ has not accepted, then respond with $\perp$.
- If $t = J$ or if the J-th oracle has been generated as $\Pi^J_{a,b}$ and $ID_a = ID_j, ID_b = ID_i$ and $\Pi^J_{a,b}$ and $\Pi^t_{i,j}$ have the same session ID, then abort the game (**Event 3**).
- If $SK^t_{i,j} \neq \perp$, return $SK^t_{i,j}$.
- Otherwise,
  - $*$ If $r^t_{i,j} \neq \perp$ (so $\ell_i \neq \perp$ and $d_i = \ell_i aP_1 = \ell_i\psi(aP_2)$),
    - · Compute $K^t_{i,j} = \hat{e}(r^t_{i,j}Q_j, aP_2)\cdot\hat{e}(\ell_i\psi(aP_2), M)$ where Q_j is found from H_1^{list} for identifier ID_j and M is the received message on $tran^t_{i,j}$. By making an H_2 query, set $SK^t_{i,j} = H_2(ID_i, ID_j, r^t_{i,j}P_2, M, r^t_{i,j}M, K^t_{i,j})$ if $\Pi^t_{i,j}$ is an initiator oracle, or $SK^t_{i,j} = H_2(ID_j, ID_i, M, r^t_{i,j}P_2, r^t_{i,j}M, K^t_{i,j})$ otherwise, and update Ω by putting $SK^t_{i,j}$ then return $SK^t_{i,j}$ as the response.
  - $*$ Otherwise, i.e., it should have $r^t_{i,j} = f^t_{i,j}a$ and $d_i = abP_1$. Algorithm A does not know both values and should not be able to compute $K^t_{i,j} = \hat{e}(f^t_{i,j}aQ_j, aP_2)\cdot\hat{e}(abP_1, M)$ and $f^t_{i,j}aM$ (note that the model requires that $i \neq j$). Algorithm A proceeds as follows:
    - · Go through the list H_2^{list} to find a tuple $(ID_i, ID_j, f^t_{i,j}aP_2, M, Z_u, K_u, h_u)$ if ID_i is the initiator or a tuple $(ID_j, ID_i, M, f^t_{i,j}aP_2, Z_u, K_u, h_u)$ otherwise, meeting the equation $\hat{e}(\psi(f^t_{i,j}aP_2), M) = \hat{e}(P_1, Z_u)$ and for Type 2 or 4 pairings $\hat{e}(\psi(M), f^t_{i,j}aP_2) = \hat{e}(\psi(Z_u), P_2)$ as well.
    - · If such Z_u is found, then compute
$$\begin{aligned}K^t_{i,j} &= \hat{e}(f^t_{i,j}aQ_j, R)\cdot\hat{e}(d_i, M)\\ &= \hat{e}(f^t_{i,j}a\ell_jP_1, aP_2)\cdot\hat{e}(abP_1, M),\ \text{since } d_i = abP_1, Q_j = \ell_jP_1\\ &= \hat{e}(f^t_{i,j}\ell_j\psi(aP_2), aP_2)\cdot\hat{e}(bP_1, \frac{1}{f^t_{i,j}}Z_u)\ \text{since } f^t_{i,j}aM = Z_u.\end{aligned}$$
      and set $SK^t_{i,j} = H_2(ID_i, ID_j, f^t_{i,j}aP_2, M, Z_u, K^t_{i,j})$ if oracle $\Pi^t_{i,j}$ is the initiator or $SK^t_{i,j} = H_2(ID_j, ID_i, M, f^t_{i,j}aP_2, Z_u, K^t_{i,j})$ otherwise.
    - · Otherwise, randomly sample $SK^t_{i,j} \in \{0,1\}^n$, put $(ID_i, ID_j, f^t_{i,j}aP_2, M, \Pi^t_{i,j})$ if ID_i is the initiator or $(ID_j, ID_i, M, f^t_{i,j}aP_2, \Pi^t_{i,j})$ into list $\mathcal{L}$.
    - · A responds with $SK^t_{i,j}$ and updates Ω by putting $SK^t_{i,j}$.

– **Test($\Pi^t_{i,j}$)**: If $t \neq J$ or ($t = J$ but) there is an oracle $\Pi^w_{j,i}$ with the same session ID with $\Pi^t_{i,j}$ that has been revealed, then A aborts the game (**Event 4**) (note that according to the rules of the game, $\Pi^t_{i,j}$ should have accepted, i.e., there are two messages on the oracle's transcript, so the check can be done properly). Otherwise ($\ell_j = \perp, Q_j = bP_1$ and $r^t_{i,j} = \perp$), A randomly chooses $\zeta \in \{0,1\}^n$ and responds to B with ζ.

Once B finishes queries and returns its guess, A proceeds with the following steps:

– Compute
$$D = \hat{e}(\ell_i\psi(aP_2), M),$$

where M is the incoming message of $\Pi_{i,j}^J$ and note that because $i \neq j$ according to Definition 6, it has $d_i = \ell_i a P_1 = \ell_i \psi(aP_2)$ where $\ell_i \neq \perp$ found from H_1^{list} corresponding to identifier ID_i of $\Pi_{i,j}^J$, and

$$\begin{aligned} K_{i,j}^t &= \hat{e}(cbP_1, aP_2) \cdot \hat{e}(\ell_i \psi(aP_2), M) \\ &= \hat{e}(P_1, P_2)^{abc} \cdot D \end{aligned}$$

– Algorithm A randomly chooses K_ℓ from H_2^{list} and returns K_ℓ / D as the response to the BDH challenge.

Claim 1 *If A did not abort the game, B could not find inconsistence between the simulation and the real world.*

Proof: The simulations of all the random oracles are valid and the messages of the oracles are uniformly distributed in the message space. Particularly, the simulator makes use of the programmability of random oracle H_2 and the pairing as the decisional algorithm of DH on $\mathbb{G}_2$ to keep the consistence of responses to the H_2 queries and the Reveal queries. So the adversary should not notice any difference from the real attack environment.

Claim 2 *Let* **Event 5** *be that $K = \hat{e}(cbP_1, aP_2) \cdot \hat{e}(\ell_i \psi(aP_2), M)$ was not queried on H_2. Then* $\Pr[\overline{\texttt{Event 5}}] \geq \epsilon(k)$.

Proof: Because H_2 is a random oracle, if `Event 5` happens (i.e., K for the challenge oracle is not queried on H_2), B could win the game only in two ways: (a) B random guesses whether ζ is $SK_{i,j}^J$ or not, if $SK_{i,j}^J$ has not been decided yet. (b) B *knows* it has revealed an oracle $\Pi_{a,b}^w$ which has the same session key as the challenge oracle $\Pi_{i,j}^J$. We note here that `Event 5` could happen in case (b) *in general*, because we make use of the programmability of random oracle H_2 and A may have responded to the Reveal query without knowing the corresponding K in the simulation. Because the random oracle should respond uniquely for each query in the simulation, the adversary can surely win the game if it knows the session key $SK_{i,j}^J$, even if it is not generated by a query to the random oracle H_2.

Because the identifiers and the transcript are used as the inputs of H_2 to generate the session key, $\Pi_{a,b}^w$ has the same session key with $\Pi_{i,j}^t$ with probability greater than $1/2^n$ only if $\Pi_{a,b}^w$ is either the chosen fresh oracle $\Pi_{i,j}^t$ or an oracle $\Pi_{j,i}^w$ which has the same transcript of $\Pi_{i,j}^t$ for some w. While by the rules of the game, B is not allowed to reveal either of the two oracles (the fresh test oracle or its partner with matching conversation should not be revealed). So in this protocol, because the identifiers and the transcript are part of the input of H_2, case (b) happens with probability at most $1/2^n$. Then we have

$$\Pr[B \text{ wins} \mid \texttt{Event 5}] \leq 1/2.$$

Then

$$\begin{aligned} \epsilon(k) + 1/2 = \Pr[B \text{ wins}] &= \Pr[B \text{ wins} \mid \texttt{Event 5}] \Pr[\texttt{Event 5}] \\ &\quad + \Pr[B \text{ wins} \mid \overline{\texttt{Event 5}}] \Pr[\overline{\texttt{Event 5}}] \\ &\leq 1/2 + \Pr[\overline{\texttt{Event 5}}]. \end{aligned}$$

The claim follows.

Let **Event 6** be that, in the attack, adversary B indeed chose oracle $\Pi_{i,j}^J$ as the challenger oracle where ID_j was queried on H_1 as the I-th distinct identifier query. Then following the rules of the game defined in Section 3, it's clear that **Event 1, 2, 3, 4** would not happen and the game won't abort. So,

$$\Pr[\overline{(\texttt{Event 1} \vee \texttt{Event 2} \vee \texttt{Event 3} \vee \texttt{Event 4})}] = \Pr[\texttt{Event 6}] \geq \frac{1}{q_1 \cdot q_o}.$$

Let **Event 7** be that A found the correct K_ℓ. Overall, we have

$$\begin{aligned}\Pr[A \text{ wins}] &= \Pr[\texttt{Event 6} \wedge \overline{\texttt{Event 5}} \wedge \texttt{Event 7}] \\ &\geq \tfrac{1}{q_1 \cdot q_o \cdot q_2} \Pr[\overline{\texttt{Event 5}}] \\ &\geq \tfrac{1}{q_1 \cdot q_o \cdot q_2} \epsilon(k).\end{aligned}$$

This concludes the proof. □

Theorem 2 *The SCK-1 scheme has master key forward secrecy, provided the $DH^{\psi}_{2,2,2}$ assumption is sound and H_2 is modelled as random oracle. Specifically, suppose an adversary B wins the game with advantage $\epsilon(k)$. Then there exists an algorithm A to solve the $DH^{\psi}_{2,2,2}$ problem with advantage*

$$Adv_A^{DH^{\psi}_{2,2,2}} \geq \frac{1}{2}\epsilon(k).$$

Again our comment with respect to the computability of the function ψ holds.

