Identity-Based Key Exchange Protocols without Pairings*

Dario Fiore[1]** and Rosario Gennaro[2]

[1] École Normale Supérieure, CNRS - INRIA, Paris
dario.fiore@ens.fr
[2] IBM T.J. Watson Research Center – Hawthorne, New York 10532.
rosario@us.ibm.com

Abstract. This paper presents a new identity based key agreement protocol. In id-based cryptography (introduced by Adi Shamir in [34]) each party uses its own identity as public key and receives his secret key from a master Key Generation Center, whose public parameters are publicly known.
The novelty of our protocol is that it can be implemented over any cyclic group of prime order, where the Diffie-Hellman problem is supposed to be hard. It does not require the computation of expensive bilinear maps, or additional assumptions such as factoring or RSA.
The protocol is extremely efficient, requiring only twice the amount of bandwidth and computation of the *unauthenticated* basic Diffie-Hellman protocol. The design of our protocol was inspired by MQV (the most efficient authenticated Diffie-Hellman based protocol in the public-key model) and indeed its performance is competitive with respect to MQV (especially when one includes the transmission and verification of certificates in the MQV protocol, which are not required in an id-based scheme). Our protocol requires a single round of communication in which each party sends only 2 group elements: a very short message, especially when the protocol is implemented over elliptic curves.
We provide a full proof of security in the Canetti-Krawczyk security model for key exchange, including a proof that our protocol satisfies additional security properties such as forward secrecy, and resistance to reflection and key-compromise impersonation attacks.

1 Introduction

Identity-based cryptography was introduced in 1984 by Adi Shamir [34]. The goal was to simplify the management of public keys and in particular the association of a public key to the identity of its holder. Usually such binding of a public key to an identity is achieved by means of *certificates* which are signed statements by trusted third parties that a given public key belongs to a user. This requires users

* An extended abstract of this paper appears in the proceedings of CT-RSA 2010 [18].
** Part of this work have been done while student at University of Catania and visiting NYU and IBM Research.

to obtain and verify certificates whenever they want to use a specific public key, and the management of public key certificates remains a technically challenging problem.

Shamir's idea was to allow parties to use their identities as public keys. An id-based scheme works as follows. A trusted *Key Generation Center* (KGC) generates a master public/secret key pair, which is known to all the users. A user with identity ID receives from the KGC a secret key S_{ID} which is a function of the string ID and the KGC's secret key (one can think of S_{ID} as a signature by the KGC on the string ID). Using S_{ID} the user can then perform cryptographic tasks. For example in the case of *id-based encryption* any party can send an encrypted message to the user with identity ID using the string ID as a public key and the user (and only the user and the KGC) will be able to decrypt it using S_{ID}. Note that the sender can do this even if the recipient has not obtained yet his secret key from the KGC. All the sender needs to know is the recipient's identity and the public parameters of the KGC. This is the major advantage of id-based encryption.

Id-Based Key Agreement and its Motivations. This paper is concerned with the task of *id-based key agreement*. Here two parties Alice and Bob, with identities A, B and secret keys S_A, S_B respectively, want to agree on a common shared key, in an *authenticated* manner (i.e. Alice must be sure that once the key is established, only Bob knows it – and viceversa). Since key agreement is inherently an interactive protocol (both parties are "live" and ready to establish a session) there is a smaller gain in using an id-based solution: indeed certificates and public keys can be easily sent as part of the protocol communication.

Yet the ability to avoid sending and verifying public key certificates is a significant practical advantage (see e.g. [37]). Indeed known shortcomings of the public key setting are the requirement of centralized certification authorities, the need for parties to cross-certify each other (via possibly long certificate chains), and the management of some form of large-scale coordination and communication (possibly on-line) to propagate certificate revocation information. Identity-based schemes significantly simplify identity management by bypassing the certification issues. All a party needs to know in order to generate a shared key is its own secret key, the public information of the KGC, and the identity of the communication peer (clearly, the need to know the peer's identity exists in any scheme including a certificate-based one).

Another advantage of identity-based systems is the versatility with which identities may be chosen. Since identities can be arbitrary string, they can be selected according to the function and attributes of the parties (rather than its actual "name"). For example in vehicular networks a party may be identified by its location ("the checkpoint at the intersection of a and b") or in military applications a party can be identified by its role ("platoon x commander"). This allows parties to communicate securely with the intended recipient even without knowing its "true" identity but simply by the definition of its function in the network.

Finally, identities can also include additional attributes which are temporal in nature: in particular an "expiration date" for an identity makes revocation of the corresponding secret key much easier to achieve.

For the reasons described above, id-based KA protocols are very useful in many systems where bandwidth and computation are at a premium (e.g. sensor networks), and also in ad-hoc networks where large scale coordination is undesirable, if not outright impossible. Therefore it is an important question to come up with very efficient and secure id-based KA protocols.

Previous work on id-based key agreement. Following Shamir's proposal of the concept of id-based cryptography, some early proposals for id-based key agreement appeared in the literature: we refer in particular to the works of Okamoto [29] (later improved in [30]) and Gunther [22]. A new impetus to this research area came with the breakthrough discovery of bilinear maps and their application to id-based encryption in [5]: starting with the work of Sakai *et al.* [33] a large number of id-based KA protocols were designed that use pairings as tool. We refer the readers to [6] and [12] for surveys of these pairing-based protocols.

The main problem with the current state of the art is that many of these protocols lack a proof of security, and some have even been broken. Indeed only a few (e.g., [8, 38]) have been proven according to a formal definition of security.

Our Contribution. By looking at prior work we see that provably secure id-based KAs require either groups that admit bilinear maps [8, 38], or to work over a composite RSA modulus [30].

This motivated us to ask the following question: can we find an efficient and provably secure id-based KA protocol such that:

1. it that can be implemented over *any* cyclic group in which the Diffie-Hellman problem is supposed to be hard. The advantages of such a KA protocol would be several, in particular: (i) it would avoid the use of computationally expensive pairing computations; (ii) it could be implemented over much smaller groups (since we could use 'regular' elliptic curves, rather than the ones that admit efficient pairings computations for high security levels, or the group Z_N^* for a composite N needed for Okamoto-Tanaka).
2. it is more efficient than any KA protocols in the public key model (such as MQV [27]), when one includes the transmission and verification of certificates which are not required in an id-based scheme. This is a very important point since, as we pointed out earlier in this Section, id-based KA protocols are only relevant if they outperform PKI based ones in efficiency.