Proof: Given a set of pairing parameters and a $\mathrm{DH}^{\psi}_{2,2,2}$ problem instance (aP_2, bP_2), we construct an algorithm A to make use of B to solve the $\mathrm{DH}^{\psi}_{2,2,2}$ problem. Algorithm A simulates the system setup to adversary B as follow, by randomly sampling $s \in \mathbb{Z}_q^*$ and setting the master public key to be $R = sP_2$, and the master secret key as s. The hash function H_2 will be modelled as a random oracle under the control of A, and H_1 will be a cryptographic hash function. Moreover, the master secret key s is passed to B as well, so A no longer simulates the corrupt query.

As in Theorem 1, we use $\Pi^t_{i,j}$ to represent the t-th one among all oracles created in the attack. Again algorithm A answers the following queries, which are asked by adversary B in an arbitrary order.

- $H_2(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K_u)$: Algorithm A maintains an initially empty list H_2^{list} with entries of the form $(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K_u, h_u)$. A responds to the query in the following way.
  - If a tuple indexed by $(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K_u)$ is on the list, then A responds with h_u.
  - Otherwise, A goes through the list $\mathcal{L}$ (maintained in the **Reveal** query) with tuples of the form $(ID_i, ID_j,\ X_i,\ Y_j, K^t_{i,j}, \Pi^t_{i,j})$ to find a tuple with values $(ID^a_u, ID^b_u, X_u, Y_u, K_u, \Pi^t_{i,j})$ and proceeds as following (again, we assume X_u is the incoming message to $\Pi^t_{i,j}$):
    - ∗ Test if $\hat{e}(\psi(Y_u), X_u) = \hat{e}(P_1, Z_u)$ and for Type 2 or 4 pairing test $\hat{e}(\psi(X_u), Y_u) = \hat{e}(\psi(Z_u), P_2)$ as well. If the equation hold then,
      - · Find the value $SK^t_{i,j}$ from the list Ω.
      - · Remove $(ID^a_u, ID^b_u,\ X_u, Y_u, K_u, \Pi^t_{i,j})$ from the list $\mathcal{L}$. Put $(ID^a_u,\ ID^b_u,\ X_u, Y_u, Z_u,\ K_u,\ SK^t_{i,j})$ in the list H_2^{list} and return $SK^t_{i,j}$. Note that $\Pi^t_{i,j}$ is put on the list $\mathcal{L}$ only when it has been revealed, so $SK^t_{i,j}$ has been sampled.
    - ∗ Otherwise (no tuple on $\mathcal{L}$ meets the test), algorithm A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID^a_u, ID^b_u,\ X_u,\ Y_u,\ Z_u,\ K_u, h_u)$ into the list and returns h_u
  - Otherwise, A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID^a_u, ID^b_u,\ X_u,\ Y_u,\ Z_u,\ K_u, h_u)$ into the list and returns h_u.
- **Send($\Pi^t_{i,j}$, M)**: A maintains a list Ω for each oracle of the form $(\Pi^t_{i,j}, tran^t_{i,j},\ f^t_{i,j},\ K^t_{i,j},\ SK^t_{i,j},\ c^t_{i,j})$ where $tran^t_{i,j}$ is the transcript of the oracle so far; $f^t_{i,j}, c^t_{i,j}$ are used for special purpose explained below, and $K^t_{i,j}, SK^t_{i,j}$ are set $\perp$ initially. This list is updated in the send query as well as in the reveal query and H_2 query. A proceeds in the following way:
  - If M is not the second message on the transcript,
    - ∗ Randomly sample $f^t_{i,j} \in \mathbb{Z}_q^*$.
    - ∗ Randomly flip $c^t_{i,j} \in \{0,1\}$. If $c^t_{i,j} = 0$, set $V = f^t_{i,j}aP_2$, else $V = f^t_{i,j}bP_2$. If $V = P_2$, then responds to the DH challenge with $\frac{1}{f^t_{i,j}}bP_2$ if $c^t_{i,j} = 0$, or $\frac{1}{f^t_{i,j}}aP_2$ otherwise (**Event 1**).

  * If $M \neq \lambda$, compute

$$K_{i,j}^t = \hat{e}(f_{i,j}^t H_1(ID_j), saP_2) \cdot \hat{e}(sH_1(ID_i), M),$$

  if $c_{i,j}^t = 0$, else set

$$K_{i,j}^t = \hat{e}(f_{i,j}^t H_1(ID_j), sbP_2) \cdot \hat{e}(sH_1(ID_i), M)$$

  and accept the session.
  * Return V.
- Otherwise, compute $K_{i,j}^t = \hat{e}(f_{i,j}^t H_1(ID_j), saP_2) \cdot \hat{e}(sH_1(ID_i), M)$ if $c_{i,j}^t = 0$, else compute $K_{i,j}^t = \hat{e}(f_{i,j}^t H_1(ID_j), sbP_2) \cdot \hat{e}(sH_1(ID_i), M)$, and accept the session.

– **Reveal($\Pi_{i,j}^t$)**: Algorithm A maintains a list $\mathcal{L}$ with tuples of the form $(ID_i, ID_j, X_i, Y_j, K_{i,j}^t, \Pi_{i,j}^t)$. The algorithm A proceeds in the following way to respond:
- Get the tuple of oracle $\Pi_{i,j}^t$ from Ω.
- If $\Pi_{i,j}^t$ has not accepted, return $\perp$.
- If the Test($\Pi_{a,b}^w$) query has been issued and if $\Pi_{i,j}^t = \Pi_{a,b}^w$, or $ID_a = ID_j$ and $ID_b = ID_j$ and two oracles have the same session ID, then disallow the query (this should not happen if the adversary obey the rules of the game).
- If $SK_{i,j}^t \neq \perp$, return $SK_{i,j}^t$.
- Otherwise,
  * Go through the list H_2^{list} to find a tuple $(ID_i, ID_j, M_i, M_j, Z_u, K_{i,j}^t, h_u)$ if ID_i is the initiator or a tuple $(ID_j, ID_i, M_j, M_i, Z_u, K_{i,j}^t, h_u)$ otherwise, meeting the equation $\hat{e}(\psi(M_i), M_j) = \hat{e}(P_1, Z_u)$ and for Type 2 or 4 pairing $\hat{e}(\psi(M_j), M_i) = \hat{e}(\psi(Z_u), P_2)$ as well, where M_i and M_j are the messages of party i and j in $tran_{i,j}^t$.
  * If such Z_u is found, then return $SK_{i,j}^t = h_u$.
  * Otherwise, randomly sample $SK_{i,j}^t \in \{0,1\}^n$, put $(ID_i, ID_j, M_i, M_j, K_{i,j}^t, \Pi_{i,j}^t)$ if ID_i is the initiator or $(ID_j, ID_i, M_j, M_i, K_{i,j}^t, \Pi_{i,j}^t)$ into list $\mathcal{L}$. A responds with $SK_{i,j}^t$ and puts $SK_{i,j}^t$ into Ω.

– **Test($\Pi_{i,j}^t$)**: By the rule of the game, there is a partner oracle $\Pi_{j,i}^w$ with the same session ID with $\Pi_{i,j}^t$ and both should not be revealed. A proceeds as follows:
- Check if $c_{i,j}^t = c_{j,i}^w$. If it is true, then abort the game (**Event 2**).
- Otherwise, without loosing generality, we assume $c_{i,j}^t = 0$ and $c_{j,i}^w = 1$, i.e., $M_i = f_{i,j}^t aP_2$ and $M_j = f_{j,i}^w bP_2$. A randomly chooses $\zeta \in \{0,1\}^n$ and responds to B with ζ.

Once B finishes queries and returns its guess, A proceeds with the following steps:

– For every pair (X_u, Y_u, Z_u) on H_2^{list} with $X_u = M_i, Y_u = M_j$ if the tested oracle $\Pi_{i,j}^t$ is an initiator oracle, otherwise with $X_u = M_j, Y_u = M_i$, check if $\hat{e}(\psi(X_u), Y_u) = \hat{e}(P_1, Z_u)$ and for Type 2 or 4 pairings, $\hat{e}(\psi(X_u), Y_u) = \hat{e}(\psi(Z_u), P_2)$ as well hold (M_i and M_j are found in $tran_{i,j}^t$). If no such Z_u meets the equation, abort the game (**Event 3**).

– Otherwise, return $\frac{1}{f_{i,j}^t \cdot f_{j,i}^w} Z_u$ as the response to the DH challenge.

Following similar arguments as in Theorem 1, we have following two claims:

Claim 3 *If A did not abort the game, B could not find inconsistence between the simulation and the real world.*

Claim 4 $\Pr[\overline{\texttt{Event 3}}] \geq \epsilon(k)$.

As $\Pr[\overline{\texttt{Event 2}}] = 1/2$, we have

$$\begin{aligned}\Pr[A \text{ wins}] &= \Pr[\texttt{Event 1} \vee (\overline{\texttt{Event 2}} \wedge \overline{\texttt{Event 3}})] \\ &\geq \epsilon(k)/2.\end{aligned}$$

□

A few subtle points of the proofs are worthy of mentioning. First, the proofs need an extra operation, which is not explicitly specified in the protocol. Namely the proofs assume that all values lie in the correct subgroups. One could ask whether this is an artifact of either the proofs or the security model. We discuss this issue further in Appendix A. Second, we assume that xE_B (yE_A resp.) has a unique representation. This assumption has no significance in practice but is needed to keep the indistinguishability of the proofs from the real world. Third, for Type 4 pairings, there is negligible probability ($1/q$) that the pairing is trivial. We ignored this issue in the reduction.

We note that the above proofs can be replicated for the SCK-2 with Type 1 or 3 pairings based on assumption $\mathrm{BDH}^{\psi}_{2,2,1}$ and $\mathrm{DH}^{\psi}_{2,2,1}$ respectively with very little changes. The reduction for Type 4 pairing is more involved notationally, but the essential proof technique remains the same. We do not give these second proofs in this paper due to lack of space.

We now turn to considering security proofs for the SYL scheme. Note that the SYL scheme can only be implemented when the isomorphism exists and hashing in $\mathbb{G}_2$ can be done efficiently. So the SYL scheme works only with Type 1 and Type 4 pairings. Here we formally present the proof for Type 1 pairing and leave the details of proof for Type 4 pairings to the interested readers.

The security of the SYL scheme can be summarised by Theorem 3, 4.

Theorem 3 *The SYL scheme is a secure AK, provided the BDH assumption is sound and the hash functions are modelled as random oracles. Specifically, suppose in the attack, an adversary B which makes q_i queries to H_i for $i = 1, 2$ and creates q_o oracles, wins the game with advantage $\epsilon(k)$. Then there exists an algorithm A to solve the BDH problem with advantage*

$$Adv_A^{BDH}(k) \geq \frac{1}{q_1 \cdot q_o \cdot q_2}\epsilon(k).$$

Proof: Given a BDH problem instance (P, aP, bP, cP), we construct an algorithm A using the adversary B against the protocol to solve the BDH problem.

A simulates the system setup to adversary B as follows. The system public parameters are defined to be the pairing parameters of the input problem. The master public key is set to be $R = aP$, hence A does not know the master secret key. The functions H_1 and H_2 are instantiated as random oracles under the control of A.

Algorithm A randomly chooses $1 \leq I \leq q_1$ and $1 \leq J \leq q_o$ and starts simulating the real world where the adversary B launches the attack. Algorithm A answers the following queries, which are asked by adversary B in an arbitrary order.

- $H_1(ID_i)$: Algorithm A maintains an initially empty list H_1^{list} with entries of the form (ID_i, Q_i, ℓ_i). The algorithm A responds to the query in the following way.
  - If ID_i already appears on H_1^{list} in a tuple (ID_i, Q_i, ℓ_i), then A responds with $H_1(ID_i) = Q_i$.
  - Otherwise, if ID_i is the I-th unique identifier query, then A inserts $(ID_i, bP, \perp)$ into the list and returns bP.
  - Otherwise, A randomly chooses $\ell_i \in \mathbb{Z}_q^*$, inserts $(ID_i, \ell_i P, \ell_i)$ into the list and returns $\ell_i P$.
- $H_2(ID_u^a, ID_u^b, X_u, Y_u, Z_u, K_u)$: Algorithm A maintains an initially empty list H_2^{list} with entries of the form $(ID_u^a, ID_u^b, X_u, Y_u, Z_u, K_u, h_u)$. The algorithm A responds to the query in the following way.
  - If a tuple indexed by $(ID_u^a, ID_u^b, X_u, Y_u, Z_u, K_u)$ is on the list, then A responds with h_u.
  - Otherwise, A goes through the list $\mathcal{L}$ (maintained in the **Reveal** query) with tuples of the form $(ID_i, ID_j,\ X_i,\ Y_j, \mathit{\Pi}_{i,j}^t)$ to find a tuple with values $(ID_u^a, ID_u^b, X_u, Y_u, \mathit{\Pi}_{i,j}^t)$ and proceeds as following:
    - ∗ Test if $\hat{e}(X_u, Y_u) = \hat{e}(P, Z_u)$. If the equation holds then,
      - · Find the values $f_{i,j}^t$ and $SK_{i,j}^t$ corresponding to oracle $\mathit{\Pi}_{i,j}^t$ from the list $\mathit{\Omega}$.
      - · Find the value ℓ_j from H_1^{list} for party with ID_j.

  · Compute the shared secret via the equation where $M = X_u$ if X_u is the incoming message to the oracle $\Pi^t_{i,j}$, or $M = Y_u$ otherwise,

$$\begin{aligned} K^t_{i,j} &= \hat{e}(M + Q_j, x_iR + d_i), \\ &= \hat{e}(M + \ell_jP, (f^t_{i,j}a)aP + abP), \text{ since } x_i = f^t_{i,j}a, d_i = abP \\ &= \hat{e}(M, f^t_{i,j}aaP) \cdot \hat{e}(M, abP) \cdot \hat{e}(\ell_jP, f^t_{i,j}aaP) \cdot \hat{e}(\ell_jP, abP), \\ &= \hat{e}(Z_u, aP) \cdot \hat{e}(\frac{1}{f^t_{i,j}}Z_u, bP) \cdot \hat{e}(f^t_{i,j}\ell_jaP, aP) \cdot \hat{e}(\ell_jbP, aP), \\ &\qquad\qquad \text{since } f^t_{i,j}aM = Z_u \\ &= \hat{e}(Z_u + f^t_{i,j}\ell_jaP + \ell_jbP, aP) \cdot \hat{e}(\frac{1}{f^t_{i,j}}Z_u, bP) \end{aligned}$$

  Note that $\Pi^t_{i,j}$ is put on the list $\mathcal{L}$ only when $\Pi^t_{i,j}$ has been revealed and $d_i = abP$, but $H_2(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K^t_{i,j})$ had not been queried before the reveal query. So, $SK^t_{i,j}$ has been randomly sampled.
  · Set $h_u = SK^t_{i,j}$.
  · Remove $(ID^a_u, ID^b_u, X_u, Y_u, \Pi^t_{i,j})$ from the list $\mathcal{L}$. Put $(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K^t_{i,j}, h_u)$ in the list H^{list}_2.
  · Check if $K^t_{i,j} = K_u$. If it is not true, A randomly chooses new $h_u \in \{0,1\}^n$, inserts $(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K_u, h_u)$ into the list H^{list}_2.
  · Return h_u.
 * Otherwise (no tuple on $\mathcal{L}$ meets the test), algorithm A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K_u, h_u)$ into the list and returns h_u
- Otherwise, A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID^a_u, ID^b_u, X_u, Y_u, Z_u, K_u, h_u)$ into the list and returns h_u.

– **Corrupt(ID_i)**: A looks through list H^{list}_1. If ID_i is not on the list, A queries $H_1(ID_i)$. A checks the value of ℓ_i: if $\ell_i \neq \perp$, then A responds with ℓ_iaP; otherwise, A aborts the game (**Event 1**).

– **Send($\Pi^t_{i,j}$, M)**: A maintains a list Ω for each oracle of the form $(\Pi^t_{i,j}, tran^t_{i,j}, r^t_{i,j}, K^t_{i,j}, SK^t_{i,j}, f^t_{i,j})$ where $tran^t_{i,j}$ is the transcript of the oracle so far; $r^t_{i,j}$ is the random integer used by the oracle to generate message, $f^t_{i,j}$ is used for special purpose explained below, and $K^t_{i,j}$ and $SK^t_{i,j}$ are set $\perp$ initially. Note that this list is updated in the reveal query and H_2 query as well. A proceeds in the following way:
- If M is the second message on the transcript, do nothing but simply accept the session. Otherwise,
- Query $H_1(ID_i)$ and $H_1(ID_j)$.
- If $t = J$,
  * If $\ell_j \neq \perp$, then abort the game (**Event 2**).
  * Otherwise, respond with cP and set $r^t_{i,j} = \perp$ (if $M = \lambda$, then party ID_i is an initiator, otherwise a responder as M is the first message of the session).
- Otherwise,
  * If $\ell_i = \perp$, randomly choose $f^t_{i,j} \in \mathbb{Z}^*_q$ and respond with $f^t_{i,j}aP$ and set $r^t_{i,j} = \perp$.
  * Otherwise, randomly choose $r^t_{i,j} \in \mathbb{Z}^*_q$ and respond with $r^t_{i,j}P$.

– **Reveal($\Pi^t_{i,j}$)**: A maintains a list $\mathcal{L}$ with tuples of the form $(ID_i, ID_j, X_i, Y_j, \Pi^t_{i,j})$, A proceeds in the following way to respond:
- Get the tuple corresponding to oracle $\Pi^t_{i,j}$ from Ω.
- If oracle $\Pi^t_{i,j}$ has not accepted, then respond with $\perp$.
- If $t = J$ or if the J-th oracle has been generated as $\Pi^J_{a,b}$ and $ID_a = ID_j, ID_b = ID_i$ and $\Pi^J_{a,b}$ and $\Pi^t_{i,j}$ have the same session ID, then abort the game (**Event 3**).