Our new protocol presented in this paper (whose description appears in Figure 1), achieves all these features.

It can be implemented over any cyclic group over which the Diffie-Hellman problem is assumed to be hard. In addition it requires an amount of bandwidth and computation similar to the *unauthenticated* basic Diffie-Hellman protocol. Indeed our new protocol requires a single round of communication in which each party sends just two group elements (as opposed to one in the Diffie-Hellman

The IB-KA Protocol

Setting: A Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q together with a random generator $g \in \mathbb{G}$ and an exponent $x \xleftarrow{\$} \mathbb{Z}_q$. KGC publishes $\mathbb{G}, q, g, y = g^x$ and two hash functions H_1, H_2.

Key Derivation: A user with identity U receives its private key (r_U, s_U) from the KGC computed as the Schnorr's signature of the string U under public key y. That is $r_U = g^{k_U}$ for $k_U \xleftarrow{\$} \mathbb{Z}_q$ and $s_U = k_U + xH_1(U, r_U) \bmod q$.

Key agreement: A and B choose ephemeral private exponents t_A and t_B, respectively.

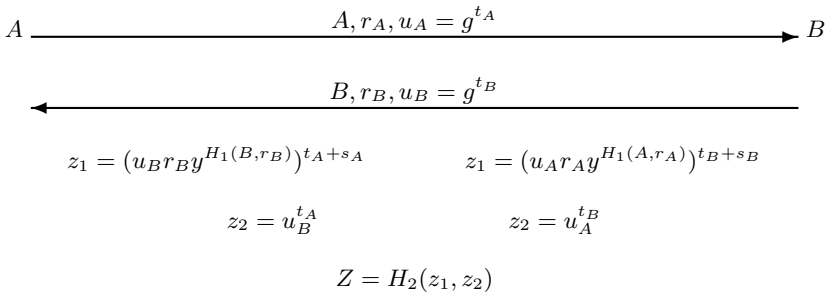

Fig. 1. A and B share session key Z. See Section 3 for more specific details.

protocol). Each party must compute four exponentiations to compute the session key (as opposed to two in the Diffie-Hellman protocol).

A similar favorable comparison holds with the Okamoto-Tanaka protocol in [30]. While that protocol requires only two exponentiations, it does works over Z_N^* therefore requiring the use of a larger group size, which almost totally absorbs the computational advantage, and immediately implies a much larger bandwidth requirement. Detailed efficiency comparisons to other protocols in the literature are discussed in Section 6.

We present a full proof of security of our protocol in the Canetti-Krawczyk security model. Our results hold in the random oracle model, under the Strong Diffie-Hellman Assumption. We also present some variations of our protocol that can be proven secure under the basic Computational Diffie-Hellman Assumption. Our protocol can be proven to satisfy additional desirable security properties such as perfect forward secrecy[3], and resistance to reflection and key-compromise impersonation attacks.

[3] We can prove PFS only in the case the adversary was passive in the session that he is attacking – though he can be active in other sessions. As proven by Krawczyk in [26], this is the best that can be achieved for 1-round protocols with *implicit* authentication, such as ours.

Our Approach. The first direction we took in our approach was to attempt to analyze the id-based KA protocols by Gunther [22] and Saeednia [32]. They also work over any cyclic group where the Diffie-Hellman problem is assumed to be hard, but their protocols lack a formal proof of security. While the original protocols cannot be shown to be secure, we were able to prove the security of modified versions of them. Nevertheless these two protocols were not very satisfactory solutions for the problem we had set out to solve, particularly for reasons of efficiency since they required a large number of exponentiations, which made them less efficient than say MQV with certificates.

Our protocol improves over these two protocols by using Schnorr's signatures [35], rather than ElGamal, to issue secret keys to the users. The simpler structure of Schnorr's signatures permits a much more efficient computation of the session key, resulting in less exponentiations and a single round protocol. Our approach was inspired by the way the MQV protocol [27] achieves *implicit authentication* of the session key. Indeed our protocol can be seen as an id-based version of the MQV protocol.

Organization. In Section 2 we recall a few preliminary notions, such as the Canetti-Krawczyk security model for KA protocols, and the computational assumptions that we will use in our proofs. Our new protocol is described in Section 3, and its proof in Section 4. Comparison to other id-based KA protocols is in Section 6. The modifications and proofs of the Gunther and Saeednia protocols are in Section 7.

2 Preliminaries

In this section we present some standard definitions needed in the rest of the paper.

Let $\mathbb{N}$ the set of natural numbers. We will denote with $\ell \in \mathbb{N}$ the security parameter. The participants to our protocols are modeled as probabilistic Turing machines whose running time is bounded by some polynomial in ℓ. If S is a set, we denote with $s \stackrel{\$}{\leftarrow} S$ the process of selecting an element uniformly at random from S.

Definition 1 (Negligible function). *A function $\epsilon(\ell)$ is said to be negligible if for every polynomial $p(\ell)$ there exists a positive integer $c \in \mathbb{N}$ such that $\forall \ell > c$ we have $\epsilon(\ell) < 1/p(\ell)$.*

In the following assume $\mathbb{G}$ to be a cyclic multiplicative group of order q where q is a ℓ-bit long prime. We assume that there are efficient algorithms to perform multiplication and membership test in $\mathbb{G}$. Finally we denote with g a generator of $\mathbb{G}$.

Assumption 1 (Computational Diffie-Hellman [16]) *We say that the Computational Diffie-Hellman (CDH) Assumption (for $\mathbb{G}$ and g) holds if for any probabilistic polynomial time adversary $\mathcal{A}$ the probability that $\mathcal{A}$ on input $(\mathbb{G}, g, g^u, g^v)$ outputs W such that $W = g^{uv}$ is negligible in ℓ. The probability of success of $\mathcal{A}$ is taken over the uniform random choice of $u, v \in \mathbb{Z}_q$ and the coin tosses of $\mathcal{A}$.*

The CDH Assumption has a *Decisional* version in which no adversary can actually *recognize* the value g^{uv} when given g^u, g^v. In the proof of our basic protocol we are going to need the ability to perform such decisions when one of the two elements is fixed, while still assuming that the CDH holds. The assumption below basically says that the CDH Assumption still holds in the presence of an oracle $\mathsf{DH}(U, \cdot, \cdot)$ that solves the decisional problem[4] for a fixed U.