- If $SK_{i,j}^t \neq \perp$, return $SK_{i,j}^t$.
  - ∗ If $r_{i,j}^t \neq \perp$ (so $\ell_i \neq \perp$ and $d_i = \ell_i aP$),
    - · Compute $K_{i,j}^t = \hat{e}(M + Q_j, (r_{i,j}^t + \ell_i)aP)$ where Q_j is found from H_1^{list} for identifier ID_j and M is the received message on $tran_{i,j}^t$. Set $SK_{i,j}^t = H_2(ID_i, ID_j, r_{i,j}^t P, M, r_{i,j}^t M, K_{i,j}^t)$ if $\Pi_{i,j}^t$ is an initiator oracle, or $SK_{i,j}^t = H_2(ID_j, ID_i, M, r_{i,j}^t P, r_{i,j}^t M, K_{i,j}^t)$ otherwise, and return $SK_{i,j}^t$ as the response.
  - ∗ Otherwise, i.e., it should have $r_{i,j}^t = f_{i,j}^t a$ and $d_i = abP$. Algorithm A does not know both values and should compute $K_{i,j}^t = \hat{e}(M + \ell_j P, f_{i,j}^t aR + d_i)$ and $f_{i,j}^t aM$ (note that the model requires that $i \neq j$). Algorithm A proceeds as follows:
    - · Go through the list H_2^{list} to find a tuple $(ID_i, ID_j, f_{i,j}^t aP, M, Z_u, K_u, h_u)$ if ID_i is the initiator or a tuple $(ID_j, ID_i, M, f_{i,j}^t aP, Z_u, K_u, h_u)$ otherwise, meeting the equation $\hat{e}(M, f_{i,j}^t aP) = \hat{e}(P, Z_u)$.
    - · If such Z_u is found, then compute

$$\begin{aligned} K_{i,j}^t &= \hat{e}(M + \ell_j P, f_{i,j}^t aR + d_i) \\ &= \hat{e}(\frac{1}{f_{i,j}^t a} Z_u + \ell_j P, f_{i,j}^t aaP + abP) \text{ since } M = \frac{1}{f_{i,j}^t a} Z_u, \\ &= \hat{e}(Z_u, aP) \cdot \hat{e}(\frac{1}{f_{i,j}^t} Z_u, bP) \cdot \hat{e}(f_{i,j}^t \ell_j aP, aP) \cdot \hat{e}(\ell_j bP, aP), \\ &= \hat{e}(Z_u + f_{i,j}^t \ell_j aP + \ell_j bP, aP) \cdot \hat{e}(\frac{1}{f_{i,j}^t} Z_u, bP) \end{aligned}$$

      and return $SK_{i,j}^t = H_2(ID_i, ID_j, f_{i,j}^t aP, M, Z_u, K_{i,j}^t)$ if party i is the initiator or $SK_{i,j}^t = H_2(ID_j, ID_i, M, f_{i,j}^t aP, Z_u, K_{i,j}^t)$ if i is the responder.
    - · Otherwise, randomly sample $SK_{i,j}^t \in \{0,1\}^n$, put $(ID_i, ID_j, f_{i,j}^t aP, M, \Pi_{i,j}^t)$ if ID_i is the initiator or $(ID_j, ID_i, M, f_{i,j}^t aP, \Pi_{i,j}^t)$ into list $\mathcal{L}$. A responds with $SK_{i,j}^t$.

– **Test($\Pi_{i,j}^t$)**: If $t \neq J$ or ($t = J$ but) there is an oracle $\Pi_{j,i}^w$ with the same session ID with $\Pi_{i,j}^t$ that has been revealed, then A aborts the game (**Event 4**). Otherwise ($\ell_j = \perp, Q_j = bP$ and $r_{i,j}^t = \perp$), A randomly chooses $\zeta \in \{0,1\}^n$ and responds to B with ζ.

Once B finishes queries and returns its guess, A proceeds with the following steps:

– For every pair (X_u, Y_u, Z_u) on H_2^{list} with $X_u = cP, Y_u = M$ if the tested oracle $\Pi_{i,j}^t$ is an initiator oracle, otherwise with $X_u = M, Y_u = cP$ where M is the received message on $tran_{i,j}^t$, check if $\hat{e}(X_u, Y_u) = \hat{e}(P, Z_u)$ holds. If no such Z_u meets the equation, abort the game (**Event 5**).

– Otherwise, compute

$$D = \hat{e}(\ell_i(bP + M) + Z_u, aP).$$

Note that because $i \neq j$ according to Definition 6, it has $d_i = \ell_i aP$ where $\ell_i \neq \perp$ found from H_1^{list} corresponding to identifier ID_i and

$$\begin{aligned} K_{i,j}^t &= \hat{e}(M + Q_j, x_i R + d_i), \\ &= \hat{e}(M + bP, caP + \ell_i aP) \\ &= \hat{e}(P,P)^{abc} \cdot \hat{e}(bP, \ell_i aP) \cdot \hat{e}(M, caP + \ell_i aP), \\ &= \hat{e}(P,P)^{abc} \cdot \hat{e}(\ell_i bP, aP) \cdot \hat{e}(Z_u, aP) \cdot \hat{e}(\ell_i M, aP) \text{ since } M = \frac{1}{c} Z_u, \\ &= \hat{e}(P,P)^{abc} \cdot D. \end{aligned}$$

– Algorithm A randomly chooses K_ℓ from H_2^{list} and returns K_ℓ / D as the response to the BDH challenge.

Following the similar argument as in Theorem 1, we have following two claims.

Claim 5 *If A did not abort the game, B could not find inconsistence between the simulation and the real world.*

Claim 6 *Let* **Event 6** *be that $K = \hat{e}(M + bP, (c + \ell_i)aP)$ was not queried on H_2. Then* $\Pr[\overline{\texttt{Event 5}} \wedge \overline{\texttt{Event 6}}] \geq \epsilon(k)$.

Let **Event 7** be that, in the attack, adversary B indeed chose oracle $\Pi^J_{i,j}$ as the challenger oracle where ID_j was queried on H_1 as the I-th distinct identifier query. Then following the rules of the game defined in Section 3, it's clear that **Event 1, 2, 3, 4** would not happen. So,

$$\Pr[\overline{(\texttt{Event 1} \vee \texttt{Event 2} \vee \texttt{Event 3} \vee \texttt{Event 4})}] = \Pr[\texttt{Event 7}] \geq \frac{1}{q_1 \cdot q_o}.$$

Let **Event 8** be that A did not abort in the game. Let **Event 9** be that A found the correct K_ℓ. Overall, we have

$$\begin{aligned}\Pr[A \text{ wins}] &= \Pr[\texttt{Event 8} \wedge \overline{\texttt{Event 6}} \wedge \texttt{Event 9}] \\ &= \Pr[\texttt{Event 7} \wedge \overline{\texttt{Event 5}} \wedge \overline{\texttt{Event 6}} \wedge \texttt{Event 9}] \\ &\geq \tfrac{1}{q_1 \cdot q_o \cdot q_2} \Pr[\overline{\texttt{Event 5}} \wedge \overline{\texttt{Event 6}}] \\ &\geq \tfrac{1}{q_1 \cdot q_o \cdot q_2} \epsilon(k).\end{aligned}$$

This concludes the proof. □

Theorem 4 *The SYL scheme has master key forward secrecy, provided the DH assumption on $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$ holds and H_2 is modelled as random oracle. Specifically, suppose an adversary B wins the game with advantage $\epsilon(k)$. Then there exists an algorithm A to solve the DH problem in $\mathbb{G}$ with advantage*

$$Adv^{DH}_A \geq \frac{1}{2}\epsilon(k).$$

Proof: The proof is very similar to Theorem 2. Here we only specify the difference from the proof in Theorem 2 which is mainly how the agreed secret K is computed.

- **Send($\Pi^t_{i,j}$, M)**:
  - If M is not the second message on the transcript,
    - If $M \neq \lambda$, compute $K^t_{i,j} = \hat{e}(M + H_1(ID_j), V + H_1(ID_i))^s$ and accept the session.
  - Otherwise, compute $K^t_{i,j} = \hat{e}(M + H_1(ID_j), f^t_{i,j} aP + H_1(ID_i))^s$ if $c^t_{i,j} = 0$, else compute $K^t_{i,j} = \hat{e}(M + H_1(ID_j), f^t_{i,j} bP + H_1(ID_i))^s$, and accept the session.

□

Following the observation of the security model in Section 3, we can confirm that the SCK scheme and the SYL scheme achieve very strong security properties including implicit mutual authentication, known session key security, key compromise impersonation resilience, unknown key share resilience and master key forward secrecy.

As analysed in Section 6, the simulator should deal with one of the following problems: either to compute the session key to be revealed or to determine if the established secret has been queried to the random oracle. Now let us take a closer look at how the built-in decisional function magically help the simulator solve the problem without resorting to any other oracles (but only the random oracles). In the above two protocols, to introduce an efficient built-in decisional function into the protocols, the DH key xyP_2 and the pairing-computed key K are used together in the session key computation through a random oracle H_2. The decisional function then is construed as $\hat{e}(\psi(X), Y) \stackrel{?}{=} \hat{e}(P_1, Z)$ where X, Y are the exchanged key tokens and Z is the DH key value input of H_2. Now the simulator surely can make use of the decisional function to find out that the session secret including xyP_2 and K has not been queried, if xyP_2 has not been queried on H_2. When the DH value has indeed been queried on H_2, the simulator has to compute K, otherwise, it will have to resort to a BDH decisional oracle. By noticing that the difficulty of computing K in some sessions is to compute $\hat{e}(\psi(abP_2), Y)$ for some $Y = yP_2$ generated by the adversary, the simulator makes use of its freedom of choosing some x for $X = xP_2$ to set $X = raP_2$ (or $X = rbP_2$) in those sessions. Now with the value $Z = raY$ (or $Z = rbY$) found by the decisional function and value r, K can be computed as $\hat{e}(\psi(abP_2), Y) = \hat{e}(\psi(bP_2), \frac{1}{r}Z) = \hat{e}(bP_1, \frac{1}{r}Z)$ (or $\hat{e}(\psi(aP_2), \frac{1}{r}Z)$). So, with the decisional function the simulator now can either find out the session secret has not been queried with the random oracle or compute the agreed secret value.

8 Enhancing the CK Scheme and MB scheme

In the last section, we have shown that by including the DH value in the computation of session key so to construct a built-in decisional function, both the SCK scheme and the SYL scheme can be proved in the model defined in Section 3 based on the BDH assumption. Here we show how to construct a built-in decisional function in protocols which use different bases to generate key tokens to enable a security reduction based on the weakest assumption. We use the CK protocol and MB protocol as the case study.

We enhance the CK protocol as follows.

Enhanced Chen-Kudla Protocol:

$$A \to B : (E_A, S_A) = (xQ_A, xH_2(xQ_A))$$
$$B \to A : (E_B, S_B) = (yQ_B, yH_2(yQ_B))$$

where H_2 is a cryptographic hash function $H_2 : \mathbb{G}_2 \to \mathbb{G}_1$.

Upon receiving the key token, A verifies if the equation $\hat{e}(H_2(E_B), E_B) = \hat{e}(S_B, Q_B)$ holds. If the equation holds, A accepts the session, otherwise rejects it. The user B proceeds in the similar way by checking $\hat{e}(H_2(E_A), E_A) = \hat{e}(S_A, Q_A)$. If the session is accepted, A and B compute a session secret shared between them as follows: A computes $K = \hat{e}(\psi(d_A), xQ_B + E_B)$ and B computes $K = \hat{e}(y\psi(Q_A) + \psi(E_A), d_B)$. The session key is computed by $SK = H_3(A, B, E_A, E_B, S_A, S_B, K)$.

Like the SYL scheme, this protocol works only with Type 1 and Type 4 pairings. Again, we only present the formal proof for the protocol used on Type 1 pairing. The security of the enhanced CK protocol is summarised in Theorem 5.

Theorem 5 *The enhanced Chen-Kudla protocol is a secure AK, provided the BDH assumption is sound and the hash functions are modelled as random oracles. Specifically, suppose in the attack, an adversary B which makes q_1 and q_3 queries to H_1 and H_3 respectively and creates q_o oracles, wins the game with advantage $\epsilon(k)$. Then there exists an algorithm A to solve the BDH problem with advantage*

$$Adv_A^{BDH}(k) \geq \frac{1}{q_1 \cdot q_3 \cdot q_o} \epsilon(k).$$

Proof: Given a BDH problem instance (P, aP, bP, cP), A simulates the system setup to B as follows. The master public key is set to be $R = aP$, i.e. the master key is a which A does not known. The hash functions H_1, H_2, H_3 are modelled as random oracles controlled by A.

A randomly chooses $1 \leq I \leq q_1$ and $1 \leq J \leq q_o$, then starts simulating the real world where B launches the attack. B can issue following queries in an arbitrary order.

- $H_1(ID_i)$: A maintains an initially empty list H_1^{list} with entries of the form (ID_i, Q_i, ℓ_i). A responds to the query in the following way.
  - If ID_i already appears on H_1^{list} in a tuple (ID_i, Q_i, ℓ_i), then A responds with $H_1(ID_i) = Q_i$.
  - Otherwise, if ID_i is the I-th unique identifier query, then A inserts $(ID_i, bP, \perp)$ into the list and returns bP.
  - Otherwise, A randomly chooses $\ell_i \in \mathbb{Z}_q^*$, inserts $(ID_i, \ell_i P, \ell_i)$ into the list and returns $\ell_i P$.
- $H_2(S_i)$: A maintains an initially empty list H_2^{list} with entries of the form (S_u, m_u, R_u, w_u). Algorithm A responds to the query in the following way.
  - If a tuple (S_i, m_i, R_i, w_i) has already appeared on H_2^{list}, then A responds with R_i.
  - Otherwise, randomly sample $m_i \in \mathbb{Z}_q^*$. If $m_i aP = P$, then respond to the BDH challenge with $\hat{e}(bP, cP)^{1/m_1}$. Otherwise insert $(S_i, m_i, m_i aP, \perp)$ into H_2^{list} and return $m_i aP$.
- $H_3(ID_u^a, ID_u^b, X_u, Y_u, L_u, N_u, K_u)$: Algorithm A maintains an initially empty list H_3^{list} with entries of the form (ID_u^a, ID_u^b, X_u, Y_u, L_u, N_u, K_u, h_u). A responds to the query in the following way.
  - If a tuple indexed by $(ID_u^a, ID_u^b, X_u, Y_u, L_u, N_u, K_u)$ is on the list, then A responds with h_u.
  - Otherwise, A randomly chooses $h_u \in \{0,1\}^n$, inserts (ID_u^a, ID_u^b, X_u, Y_u, L_u, N_u, K_u, h_u) into the list and returns h_u.
- **Corrupt**(ID_i): A looks through list H_1^{list}. If ID_i is not on the list, A queries $H_1(ID_i)$. Algorithm A then checks the value of ℓ_i: if $\ell_i \neq \perp$, then A responds with $\ell_i aP$; otherwise, A aborts the game (**Event 1**).
- **Send**$(\Pi_{i,j}^t$**, (M, N))**: A maintains a list Ω for each oracle of the form $(\Pi_{i,j}^t, tran_{i,j}^t, r_{i,j}^t, K_{i,j}^t, SK_{i,j}^t)$ where $tran_{i,j}^t$ is the transcript of the oracle so far; $r_{i,j}^t$ is the random integer used by the oracle to generate message, and $K_{i,j}^t$, $SK_{i,j}^t$ are set $\perp$ initially. Note that the values in this list are accessed by A in other queries and also updated in other queries as well. A proceeds in the following way:
  - Query $Q_i = H_1(ID_i)$ and $Q_j = H_1(ID_j)$ and $R_t = H_2(M)$.
  - If $t = J$,
    - ∗ If $\ell_j \neq \perp$ (**Event 2**) where ℓ_j is found from H_1^{list} corresponding to ID_j, then abort the game.
    - ∗ Otherwise,
      - · If either $(M,N) = \lambda$ or (M,N) is the first message of the session, first randomly sample $\xi \in \mathbb{Z}_q^*$ such that $\Phi = \xi \ell_i cP$ is not shown on H_2^{list} as some S_u, and then randomly sample $w_t \in \mathbb{Z}_q^*$ and insert a tuple $(\xi \ell_i cP, \perp, w_t P, w_t)$ into H_2^{list}. Then perform in one of the following two ways:
        1. If $(M,N) = \lambda$, respond with ($\xi \ell_i cP$, $w_t \xi cP$),i.e., $r_{i,j}^J = \xi c$ which the simulator does not know. It is easy to verify that the response can pass the message check $\hat{e}(w_t P, \xi \ell_i cP) = \hat{e}(w_t \xi cP, \ell_i P)$.
        2. If (M,N) is the first message of the session, check if $\hat{e}(R_t, M) = \hat{e}(N, Q_j)$. If the equation holds, respond with ($\xi \ell_i cP$, $\xi w_t cP$), and accept the session, otherwise reject the session and abort the game (**Event 3**).
      - · If (M,N) is the last message, check if $\hat{e}(R_t, M) = \hat{e}(N, Q_j)$. If the equation holds, do nothing but accept the session, otherwise reject the session and abort the game (**Event 4**).
  - Otherwise,
    - ∗ If $(M,N) \neq \lambda$, check if $\hat{e}(R_t, M) = \hat{e}(N, Q_j)$. If the equation does not hold, reject the session.
    - ∗ If either $(M,N) = \lambda$ or (M,N) is the first message of the session, first sample $r_{i,j}^t$ as follows: If $\ell_i \neq \perp$, randomly sample $r_{i,j}^t \in \mathbb{Z}_q^*$. Otherwise (i.e., $\ell_i = \perp$ and $Q_i = bP$), randomly sample $r_{i,j}^t \in \mathbb{Z}_q^*$, such that $r_{i,j}^t bP$ has not been shown on H_3^{list} as some X_u if $\Pi_{i,j}^t$ is the initiator or Y_u otherwise. Then, follow the items below.
    - ∗ If $(M,N) = \lambda$, respond with ($r_{i,j}^t Q_i$, $r_{i,j}^t H_2(r_{i,j}^t Q_i)$).
    - ∗ If (M,N) is the first message of the session, respond with ($r_{i,j}^t Q_i$, $r_{i,j}^t H_2(r_{i,j}^t Q_i)$). If (M,N) is the last message, then accept the session. For both cases, compute $SK_{i,j}^t$ as below.

· If $\ell_i \neq \perp$, compute $K_{i,j}^t = \hat{e}(\ell_i aP, M + r_{i,j}^t Q_j)$
· Otherwise, the agreed secret should be

$$\begin{aligned} K_{i,j}^t &= \hat{e}(abP, M + r_{i,j}^t Q_j) \\ &= \hat{e}(bP, aM) \cdot \hat{e}(bP, r_{i,j}^t \ell_j aP). \end{aligned}$$

If $M \neq \Phi$, i.e., $R_t = H_2(M) = m_t aP$ for some $m_t \neq \perp$ found on H_2^{list} for M (note that in the whole game only for Φ, $H_2(\Phi) = w_t P$ and so $m_t = \perp$ found on H_2^{list}), then because (M, N) meets the equation $\hat{e}(R_t, M) = \hat{e}(N, \ell_j P)$, i.e., $\hat{e}(m_t aP, M) = \hat{e}(N, \ell_j P)$, A can compute

$$K_{i,j}^t = \hat{e}(bP, \frac{\ell_j}{m_t} N) \cdot \hat{e}(bP, r_{i,j}^t \ell_j aP).$$

Otherwise, i.e., $M = \Phi$, the simulator does nothing (this case will be further analysed in Claim 7).
· If $K_{i,j}^t$ is computed, set $SK_{i,j}^t = H_3(ID_i,\ ID_j,\ r_{i,j}^t Q_i,\ M,\ r_{i,j}^t R_t,\ N,\ K_{i,j}^t)$ if party i is the initiator, or $SK_{i,j}^t = H_3(ID_j,\ ID_i,\ M,\ r_{i,j}^t Q_i,\ N,\ r_{i,j}^t R_t,\ K_{i,j}^t)$ otherwise.
· Otherwise, i.e., $K_{i,j}^t$ has not been computed, randomly sample $SK_{i,j}^t \in \{0,1\}^n$.

– **Reveal($\Pi_{i,j}^t$)**: Algorithm A proceeds in the following way to respond:
  - If oracle $\Pi_{i,j}^t$ has not accepted, then respond with $\perp$.
  - If $t = J$ or if the J-th oracle has been generated as $\Pi_{a,b}^J$ and $ID_a = ID_j, ID_b = ID_i$ and $\Pi_{a,b}^J$ and $\Pi_{i,j}^t$ have the same session ID, then abort the game (**Event 5**).
  - Return $SK_{i,j}^t$.