Assumption 2 (Strong-DH Assumption [1]) *We say that the Strong-DH (SDH) Assumption holds (for $\mathbb{G}$ and g) if the CDH Assumption holds even in the presence of an oracle $\mathsf{DH}(U, \cdot, \cdot)$ that on input two elements $\hat{V}, \hat{W}$ in the group generated by g, output "yes" if and only if $\hat{W}$ is the Diffie-Hellman of U and $\hat{V}$.*

Finally we recall the Gap-DH assumption that is stronger than the Strong-DH in that the oracle can be queried on an arbitrary triple (U, V, W).

Assumption 3 (Gap-DH Assumption) *We say that the Gap-DH Assumption holds (for $\mathbb{G}$ and g) if the CDH Assumption holds even in the presence of an oracle $\mathsf{DH}(\cdot, \cdot, \cdot)$ that on input three elements $U = g^u, V = g^v, W = g^w$ in the group generated by g, output "yes" if and only if $W = g^{uv}$.*

The oracle DH for the Decisional DH problem exists for some groups $\mathbb{G}$, e.g. the ones that admit a bilinear map. We stress, however that we need the oracle *only for the proof of security, and it is not needed in the execution of the protocol by the real-life parties.* This means that we can efficiently implement our protocol over any cyclic group $\mathbb{G}$.

The question, then, is the real-life meaning of a proof under the Strong-DH assumption when the protocol is implemented over a group $\mathbb{G}$ that does not admit such oracle DH. If we prove the security of our protocol under the SDH assumption, then if a successful adversary can be constructed one of two things must be true:

1. either the CDH Assumption is false
2. or we have a proof that the hardness of the Decisional problem is implied by the CDH Assumption (in other words the CDH and DDH Assumptions are equivalent). Indeed in this case the CDH holds, and the protocol is insecure, this means that the oracle DH cannot exists (if it existed, given that the CDH holds, the protocol should be secure).

In other words, while proofs under the Strong-DH assumption do not necessarily offer a constructive cryptanalysis of a conjectured hard problem in case of a successful attack, they do offer the "dual" ability to prove the equivalence of the CDH Assumption (with any other additional assumption required by the proof) with the DDH Assumption over the underlying group.

[4] We remark that in recent papers the name strong Diffie-Hellman assumption was used to denote a different conjecture defined over bilinear groups [4]. In this paper, we refer to the original terminology from [1]

2.1 Definitions for identity-based key agreement

The security of our protocols is analyzed in a version of the Canetti-Krawczyk (CK) [9, 10] model for key agreement, adapted to the identity-based setting. We present an informal summary of the model and we refer the reader to [9, 10] for details.

An identity-based key-agreement protocol is runned by parties interacting in a network where each party is identified by a unique identity which is publicly known to all the other parties (e.g. Alice's identity is a string ID_A). In addition there exists a trusted entity called *Key Generation Center* (KGC) that generates the public parameters of the system and also issues secret keys to users associated with their public identities, e.g. the KGC generates a secret key SK_A associated to ID_A.

An instance of the protocol is called a *session*. The two parties participating in the session are called its *peers*. Each peer maintains a *session state* which contains incoming and outgoing messages and its random coins. If the session is *completed* then each party outputs a *session key* and erases its session state. A session may also be *aborted*. In this case no session key is generated.

Each party assigns an unique identifier to a session he is participating in. For simplicity, we assume it to be the quadruple $(Alice, Bob, m_{Out}, m_{In})$ where Alice is the identity of the party, Bob its peer, m_{Out} and m_{In} are the outgoing and incoming messages, respectively, for Alice. If Alice holds a session $(Alice, Bob, m_{Out}, m_{In})$ and Bob holds a session $(Bob, Alice, m_{In}, m_{Out})$ then the two sessions are *matching*.

The adversary The CK definition models a very realistic adversary which basically controls all communication in the network. In particular it can intercept and modify messages exchanged by parties, delay or block their delivery, inject its own messages, schedule sessions etc. The adversary is allowed to choose the identities of the parties, and obtain private keys from the KGC for identities of its choice.

Finally we allow the adversary to access some of the parties' secret information, via the following attacks: *party corruption*, *state-reveal queries* and *session-key queries*. When an adversary *corrupts* a party, it learns its private information (the private key and all session states and session keys currently stored), and it later controls its actions. In a *state-reveal query* to a party running a session, the adversary learns the session state for that session (since we assume that session states are erased at the end of the session, such query makes sense only against sessions that are still incomplete). Finally a *session-key query* allows the adversary to learn the session key of a complete session. A session is called *exposed* if it or its matching session (if existing) is compromised by one of the attacks above.

Security Definition Let $\mathcal{A}$ be a probabilistic polynomial time adversary modeled as described above. Then consider the following experiment running $\mathcal{A}$.

At the beginning of the game the adversary receives as input the public parameters of the system (generated by the KGC) and then can perform all the actions described in the section before.

At some point, $\mathcal{A}$ chooses a *test session* among all the completed and unexposed sessions. We toss a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$ we give $\mathcal{A}$ the session key K_0 of the test session. Otherwise we take a random session key K_1 and provide $\mathcal{A}$ with K_1.

After having received K_b, the adversary can continue to perform its actions against the protocol with the exception that it cannot expose the test session. At the end of the game $\mathcal{A}$ outputs a bit b' as its guess for b.

Definition 2. *An identity-based key-agreement protocol is said to be secure if for any PPT adversary $\mathcal{A}$ the following holds:*

1. *if two uncorrupted parties complete matching sessions then they output the same session key with overwhelming probability;*
2. *the probability that $\mathcal{A}$ guesses the correct b in the above experiment is at most $1/2$ plus a negligible fraction of the security parameter.*

We define the advantage of $\mathcal{A}$ as $\mathbf{Adv}_{\mathcal{A}}^{IB-KA} = |\Pr[b = b'] - 1/2|$.

Additional security properties In addition to the notion of session key security presented above, an identity-based key-agreement protocol should satisfy other important properties: resistance to *reflection attacks*, *forward secrecy* and resistance to *key-compromise impersonation* attacks.

A *reflection attack* occurs when an adversary can compromise a session in which the two parties have the same identity (and the same private key). Though, at first glance, this seems to be only of theoretical interest, there are real-life situations in which this scenario occurs. For example consider the case when Alice is at her office and wants to establish a secure connection with her PC at home, therefore running a session between two computers with the same identity and private key.