– **Test($\Pi_{i,j}^t$)**: If $t \neq J$ or $t = J$ but there is an oracle $\Pi_{j,i}^w$ with the same session ID with $\Pi_{i,j}^t$ that has been revealed, then A aborts the game (**Event 6**). Otherwise ($\ell_j = \perp, Q_j = bP$ and $r_{i,j}^t = \perp$), and so A randomly chooses $\zeta \in \{0,1\}^n$ and responds to B with ζ.

Once B finishes queries and returns its guess, A proceeds with the following steps:

– Compute $D = \hat{e}(\ell_i aP, M)$ where (M, N) is the received message of $\Pi_{i,j}^J$. Note that because $i \neq j$ according to Definition 6, it has $d_i = \ell_i aP$ where $\ell_i \neq \perp$ found from H_1^{list} corresponding to identifier ID_i of $\Pi_{i,j}^J$ and we should have $K_{i,j}^J = \hat{e}(\ell_i aP, \xi cbP + M) = \hat{e}(P,P)^{\xi \ell_i abc} \cdot D$.
– Then A randomly chooses K_ℓ from H_3^{list} and returns $(K_\ell / D)^{\frac{1}{\xi \ell_i}}$ as the response to the BDH challenge.

Let **Event 7** be that, in the attack, adversary B indeed chose oracle $\Pi_{i,j}^J$ as the challenger oracle where ID_j was queried on H_1 as the I-th distinct identifier query. Then following the rules of the game defined in Section 3, we have

$$\begin{aligned} \Pr[\texttt{Event 7}] &= \Pr[\overline{(\texttt{Event 1} \vee \texttt{Event 2} \vee \texttt{Event 3} \vee \texttt{Event 4} \vee \texttt{Event 5}) \vee \texttt{Event 6})}] \\ &\geq \frac{1}{q_1 \cdot q_o}. \end{aligned}$$

Claim 7 *Let* **Event 8** *be that B noticed inconsistence between the simulation and the real world when the game did not abort. Then Event 8 implies that A can be tweaked to solve the BDH problem with probability $\frac{1}{q_1 \cdot q_o \cdot q_3}$.*

Proof: For most oracles, the simulator honestly follows the protocol specification to compute the session keys correctly, except for oracles $\Pi_{j,u}^t$ with $H_1(ID_j) = bP$ and incoming message $M = \Phi$ ($\Phi = \xi \ell_i cP$ is generated by the tested oracle). As each of such $\Pi_{j,u}^t$ has a unique transcript and H_3 is a random oracle which takes the transcripts as part of the input to generate session keys, if B noticed the difference between the reveal query on $\Pi^t_{j,u}$ and a query on H_3, it must have queried H_3 with $K^t_{j,u}$ such that

$$\begin{aligned} K^t_{j,u} &= \hat{e}(abP, \xi\ell_i cP + r^t_{j,u}Q_u) \\ &= \hat{e}(abP, cP)^{\xi\ell_i} \cdot \hat{e}(bP, r^t_{j,u}\ell_u aP). \end{aligned}$$

So if B is a distinguisher on the session key $K^t_{j,i}$ in the simulation from the real world, then A randomly chooses K_ℓ from H_3^{list} and returns $(K_\ell/\hat{e}(bP, r^t_{j,u}\ell_u aP))^{\frac{1}{\xi\ell_i}}$ as the response to the BDH challenge. And the answer to the BDH problem will be correct with probability $\frac{1}{q_1 \cdot q_o \cdot q_3}$. All the other queries follows the protocol in the random oracle model. The claim follows.

Following the similar argument as in Theorem 1, we have the following claim.

Claim 8 *Let* **Event 9** *be that* $K = \hat{e}(\ell_i aP, M + \xi cbP)$ *was not queried on* H_3 *conditioned on* $\overline{\texttt{Event 8}}$*. Then* $\Pr[\overline{\texttt{Event 9}}] \geq \epsilon(k)$*.*

Let **Event 10** be that A found the correct K_ℓ. Overall, we have

$$\begin{aligned} \Pr[A \text{ wins}] &= \Pr[A \text{ wins} \mid \overline{\texttt{Event 8}}] \Pr[\overline{\texttt{Event 8}}] + \Pr[A \text{ wins} \mid \texttt{Event 8}] \Pr[\texttt{Event 8}] \\ &= \Pr[\texttt{Event 7} \wedge \overline{\texttt{Event 9}} \wedge \texttt{Event 10}] \Pr[\overline{\texttt{Event 8}}] + \Pr[A \text{ wins} \mid \texttt{Event 8}] \Pr[\texttt{Event 8}] \\ &\geq \tfrac{1}{q_1 \cdot q_o} \tfrac{1}{q_3} \epsilon(k) \Pr[\overline{\texttt{Event 8}}] + \tfrac{1}{q_1 \cdot q_o \cdot q_3} \Pr[\texttt{Event 8}] \\ &\geq \tfrac{1}{q_1 \cdot q_o \cdot q_3} \epsilon(k). \end{aligned}$$

This concludes the proof. □

Using a similar approach, the MB-2 protocol can be enhanced as follow:

Enhanced McCullagh-Barreto Protocol:

$$\begin{aligned} A \rightarrow B &: (E_A, S_A) = (xT_B, xH_2(xT_B)) \\ B \rightarrow A &: (E_B, S_B) = (yT_A, yH_2(yT_A)) \end{aligned}$$

where H_2 is cryptographic function $H_2 : \mathbb{G}_2 \rightarrow \mathbb{G}_1$. However, the enhanced protocol uses the Extract 2' algorithm, i.e, $R, T_A \in \mathbb{G}_2$ and $d_A \in \mathbb{G}_1$.

Like the enhanced CK protocol, A and B should proceed the check on the key token by verifying $\hat{e}(H_2(E_B), E_B) = \hat{e}(S_B, T_A)$ and $\hat{e}(H_2(E_A), E_A) = \hat{e}(S_A, T_B)$ respectively before accepting the session. If the session is accepted, A and B compute a session secret shared between them as follows: A computes $K = \hat{e}(d_A, E_B) \cdot \hat{e}(P_1, P_2)^x$ and B computes $K = \hat{e}(d_B, E_A) \cdot \hat{e}(P_1, P_2)^y$. The session key is computed by $SK = H_3(A, B, E_A, E_B, S_A, S_B, K)$.

To ease the reduction, we use the ℓ-BCAA1 assumption which is a related assumption with ℓ-BDHI. It has been shown that an ℓ-BDHI problem can be converted to the corresponding ℓ-BCAA1 problem [6].

Theorem 6 *The enhanced McCullagh-Barreto protocol is a secure AK, provided the* ℓ*-BCAA1*$^{\psi}_{2,1}$ *assumption is sound and the hash functions are modelled as random oracles. Specifically, suppose in the attack, an adversary* B *which makes* q_1 *and* q_3 *queries to* H_1 *and* H_3 *respectively and creates* q_o *oracles, wins the game with advantage* $\epsilon(k)$*. Then there exists an algorithm* A *to solve the* $(q_1 - 1)$*-BCAA1*$^{\psi}_{2,1}$ *problem with advantage*

$$Adv_A^{(q_1-1)-BCAA1^{\psi}_{2,1}}(k) \geq \frac{1}{q_1 \cdot q_3 \cdot q_o} \epsilon(k).$$

Proof: Given an instance of the (q_1-1)-BCAA$1^{\psi}_{2,1}$ problem

$$(sP_2, h_0, (h_1, \frac{1}{h_1+s}P_1), \ldots, (h_{q_1-1}, \frac{1}{h_{q_1-1}+s}P_1))$$

where $h_i \in_R \mathbb{Z}_q^*$ for $0 \le i \le q_1 - 1$, algorithm A simulates the Setup algorithm to generate the master public key as $R = sP_2$, i.e., using s as the master key which it does not know. The hash functions H_1, H_2 and H_3 are random oracles controlled by A.

Again A randomly chooses $1 \le I \le q_1$ and $1 \le J \le q_o$, then starts simulating the real world. Adversary B can issue following queries in an arbitrary order.

- $H_1(ID_i)$**:** Algorithm A maintains a list of tuples (ID_i, h_i, d_i) as explained below. We refer to this list as H_1^{list}. The list is initially empty. When B queries the oracle H_1 at a point on ID_i, A responds as follows:
  - If ID_i already appears on the H_1^{list} in a tuple (ID_i, h_i, d_i), then A responds with $H_1(ID_i) = h_i$.
  - Otherwise, if the query is on the I-th distinct ID, then A inserts $(ID_I, h_0, \perp)$ into the tuple list and responds with $H_1(ID_I) = h_0$.
  - Otherwise, A selects a random integer $h_i(i > 0)$ from the (q_1-1)-BCAA1 instance which has not been chosen by A and inserts $(ID_i, h_i, \frac{1}{h_i+s}P_1)$ into the tuple list. Algorithm A responds with $H_1(ID_i) = h_i$.
- $H_2(S_i)$: Algorithm A maintains an initially empty list H_2^{list} with entries of the form (S_u, m_u, R_u, w_u). Algorithm A responds to the query in the following way.
  - If a tuple (S_i, m_i, R_i, w_i) has already appeared on H_2^{list}, then A responds with R_i.
  - Otherwise, randomly sample $m_i \in \mathbb{Z}_q^*$. If $m_iP_1 = \psi(sP_2)$, then respond to the (q_1-1)-BCAA1 challenge with $\frac{1}{m_i+h_0}P_1$. Otherwise insert $(S_i, m_i, m_iP_1, \perp)$ into H_2^{list} and return m_iP_1.
- $H_3(ID_u^a, ID_u^b, X_u, Y_u, L_u, N_u, K_u)$: Algorithm A maintains an initially empty list H_3^{list} with entries of the form $(ID_u^a,\ ID_u^b,\ X_u,\ Y_u,\ L_u,\ N_u,\ K_u,\ h_u)$. A responds to the query in the following way.
  - If a tuple indexed by $(ID_u^a, ID_u^b, X_u, Y_u, L_u, N_u, K_u)$ is on the list, then A responds with h_u.
  - Otherwise, A randomly chooses $h_u \in \{0,1\}^n$, inserts $(ID_u^a, ID_u^b,\ X_u,\ Y_u,\ L_u,\ N_u,\ K_u,\ h_u)$ into the list and returns h_u.
- **Corrupt(**ID_i**)**: Algorithm A looks through list H_1^{list}. If ID_i is not on the list, A queries $H_1(ID_i)$. Algorithm A checks the value of d_i: if $d_i \neq \perp$, then A responds with d_i; otherwise, A aborts the game (**Event 1**).
- **Send(**$\Pi_{i,j}^t$**, (M, N))**: Algorithm A maintains a list Ω for each oracle of the form $(\Pi_{i,j}^t, tran_{i,j}^t, r_{i,j}^t, K_{i,j}^t, SK_{i,j}^t)$ where $tran_{i,j}^t$ is the transcript of the oracle so far; $r_{i,j}^t$ is the random integer used by the oracle to generate message, and $K_{i,j}^t$ and $SK_{i,j}^t$ are set $\perp$ initially. Note that the values in this list are accessed by A in other queries and also updated in other queries as well. Algorithm A proceeds in the following way:
  - Query $T_i = H_1(ID_i)P_2 + sP_2$ and $T_j = H_1(ID_j)P_2 + sP_2$ and $R_t = H_2(M)$.
  - If $t = J$,
    - ∗ If $d_j \neq \perp$ (**Event 2**) where d_j is found from H_1^{list} corresponding to ID_j, then abort the game.
    - ∗ Otherwise,
      - · If (M, N) is not the last message, randomly sample $\alpha \in_R \mathbb{Z}_q^*$ such that $\Phi = \alpha P_2$ is not shown on H_2^{list} as some S_u, and then randomly sample $w_t \in \mathbb{Z}_q^*$ and insert the tuple $(\alpha P_2, \perp, w_t(h_0P_1 + \psi(sP_2)), w_t)$ into H_2^{list}.
      - · If $(M, N) = \lambda$, respond with $(E_i, S_i) = (\alpha P_2, w_t\alpha P_1)$, i.e., $r_{i,j}^t = \frac{\alpha}{h_0+s}$ which the simulator does not know. It is easy to verify that the message pass the check $\hat{e}(H_2(E_i), E_i) = \hat{e}(S_i, T_j)$.

· If (M, N) is the first message of the session, then check if $\hat{e}(R_t, M) = \hat{e}(N, T_i)$. If the equation holds, respond with $(\alpha P_2,\ w_t \alpha P_1)$, and accept the session, otherwise reject the session and abort the game (**Event 3**).

· If (M, N) is the last message of the session, check if $\hat{e}(R_t, M) = \hat{e}(N, T_i)$. If the equation holds, do nothing but accept the session, otherwise reject the session and abort the game (**Event 4**)

- Otherwise,
  - ∗ If (M, N) is not the last message,
    - · If $d_i \neq \perp$, randomly sample $r^t_{i,j} \in \mathbb{Z}^*_q$.
    - · Otherwise (i.e., $d_i = \perp$ and $Q_i = h_0 P_2 + sP_2$), randomly sample $r^t_{i,j} \in \mathbb{Z}^*_q$, such that $r^t_{i,j} Q_i$ has not been shown on H_3^{list} as some X_u if $\Pi^t_{i,j}$ is the initiator or Y_u otherwise.
  - ∗ If $(M, N) = \lambda$, respond with $(r^t_{i,j} T_j,\ r^t_{i,j} H_2(r^t_{i,j} T_j))$.
  - ∗ Otherwise, check if $\hat{e}(R_t, M) = \hat{e}(N, T_i)$. If the equation does not hold, reject the session. Otherwise,
    - · If (M, N) is the first message of the session, respond with $(r^t_{i,j} T_j,\ r^t_{i,j} H_2(r^t_{i,j} T_j))$, and
    - · Compute $SK^t_{i,j}$ as below and accept the session.
    - · If $d_i \neq \perp$, compute

$$K^t_{i,j} = \hat{e}(d_i, M) \cdot \hat{e}(P_1, P_2)^{r^t_{i,j}}.$$

    - · Otherwise, the agreed secret should be

$$K^t_{i,j} = \hat{e}(\frac{1}{h_0 + s} P_1, M) \cdot (P_1, P_2)^{r^t_{i,j}}.$$

If $M \neq \alpha P_2$, i.e., $m_t \neq \perp$ is found from the tuple indexed by M on H_2^{list}, then because (M, N) satisfies the equation $\hat{e}(R_t, M) = \hat{e}(N, T_i)$, i.e., $\hat{e}(m_t P_1, M) = \hat{e}(N, (h_0 + s)P_2)$, we can compute

$$K^t_{i,j} = \hat{e}(\frac{1}{m_t} N, P_2) \cdot \hat{e}(P_1, P_2)^{r^t_{i,j}}.$$

Otherwise ($M = \alpha P_2$), do nothing.

    - · If $K^t_{i,j}$ is computed, then set $SK^t_{i,j} = H_3(ID_i,\ ID_j,\ r^t_{i,j} T_i,\ M,\ r^t_{i,j} R_t,\ N,\ K^t_{i,j})$ if party i is the initiator, or $SK^t_{i,j} = H_3(ID_j,\ ID_i,\ M,\ r^t_{i,j} T_i,\ N,\ r^t_{i,j} R_t,\ K^t_{i,j})$ otherwise. If $K^t_{i,j}$ has not been computed, randomly sample $SK^t_{i,j} \in \{0,1\}^n$.

– **Reveal($\Pi^t_{i,j}$)**: Algorithm A proceeds in the following way to respond:
  - If oracle $\Pi^t_{i,j}$ has not accepted, then respond with $\perp$.
  - If $t = J$ or if the J-th oracle has been generated as $\Pi^J_{a,b}$ and $ID_a = ID_j, ID_b = ID_i$ and $\Pi^J_{a,b}$ and $\Pi^t_{i,j}$ have the same session ID, then abort the game (**Event 5**).
  - Return $SK^t_{i,j}$.

– **Test($\Pi^t_{i,j}$)**: If $t \neq J$ or ($t = J$ but) there is an oracle $\Pi^w_{j,i}$ with the same session ID with $\Pi^t_{i,j}$ that has been revealed, then A aborts the game (**Event 6**). Otherwise ($d_j = \perp, T_j = h_0 P_2 + sP_2$ and $r^t_{i,j} = \perp$), algorithm A randomly chooses $\zeta \in \{0,1\}^n$ and responds to B with ζ.

Once B finishes queries and returns its guess, A proceeds with the following steps:

– Compute $D = \hat{e}(d_i, M)$ where (M, N) is the received message of $\Pi^J_{i,j}$ and d_i is found from H_1^{list} corresponding to ID_i of $\Pi^J_{i,j}$. Note that

$$K^J_{i,j} = \hat{e}(d_i, M) \cdot \hat{e}(P_1, P_2)^{\frac{\alpha}{h_0+s}}.$$

– Algorithm A randomly chooses K_ℓ from H_3^{list} and returns $(K_\ell / D)^{1/\alpha}$ as the response to the $(q_1 - 1)$-BCAA1 challenge.

Following the similar argument as in Theorem 5, the theorem is proved. □

The above two simulators use an approach different from the one used in Section 7 to construct the decisional function. Instead of ascertaining whether a session secret has been queried with the random oracle, the decisional function can help the simulator to compute the established secret directly. In the CK protocol, the difficulty of computing K of some sessions is to compute $\hat{e}(abP, M)$ where $M = yQ_B$ for some y and M is generated by the adversary. To compute this value, the protocol is enhanced by introducing a new component $N = yH_2(M)$ in the key tokens. The built-in decisional function is by $\hat{e}(H_2(M), M) \stackrel{?}{=} \hat{e}(N, Q_B)$. As the simulator controls the random oracle H_2, it can map most of the M to raP (or rbP)) with one exception, i.e., the token $\overline{M}$ generated by the tested oracle which is mapped to wcP. If the decisional function finds the equation holds and $Q_B = jP$ for some j known to the simulator, $\hat{e}(abP, M)$ can then be computed as $\hat{e}(\frac{1}{r}N, jbP)$ (or $\hat{e}(\frac{1}{r}N, jaP)$). If the adversary resends $\overline{M}$ to the party with private key abP, as the simulator cannot compute the agreed secret, the simulator makes use of random oracle H_3 to either sample a session key which is indistinguishable from the valid one or solve the BDH problem if the adversary distinguishes the revealed (randomly sampled) session key from the valid one. The same approach works for the enhanced McCullagh-Barreto protocol.

9 Conclusions

We have presented a new approach of incorporating a built-in decisional function in two-party identity-based key agreement protocols from pairings. This decisional function is designed to transfer a hard decisional problem in the security analysis of this type of protocols to the DDH problem, which is not hard, based on the fact that these protocols are implemented in the group where a pairing exists. We have thus demonstrated how to reduce the security of a number of this type of protocols to computational rather than gap assumptions.