We would also like to achieve resistance to *key compromise impersonation* (KCI) attacks. Suppose that the adversary learns Alice's private key. Then, it is trivial to see that this knowledge enables the adversary to impersonate Alice to other parties. A KCI attack can be carried out when the knowledge of Alice's private key allows the adversary to impersonate another party to Alice.

Finally, *Forward secrecy* is probably the most important additional security property we would like to achieve. We say that a KA protocol has forward secrecy, if after a session is completed and its session key erased, the adversary cannot learn it *even if* it corrupts the parties involved in that session. In other words, learning the private keys of parties should not jeopardize the security of past completed sessions.

A relaxed notion of forward secrecy (which we call *weak*) assumes that only past sessions in which the adversary was passive (i.e. did not choose the messages) are not jeopardized.

3 The New Protocol IB-KA

Protocol setup. The Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q (where q is ℓ-bits long), a random generator $g \in \mathbb{G}$ and two hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_2 : \mathbb{Z}_q \times \mathbb{Z}_q \to \{0,1\}^\ell$. Then it picks a random $x \xleftarrow{\$} \mathbb{Z}_q$ and sets $y = g^x$. Finally the KGC outputs the public parameters $MPK = (\mathbb{G}, g, y, H_1, H_2)$ and keeps the master secret key $MSK = x$ for itself.

Key Derivation. A user with identity ID receives, as its secret key, a Schnorr's signature [35] of the message $m = ID$ under public key y. More specifically, the KGC after verifying the user's identity, creates the associated secret key as follows. First it picks a random $k \xleftarrow{\$} \mathbb{Z}_q$ and sets $r_{ID} = g^k$. Then it uses the master secret key x to compute $s_{ID} = k + H_1(ID, r_{ID})x$. (r_{ID}, s_{ID}) is the secret key returned to the user. The user can verify the correctness of its secret key by using the public key y and checking the equation $g^{s_{ID}} \stackrel{?}{=} r_{ID} \cdot y^{H_1(ID, r_{ID})}$.

A protocol session. Let's assume that Alice wants to establish a session key with Bob. Alice owns secret key (r_A, s_A) and identity A while Bob has secret key (r_B, s_B) and identity B.

Alice selects a random $t_A \xleftarrow{\$} \mathbb{Z}_q$, computes $u_A = g^{t_A}$ and sends the message $\langle A, r_A, u_A \rangle$ to Bob. Analogously Bob picks a random $t_B \xleftarrow{\$} \mathbb{Z}_q$, computes $u_B = g^{t_B}$ and sends $\langle B, r_B, u_B \rangle$ to Alice. After the parties have exchanged these two messages, they are able to compute the same session key $Z = H_2(z_1, z_2)$. In particular Alice computes

$$z_1 = (u_B r_B y^{H_1(B, r_B)})^{t_A + s_A} \quad \text{and} \quad z_2 = u_B^{t_A}.$$

On the other hand Bob computes

$$z_1 = (u_A r_A y^{H_1(A, r_A)})^{t_B + s_B} \quad \text{and} \quad z_2 = u_A^{t_B}.$$

It is easy to see that both the parties are computing the same values $z_1 = g^{(t_A + s_A)(t_B + s_B)}$ and $z_2 = g^{t_A t_B}$. The state of a user ID during a protocol session contains only the fresh random exponent t_{ID}. We assume that after a session is completed, the parties erase their state and keep only the session key.

Remark: In the next section we show that protocol IB-KA is secure under the Strong Diffie-Hellman Assumption. However, in Section 5 we show how to modify IB-KA to obtain security under the basic CDH Assumption, at the cost of a slight degradation in efficiency.

4 Security Proof

We prove the security of the protocol by a usual reduction argument. More precisely we show how to reduce the existence of an adversary breaking the protocol into an algorithm that is able to break the SDH Assumption with non-negligible probability. The adversary is modeled as a CK attacker: (see Section 2.1 for

details): in particular it will choose a test session among the complete and unexposed sessions and will try to distinguish between its real session key and a random one.

In our reduction we will make use of the General Forking Lemma, stated by Bellare and Neven in [2]. It follows the original forking lemma of Pointcheval and Stern [31], but, unlike that, it makes no mention of signature schemes and random oracles. In this sense it is more general and it can be used to prove the security of our protocol. We briefly recall it in the following.

Lemma 1 (General Forking Lemma [2]). *Fix an integer $Q \geq 1$ and a set H of size $|H| \geq 2$. Let $\mathcal{B}$ be a randomized algorithm that on input $x, h_1, \ldots, h_Q$ returns a pair (J, σ) where $J \in \{0, \ldots, Q\}$ and σ is referred as* side output. *Let IG be a randomized algorithm called the input generator. Let $acc_{\mathcal{B}} = \Pr[J \geq 1 : x \xleftarrow{\$} IG, h_1, \ldots, h_Q \xleftarrow{\$} H; (J, \sigma) \xleftarrow{\$} \mathcal{B}(x, h_1, \ldots, h_Q)]$ be the accepting probability of $\mathcal{B}$.*

The forking algorithm $F_{\mathcal{B}}$ associated to $\mathcal{B}$ is the randomized algorithm that takes as input x and proceeds as follows:

Algorithm $F_{\mathcal{B}}(x)$
Pick random coins ρ for $\mathcal{B}$
$h_1, \ldots, h_Q \xleftarrow{\$} H$
$(J, \sigma) \xleftarrow{\$} \mathcal{B}(x, h_1, \ldots, h_Q; \rho)$
If $J = 0$ then return $(0, \perp, \perp)$
$h'_1, \ldots, h'_Q \xleftarrow{\$} H$
$(J', \sigma') \xleftarrow{\$} \mathcal{B}(x, h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_Q; \rho)$
If ($J = J'$ and $h_J \neq h'_J$) then return $(1, \sigma, \sigma')$
Else return $(0, \perp, \perp)$.

Let $frk = \Pr[b = 1 : x \xleftarrow{\$} IG; (b, \sigma, \sigma') \xleftarrow{\$} F_{\mathcal{B}}(x)]$. *Then* $frk \geq acc_{\mathcal{B}}(\frac{acc_{\mathcal{B}}}{Q} - \frac{1}{|H|})$.

Roughly speaking the lemma says that if an algorithm $\mathcal{B}$ accepts with some non-negligible probability, then a "rewind" of $\mathcal{B}$ is likely to accept with a polynomially related probability (more specifically squared). If we look at the details of this lemma, the intuitions are that: (1) $h_1, \ldots, h_Q$ can be seen as the set of replies to random oracle queries made by the original adversary and (2) the forking algorithm implements the rewinding. Moreover it is important that in $F_{\mathcal{B}}$ the two executions of $\mathcal{B}$ are run with the same random coins ρ. We defer to [2] for the proof of the lemma.

Theorem 4. *Under the Strong-DH Assumption, if we model H_1 and H_2 as random oracles, then protocol* IB-KA *is a secure identity-based key agreement protocol.*

Proof. For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that has non-negligible advantage ϵ into breaking the protocol IB-KA , then we show how to build a solver algorithm S for the CDH problem.

In our reduction we will proceed into two steps. First, we describe an intermediate algorithm $\mathcal{B}$ (i.e. the simulator) that interacts with the IB-KA adversary $\mathcal{A}$ and returns a side output σ. Second, we will show how to build an algorithm S that exploits $F_{\mathcal{B}}$, the forking algorithm associated with $\mathcal{B}$, to solve the CDH problem under the Strong-DH Assumption.

$\mathcal{B}$ receives as input a tuple $(\mathbb{G}, g, U, V)$, where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$, and a set of random elements $h_1, \ldots, h_Q \in \mathbb{Z}_q$. The simulator is also given access to a DH oracle $\mathsf{DH}(U, \cdot, \cdot)$ that on input $(\hat{V}, \hat{W})$ answers "yes" if $(U, \hat{V}, \hat{W})$ is a valid DDH tuple . The side output of $\mathcal{B}$ is $\sigma \in \mathbb{G} \times \mathbb{Z}_q$ or $\perp$. Let n be an upper bound to the number of sessions of the protocol run by the adversary $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by $\mathcal{A}$ to the random oracles H_1, H_2 respectively. Moreover, let Q_c be the number of users corrupted by $\mathcal{A}$ and $Q = Q_1 + Q_c + 1$.

Algorithm $\mathcal{B}^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V, h_1, \ldots, h_Q)$

Initialize $ctr \leftarrow 0; bad \leftarrow false$; empty tables $\overline{H_1}, \overline{H_2}$;

Run $\mathcal{A}$ on input $(\mathbb{G}, g, y = U)$ as the public parameters of the protocol and simulates the protocol's environment for $\mathcal{A}$ as follows:

Guess the test session by choosing at random the user (let us call him Bob) and the order number of the test session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

H_2 **queries** On input a pair (z_1, z_2):

If $\overline{H_2}[z_1, z_2] = \perp$: choose a random string $Z \in \{0,1\}^\ell$ and store $\overline{H_2}[z_1, z_2] = Z$

Return $\overline{H_2}[z_1, z_2]$ to $\mathcal{A}$

H_1 **queries** On input (ID, r):

If $\overline{H_1}[ID, r] = \perp$, then $ctr \leftarrow ctr + 1; \overline{H_1}[ID, r] = h_{ctr}$

Return $\overline{H_1}[ID, r]$ to $\mathcal{A}$

Party Corruption When $\mathcal{A}$ asks to corrupt party $ID \neq B$, then:

$ctr \leftarrow ctr + 1; s \stackrel{\$}{\leftarrow} \mathbb{Z}_q; r = g^s y^{-h_{ctr}}$

If $\overline{H_1}[ID, r] \neq \perp$ then $bad \leftarrow true$

Store $\overline{H_1}[ID, r] = h_{ctr}$ and return (r, s) as ID's private key.

For the case of Bob, the simulator simply chooses the r_B component of Bob's private key by picking a random $k_B \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and setting $r_B = g^{k_B}$. We observe that in this case $\mathcal{B}$ is not able to compute the corresponding s_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key.

Simulating sessions First we describe how to simulate sessions different from the test session. Here the main point is that the adversary is allowed to ask session-key queries and thus the simulator must be able to produce the correct session key for each of these sessions. The simulator has full information about all the users' secret keys except Bob. Therefore $\mathcal{B}$ can easily simulate all the protocol sessions that do not include Bob, and answer any of the attacker's queries about

these sessions. Hence we concentrate on describing how $\mathcal{B}$ simulates interactions with Bob.

Assume that Bob has a session with Charlie (whose identity is the string C). If Charlie is an uncorrupted party this means that $\mathcal{B}$ will generate the messages on his behalf. In this case $\mathcal{B}$ knows Charlie's secret key and also has chosen his ephemeral exponent t_C. Thus it is trivial to see that $\mathcal{B}$ has enough information to compute the correct session key. The case when the adversary presents a message $\langle C, r_C, u_C \rangle$ to Bob as coming from Charlie is more complicated. Here is where $\mathcal{B}$ makes use of the oracle $\mathsf{DH}(y, \cdot, \cdot)$ to answer a session-key query about this session. The simulator replies with a message $\langle B, r_B, u_B = g^{t_B} \rangle$ where t_B is chosen by $\mathcal{B}$. Recall that the session key is $H_2(z_1, z_2)$ with $z_1 = g^{(s_C + t_C)(s_B + t_B)}$ and $z_2 = u_C^{t_B}$. So z_1 is the Diffie-Hellman result of the values $u_C g^{s_C}$ and $u_B g^{s_B}$, where $g^{s_C} = r_C y^{H_1(C, r_C)}$ and $g^{s_B} = r_B y^{H_1(B, r_B)}$ can be computed by the simulator. Notice also that the simulator knows t_B and k_B (the discrete log of r_B in base g). Therefore it checks if $\overline{H_2}[z_1, z_2] = Z$ where $z_2 = u_C^{t_B}$ and $\mathsf{DH}(y, u_C g^{s_C}, \bar{z_1}) =$ "yes" where $\overline{z_1} = \frac{z_1}{(u_C g^{s_C})^{(k_B + t_B) H_1(B, r_B)^{-1}}}$. If $\mathcal{B}$ finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \xleftarrow{\$} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query (z_1, z_2) to H_2, if (z_1, z_2) satisfies the equation above it sets $\overline{H_2}[z_1, z_2] = \zeta$ and answers with ζ. This makes the oracle's answers consistent.

In addition observe that the simulator can easily answer any state reveal queries as it chooses the fresh exponents on its own.