References

1. M. Bellare, D. Pointcheval and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – Eurocrypt 2000*, Springer-Verlag LNCS 1807, 139–155, 2000.
2. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – Crypto '93*, Springer-Verlag LNCS 773, 232–249, 1993.
3. S. Blake-Wilson, D. Johnson and A. Menezes. Key agreement protocols and their security analysis. In *Cryptography and Coding*, Springer-Verlag LNCS 1355, 30–45, 1997.
4. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Advances in Cryptology – Crypto 2001*, Springer-Verlag LNCS 2139, 213–229, 2001.
5. C. Boyd, W. Mao and K. Paterson. Key agreement using statically keyed authenticators. In *Applied Cryptography and Network Security: Second International Conference - ACNS* Springer-Verlag LNCS 3089, 248–262, 2004.
6. L. Chen and Z. Cheng. Security proof of the Sakai-Kasahara's identity-based encryption scheme. In *Cryptography and Coding*, Springer-Verlag LNCS 3706, 442–459, 2005.
7. L. Chen and C. Kudla. Identity based authenticated key agreement from pairings. In *IEEE Computer Security Foundations Workshop*, 219–233, 2003. The modified version of this paper is available at Cryptology ePrint Archive, Report 2002/184.
8. Z. Cheng. The private communications, 2003.
9. Z. Cheng and L. Chen. On security proof of McCullagh-Barreto's key agreement protocol and its variants. Cryptology ePrint Archive, Report 2005/201.
10. Z. Cheng, M. Nistazakis, R. Comley and L. Vasiu. On the indistinguishability-based security model of key agreement protocols-simple cases. Cryptology ePrint Archive, Report 2005/129.

11. Y. Choie, E. Jeong and E. Lee. Efficient identity-based authenticated key agreement protocol from pairings. *Applied Mathematics and Computation*, **162**, 179–188, 2005.
12. K. Choo, C. Boyd and Y. Hitchcock. On session key construction in provably-secure key establishment protocols: revisiting Chen & Kudla (2003) and McCullagh & Barreto (2005) ID-based protocols. Cryptology ePrint Archive, Report 2005/206. Also available at the Proceedings of Mycrypt 2005, Springer-Verlag LNCS 3715, 116 - 131, 2005.
13. S. Galbraith, K. Paterson and N.P. Smart. Pairings for cryptographers. In preparation.
14. R. Granger, D. Page and N.P. Smart. High security pairing-based cryptography revisited. To appear *ANTS-VII*, 2006.
15. F. Hess, N.P. Smart and F. Vercauteren. The Eta pairing revisited. Cryptology ePrint Archive, Report 2006/110.
16. ISO/IEC 11770-3:1999. Information technology - Security techniques - Key management - Part 3: Mechanisms using asymmetric techniques.
17. A. Joux. A one-round protocol for tripartite Diffie–Hellman. In *Algorithmic Number Theory Symposium – ANTS-IV*, Springer-Verlag LNCS 1838, 385–394, 2000.
18. C. Kudla. *Special Signature Schemes and Key Agreement Protocols.* PhD Thesis, Royal Holloway University of London, 2006.
19. C. Kudla and K. Paterson. Modular security proofs for key agreement protocols. In *Advances in Cryptology – Asiacrypt 2005*, Springer-Verlag LNCS 3788, 549–565, 2005.
20. C.H. Lim, P.J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology – Crypto '97*, Springer-Verlag LNCS 1294, 249-263, 1997.
21. L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, **28**, 119–134, 2003.
22. S. Li, Q. Yuan and J. Li. Towards security two-part authenticated key agreement protocols. Cryptology ePrint Archive, Report 2005/300.
23. N. McCullagh and P.S.L.M. Barreto. A new two-party identity-based authenticated key agreement. In *Topics in Cryptology – CT-RSA 2005*, Springer-Verlag LNCS 3376, 262–274, 2005.
24. C. Mitchell, M. Ward and P. Wilson. Key control in key agreement protocols. *Electronics Letters*, **34**, 980–981, 1998.
25. E. Okamoto. Proposal for identity-based key distribution system. *Electronics Letters*, **22**, 1283–1284, 1986.
26. E. Ryu, E. Yoon and K. Yoo, An efficient ID-based authenticated key agreement protocol from pairings. In *Networking 2004*, Springer-Verlag LNCS 3042, 1458–1463, 2004.
27. R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054.
28. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. In *Symposium on Cryptography and Information Security*, Okinawa, Japan, 2000.
29. M. Scott. Authenticated ID-based key exchange and remote log-in with insecure token and PIN number. Cryptology ePrint Archive, Report 2002/164.
30. H. Shacham. *New Paradigms in Signature Schemes.* PhD Thesis, U. Stanford, 2005.
31. A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology – Crypto '84*, Springer-Verlag LNCS 196, 47–53, 1984.
32. K. Shim. Efficient ID-based authenticated key agreement protocol based on the Weil pairing. *Electronics Letters*, **39**, 653–654, 2003.
33. K. Shim. Cryptanalysis of two ID-based authenticated key agreement protocols from pairings. Cryptology ePrint Archive, Report 2005/357.
34. N.P. Smart. An identity based authenticated key agreement protocol based on the Weil pairing. *Electronics Letters*, **38**, 630–632, 2002.
35. H. Sun and B. Hsieh. Security analysis of Shim's authenticated key agreement protocols from pairings. Cryptology ePrint Archive, Report 2003/113.
36. K. Tanaka and E. Okamoto. Key distribution system for mail systems using ID-related information directory. *Computers & Security*, **10**, 25–33, 1991.
37. Y. Wang. Efficient identity-based and authenticated key agreement protocol. Cryptology ePrint Archive, Report 2005/108.
38. G. Xie. An ID-based key agreement scheme from pairing. Cryptology ePrint Archive, Report 2005/093.

39. Q. Yuan and S. Li. A new efficient ID-based authenticated key agreement protocol. Cryptology ePrint Archive, Report 2005/309.

Appendices

A Subgroup Membership Testing

Here we show, with two examples, that subgroup membership testing is crucial to our protocols.

Consider the SCK-1 scheme in the Type 2 setting. According to our security model the following is a legitimate attack on the protocol. The adversary wishes to obtain the session key for a session with message flow

$$\begin{aligned} A \to B &: E_A = xP_2, \\ B \to A &: E_B = yP_2. \end{aligned}$$

The shared session key is computed by the legitimate party B as

$$yE_A, \quad \hat{e}(yQ_A, R') \cdot \hat{e}(d_B, E_A).$$

Let N denote a point of low order l (say) in the group $\mathcal{G}$. The adversary E engages in the following "man-in-the-middle" attack.

$$\begin{array}{ccccc} A & & E & & B \\ & \xrightarrow{E_A = xP_2} & & & \\ & & & \xrightarrow{E_A + N} & \\ & & & \xleftarrow{E_B = yP_2} & \\ & \xleftarrow{E_B = yP_2} & & & \end{array}$$

There is no matching conversation between party A and B hence the adversary is allowed to make a Reveal query on B even if A corresponds to the Test query. However on making the Reveal query on B the adversary learns, with probability $1/l$, the value of the key which A thinks it shares with B. This is because the addition of an element of the small subgroup passes is ignored by the pairing, and will be killed off by the computation of $y(E_A + N)$ if y is divisible by l. In the above one can always take $l = 2$.

One could argue that the above attack is in some sense a symptom of the rather strong security model we have taken. However, the following example shows that in practice one can mount impersonation attacks if the subgroup membership tests are not carried out fully.

Suppose one optimized the SYL protocol with Type 4 pairings in the following way.

$$\begin{aligned} A \to B &: E_A = xP_2, \\ B \to A &: E_B = yP_1, \end{aligned}$$

Upon completion of the message exchange, A computes $xE_B = xyP_1$ and $K = \hat{e}(E_B + \psi(Q_B), xR' + d_A)$ and B computes $y\psi(E_A) = xyP_1$ and $K = \hat{e}(y\psi(R') + \psi(d_B), E_A + Q_A)$. Recall that $R' = sP_2 = s\frac{1}{k}\mathcal{P}_1 + s\mathcal{P}_2$ is the master public key and s is the master private key. This optimised version has smaller bandwidth cost and better performance than the original SYL specification. Suppose to save time that party B does not check whether E_A lies in the subgroup generated by P_2, but only whether it lies in $\mathbb{G}_2$, i.e. it computes some of the subgroup membership test but not all of it. We show in this situation that an adversary C can impersonate party A to B

Without loosing generality, we assume

$$Q_A = a\mathcal{P}_1 + b\mathcal{P}_2 \in \mathbb{G}_2 \text{ and } Q_B = c\mathcal{P}_1 + d\mathcal{P}_2 \in \mathbb{G}_2.$$

Adversary C chooses random integers x and z from $\mathbb{Z}_q^*$, and generates the message E_A in the following attack.

$$\begin{aligned} C_A \to B : E_A &= x\frac{1}{k}\mathcal{P}_1 - b\mathcal{P}_2 + z\mathcal{P}_2, \\ B \to C_A : E_B &= yP_1 = y\mathcal{P}_1, \end{aligned}$$

Now B computes $y\psi(E_A) = yxP_1 = xE_B$ and

$$\begin{aligned} K &= \hat{e}(y\psi(R') + \psi(d_B), E_A + Q_A) \\ &= \hat{e}(ys\mathcal{P}_1 + \psi(sQ_B), x\frac{1}{k}\mathcal{P}_1 - b\mathcal{P}_2 + z\mathcal{P}_2 + a\mathcal{P}_1 + b\mathcal{P}_2) \\ &= \hat{e}(ys\mathcal{P}_1 + s\psi(Q_B), z\mathcal{P}_2) \\ &= \hat{e}(E_B + \psi(Q_B), s\mathcal{P}_2)^z \\ &= \hat{e}(E_B + \psi(Q_B), R')^z \end{aligned}$$

Both xE_B and $K = \hat{e}(E_B + \psi(Q_B), R')^z$ can be computed by C. To prevent this attack, party B should also check that for $E_A = x_1\frac{1}{k}\mathcal{P}_1 + x_2\mathcal{P}_2$, $x_1 = x_2$, i.e., E_A is in the cyclic group generated by P_2. The test method can be found in Section 2.2.

A careful analysis of the proof of the SYL protocol reveals that for the above optimization one is unable to obtain a proof if full subgroup membership testing is not performed. Yet if full subgroup membership testing is performed then the protocol is secure. This example shows that the implication of a reduction should be carefully investigated, and when optimising a protocol one should be careful.