Simulating the test session Let $\langle B, \rho_B, u_B = g^{t_B} \rangle$ be the message from Bob to Alice sent in the test session. We notice that such message may be sent by the adversary who is trying to impersonate Bob. In this case $\mathcal{A}$ may use a value $\rho_B = g^{\lambda_B}$ of its choice as the public component of Bob's private key (i.e. different than $r_B = g^{k_B}$ which $\mathcal{B}$ simulated and for which it knows k_B). $\mathcal{B}$ responds with the message $\langle A, r_A, u_A = V \rangle$ as coming from Alice. Finally $\mathcal{B}$ provides $\mathcal{A}$ with a random session key.

Run until $\mathcal{A}$ halts and outputs its decision bit

If $\overline{H_1}[B, \rho_B] = \perp$ then set $ctr \leftarrow ctr + 1$ and $\overline{H_1}[B, \rho_B] = h_{ctr}$

If $bad = true$ then return $(0, \perp)$

Let $i \in \{1, \ldots, Q\}$ such that $H_1(B, \rho_B) = h_i$

Let $Z = H_2(z_1, z_2)$ be the correct session key for the test session where $z_1 = (u_A r_A y^{H_1(A, r_A)})^{(t_B + \lambda_B + x h_i)}$ and $z_2 = u_A^{t_B}$.

If $\mathcal{A}$ has success into distinguishing Z from a random value it must necessarily query the correct pair (z_1, z_2) to the random oracle H_2. This means that $\mathcal{B}$ can efficiently find the pair (z_1, z_2) in the table $\overline{H_2}$ using the Strong-DH oracle.

Compute $\tau = \frac{z_1}{z_2 (u_B \rho_B y^{h_i})^{s_A}} = \rho_B^v W^{h_i}$

Return $(i, (\tau, h_i))$

Let IG be the algorithm that generates a random Diffie-Hellman tuple $(\mathbb{G}, g, U, V)$ and $acc_{\mathcal{B}}$ be the accepting probability of $\mathcal{B}$.[5] Then we have that:

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \Pr[bad = true].$$

The probability that $bad = true$ is the probability that the adversary has guessed the "right" r for a corrupted party ID before corrupting it, in one of the H_1 oracle queries beforehand. Since r is uniformly distributed the probability of guessing it is $1/q$, and since the adversary makes at most Q queries to H_1 and corrupts at most Q_c parties (and $q > 2^\ell$) we have that

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \frac{Q_c(Q)}{2^\ell}.$$

which is still non-negligible, since ϵ is non-negligible.

Once we have described the algorithm $\mathcal{B}$ we can now show how to build a solver algorithm S that can exploit $F_{\mathcal{B}}$, the forking algorithm associated with the above $\mathcal{B}$.

The algorithm S plays the role of a CDH solver under the Strong-DH Assumption. It receives as input a CDH tuple $(\mathbb{G}, g, U, V)$ where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$. S is also given access to a decision oracle $\mathsf{DH}(U, \cdot, \cdot)$ that on input $(\hat{V}, \hat{W})$ answers "yes" if $(U, \hat{V}, \hat{W})$ is a valid DH tuple .

Algorithm $S^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V)$
  $(b, \tau, \tau') \xleftarrow{\$} F_{\mathcal{B}}^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V)$
  If $b = 0$ then return 0 and halt
  Parse σ as (τ, h) and σ' as (τ', h')
  Return $W = (\tau/\tau')^{(h-h')^{-1}}$

If the forking algorithm $F_{\mathcal{B}}$ has success, this means that there exist random coins ρ, an index $J \geq 1$ and $h_1, \ldots, h_Q, h'_J, \ldots, h'_Q \in \mathbb{Z}_q$ with $h = h_J \neq h'_J = h'$ such that: the first execution of $\mathcal{B}(\mathbb{G}, g, U, V, h_1, \ldots, h_Q; \rho)$ outputs $\tau = \rho_B^v W^h$ where $\overline{H_1}[B, \rho_B] = h$; the second execution of $\mathcal{B}(\mathbb{G}, g, U, V, h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_Q; \rho)$ outputs $\tau' = (\rho'_{B'})^v W^{h'}$ where $\overline{H_1}[B', \rho'_{B'}] = h'$. Since the two executions of $\mathcal{B}$ are the same until the response to the J-th query to H_1, then we must have $B = B'$ and $\rho_B = \rho'_{B'}$. Thus it is easy to see that S achieves its goal by computing $W = (\tau/\tau')^{\frac{1}{h-h'}} = g^{uv}$.

Finally, by the General Forking Lemma, we have that if $\mathcal{A}$ has non-negligible advantage into breaking the security of IB-KA , then S's success probability is also non-negligible.

4.1 Additional Security Properties of IB-KA

Below we describe the additional security properties enjoyed by IB-KA .

[5] We say that $\mathcal{B}$ accepts if it outputs (J, σ) such that $J \geq 1$.

Forward secrecy The following theorem shows that the protocol IB-KA satisfies *weak forward secrecy* as described in Section 2.1.

Theorem 5. *Let $\mathcal{A}$ be a PPT adversary that is able to break the weak forward secrecy of the* IB-KA *protocol with advantage ϵ. Let n be the an upper bound to the number of sessions of the protocol run by $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by the adversary to the random oracles H_1, H_2 respectively. Then we can solve the CDH problem with probability at least $\epsilon/(nQ_2)$.*

Proof. For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that is able to break the weak forward secrecy of the protocol IB-KA with non-negligible advantage ϵ. Then we show how to build a simulator S that uses $\mathcal{A}$ to solve the CDH problem with probability at least ϵ/nQ_2. S receives as input a tuple $(\mathbb{G}, g, U, V)$ where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$. The simulator plays the role of the CDH solver and its goal it to compute the value $W = g^{uv}$.

Setup. S sets up a simulated execution of the protocol, with simulated KGC, users and sessions. First of all S defines the public parameters of the protocol simulating the KGC. So it chooses a random $x \xleftarrow{\$} \mathbb{Z}_q$ and sets $y = g^x$. Then it provides the adversary with input $(\mathbb{G}, g, y)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r_{ID}) S chooses a random $e_{ID} \xleftarrow{\$} \mathbb{Z}_q$ and sets $H_1(ID, r_{ID}) = e_{ID}$.

Since S knows the master secret key x, it can simulate the KGC in full, and give secret keys to all the parties in the network, including answering private key queries from the adversary.

At the beginning of the game S guesses the test session and its peers Alice and Bob.

Simulating protocol sessions. Sessions different from the test session are easily simulated since S knows all the information needed to compute the session keys and answer any query (including session key and state reveal queries) from the adversary.

Simulating the test session. We now show how to simulate the test session in order to extract $W = g^{uv}$ from the adversary. Since in this game the adversary is assumed to be passive during the test session, the parties (i.e. the simulator in this case) choose the messages exchanged in this session.

Let $(A, r_A, s_A), (B, r_B, s_B)$ be the identity information and the secret keys of Alice and Bob respectively (S knows these values). The simulator sets Alice's message as $(A, r_A, u_A = U)$ while the one from Bob is $(B, r_B, u_B = V)$. S is implicitly setting $t_A = u, t_B = v$. In this case the correct session key is $Z = H_2(g^{(s_A+u)(s_B+v)}, g^{uv})$. Since H_2 is modeled as a random oracle, if $\mathcal{A}$ has success into distinguishing Z from a random value, it must have queried H_2 on the correct input $(z_1 = g^{(s_A+u)(s_B+v)}, z_2 = g^{uv})$. Thus S can choose a random value among all the queries that it received from the adversary. Since the number

of queries Q_2 is polynomially bounded, S can find the correct $z_2 = W$ with non-negligible probability ϵ/nQ_2. This completes the proof of this case[6].

Resistance to reflection attacks A *reflection attack* occurs when an adversary can compromise a session in which the two parties have the same identity (and the same private key). Though, at first glance, this seems to be only of theoretical interest, there are real-life situations in which this scenario occurs. For example consider the case when Alice is at her office and wants to establish a secure connection with her PC at home, therefore running a session between two computers with the same identity and private key.

Here we extend the proof of security given in Section 4 to support reflection attacks. We observe that in the case when the test session has a matching session the proof remains valid even if the test session is between Bob and himself. On the other hand, when there is no matching session we have to show a little modification of the proof. In fact the current proof actually does not work when the adversary sends a message with the same value r_B provided by the KGC (for which the simulator knows the discrete logarithm k_B, but cannot compute the corresponding s_B). The issue is that the knowledge of s_B would be needed to extract the solution of the CDH problem.

We point out that a reflection attack using a value $\rho_B \neq r_B$ is captured by the current proof. Moreover it is reasonable to assume that a honest party refuses connections from itself that use a "wrong" key. However it is possible to adapt the proof in this specific case. In particular we can show that a successful run of the adversary enables the simulator to compute g^{u^2} instead of g^{uv}. As showed in [28] by Maurer and Wolf, such an algorithm can be easily turned into a solver for CDH.

In this section we show how to adapt the proof in this specific case. In particular, we show that a successful run of the adversary enables the simulator to compute g^{u^2} instead of g^{uv}. As showed in [28] by Maurer and Wolf, such an algorithm can be easily turned into a solver for CDH.

Let us consider the following modification of the proof given in Section 4. If in the test session the adversary sends a message from Bob to Bob of type $\langle B, r_B, u_B = g^{t_B} \rangle$ then the simulator picks a random $e \xleftarrow{\$} \mathbb{Z}_q$ and replies with message $\langle B, r_B, u'_B = U^e \rangle$. Let h^* be the random oracle response to $H_1(B, r_B)$. We observe that in this case the correct session key is the hash $Z = H_2(z_1, z_2)$ where $z_1 = g^{(k_B + uh^* + ue)(k_B + uh^* + t_B)}$ and $z_2 = g^{uet_B}$. If the adversary has success into distinguishing Z from a random value it must necessarily query the correct pair (z_1, z_2) to the random oracle H_2. This means that S can efficiently find the pair (z_1, z_2) in the table $\overline{H_2}$ using the Strong-DH oracle. Once it has recovered

[6] We could give the simulator access to the Strong-DH oracle DH, and then S could use it to "test" all queries to H_2 to find the correct W. The reduction would be tighter (removing the factor of Q_2^{-1} from the success probability) but would require the Strong-DH Assumption also in this case.

these values, it can compute:

$$g^{u^2} = \left(\frac{z_1}{g^{k_B^2} U^{2k_B h^*} U^{e k_B} u_B^{k_B} z_2 z_2^{h^*/e}} \right)^{\frac{1}{h^*(h^*+e)}}.$$

Resistance to Key Compromise Impersonation Suppose that the adversary learns Alice's private key. Then, it is trivial to see that this knowledge enables the adversary to impersonate Alice to other parties. A *key compromise impersonation* (KCI) attack can be carried out when the knowledge of Alice's private key allows the adversary to impersonate another party to Alice.

To see that the protocol IB-KA is resistant to KCI attacks it suffices to observe that in the proof of security, when the adversary tries to impersonate Bob to Alice, we are able to output Alice's private key whenever it is asked by the adversary. This means that the proof continues to be valid even in this case.

Ephemeral Key Compromise Impersonation A recent paper by Cheng and Ma [14] shows that our protocol is susceptible to an ephemeral key compromise attack. Roughly speaking this attack considers the case when the adversary can make state-reveal queries (in order to learn the ephemeral key of a user) even in the test session. Though the paper is correct, we point out that this kind of attack is not part of the standard Canetti-Krawczyk security model that is considered in this paper.

5 A protocol secure under CDH

The protocol IB-KA given in section Section 3 is proven secure under the Strong-DH Assumption. In this section we show how to modify that protocol so that its security can be based directly on CDH. The cost is a few more exchanged elements and a few more exponentiations. We call this modified protocol $\mathsf{2IB-KA}$.

The basic idea is to use the *Twin Diffie-Hellman* (2DH) Assumption introduced by Cash *et al.* in [11]. Informally 2DH states that an adversary which is given in input random $U_1, U_2, V \in \mathbb{G}$, should not be able to compute a pair (W_1, W_2) such that W_1 and W_2 are the DH of U_1, V and U_2, V respectively. It is easy to see that this assumption is equivalent to the well known CDH. The valuable contribution of [11] was to show that its "strong" version is equivalent to CDH too.

Informally the Strong-2DH assumption says that 2DH holds even in the presence of an oracle $\mathsf{2DH}(U_1, U_2, \cdot, \cdot, \cdot)$ that solves its decisional version for fixed U_1, U_2.

Therefore we modify the IB-KA protocol in such a way it can be proven secure under the Strong-2DH Assumption. Then, since Cash *et al.* proved in [11] that Strong-2DH and CDH are equivalent, we obtain a protocol secure under CDH.

In order to modify the protocol we apply the idea of "twinning" some elements so that the construction can be proven under the Strong-2DH assumption. The new protocol is almost the same as IB-KA except for the following:

– the master public key consists of two group elements y_1, y_2. This means that each user ID owns a secret key $(r^1_{ID}, s^1_{ID}, r^2_{ID}, s^2_{ID})$ which are two Schnorr's signatures of its identity corresponding to public keys y_1, y_2 respectively.
– each user ID generates two elements $u^1_{ID} = g^{t^1_{ID}}, u^2_{ID} = g^{t^2_{ID}}$ and sends $\langle r^1_{ID}, r^2_{ID}, u^1_{ID}, u^2_{ID}\rangle$.
– the session key of a session between users with identities A and B is

$$K = H(z_{11}, z_{12}, z_{21}, z_{22}, \omega_{11}, \omega_{12}, \omega_{21}, \omega_{22})$$

where $z_{11} = g^{(s^1_A + t^1_A)(s^1_B + t^1_B)}$, $z_{12} = g^{(s^1_A + t^1_A)(s^2_B + t^2_B)}$, $z_{21} = g^{(s^2_A + t^2_A)(s^1_B + t^1_B)}$, $z_{22} = g^{(s^2_A + t^2_A)(s^2_B + t^2_B)}$, $\omega_{11} = g^{t^1_A t^1_B}$, $\omega_{12} = g^{t^1_A t^2_B}$, $\omega_{21} = g^{t^2_A t^1_B}$ and $\omega_{22} = g^{t^2_A t^2_B}$.

It is also possible to instantiate a simpler version of this protocol in which the public key is only y as in IB-KA . This is slightly more efficient since a user has to send one less element. This variant can also be proven secure under the CDH provided that the adversary is not allowed to issue state-reveal queries.

The following theorem prove the security of the above protocol.

Theorem 6. *Under the CDH Assumption, if we model H_1 and H_2 as random oracles, then protocol* 2IB-KA *is a secure identity-based key agreement protocol.*

Proof. For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that has non-negligible advantage ϵ into breaking the protocol 2IB-KA , then we show how to build a solver algorithm S for the CDH problem under Strong-2DH.

In our reduction we will proceed into two steps. First, we describe an intermediate algorithm $\mathcal{B}$ (i.e. the simulator) that interacts with the IB-KA adversary $\mathcal{A}$ and returns a side output σ. Second, we will show how to build an algorithm S that exploits $F_{\mathcal{B}}$, the forking algorithm associated with $\mathcal{B}$, to solve the CDH problem under the Strong-2DH Assumption.

$\mathcal{B}$ receives as input a set of random elements $h_1, \ldots, h_{2Q} \in \mathbb{Z}_q$ and a tuple $(q, \mathbb{G}, g, U_1, U_2, V)$, where $U_1 = g^{u_1}, U_2 = g^{u_2}, V = g^v$ and u_1, u_2, v are random exponents in $\mathbb{Z}_q$. The simulator is also given access to a 2DH oracle $\mathsf{2DH}(U_1, U_2, \cdot, \cdot, \cdot)$ that on input $(\hat{V}, \hat{W}_1, \hat{W}_2)$ answers "yes" if $(U_1, \hat{V}, \hat{W}_1)$ and $(U_2, \hat{V}, \hat{W}_2)$ are valid DDH tuples. The side output of $\mathcal{B}$ is $\sigma \in \mathbb{G}^2 \times \mathbb{Z}_q^2$ or $\perp$. Let n be an upper bound to the number of sessions of the protocol run by the adversary $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by $\mathcal{A}$ to the random oracles H_1, H_2 respectively. Moreover, let Q_c be the number of users corrupted by $\mathcal{A}$ and $Q = Q_1 + Q_c + 1$.

Algorithm $\mathcal{B}^{\mathsf{2DH}(U_1, U_2, \cdot, \cdot, \cdot)}(q, \mathbb{G}, g, U_1, U_2, V, h_1, \ldots, h_{2Q})$
Initialize $ctr \leftarrow 0$; $bad \leftarrow false$; empty tables $\overline{H_1}, \overline{H_2}$;
Run $\mathcal{A}$ on input $(q, \mathbb{G}, g, y_1 = U_1, y_2 = U_2)$ as the public parameters of the protocol and simulates the protocol's environment for $\mathcal{A}$ as follows:
Guess the test session by choosing at random the user (let us call him Bob) and the order number of the test session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

H_2 **queries** On input a tuple $z = (z_{11}, z_{12}, z_{21}, z_{22}, \omega_{11}, \omega_{12}, \omega_{21}, \omega_{22})$:
If $\overline{H_2}[z] = \perp$: choose a random string $Z \in \{0,1\}^\ell$ and store $\overline{H_2}[z] = Z$
Return $\overline{H_2}[z]$ to $\mathcal{A}$

H_1 **queries** On input (ID, r):
If $\overline{H_1}[ID, r] = \perp$, then $ctr \leftarrow ctr + 1; \overline{H_1}[ID, r] = h_{ctr}$
Return $\overline{H_1}[ID, r]$ to $\mathcal{A}$

Party Corruption When $\mathcal{A}$ asks to corrupt party $ID \neq B$, then:
$ctr \leftarrow ctr+1; s^1_{ID} \stackrel{\$}{\leftarrow} \mathbb{Z}_q; r^1_{ID} = g^{s^1_{ID}} y^{-h_{ctr}}$; Store $\overline{H_1}[ID, r^1_{ID}] = h_{ctr}$
$ctr \leftarrow ctr + 1; s^2_{ID} \stackrel{\$}{\leftarrow} \mathbb{Z}_q; r^2_{ID} = g^{s^2_{ID}} y^{-h_{ctr}}$ Store $\overline{H_1}[ID, r^2_{ID}] = h_{ctr}$
If $\overline{H_1}[ID, r^1_{ID}] \neq \perp$ or $\overline{H_1}[ID, r^2_{ID}] \neq \perp$ then $bad \leftarrow true$
Return $(r^1_{ID}, s^1_{ID}, r^2_{ID}, s^2_{ID})$ as ID's private key.
For the case of Bob, the simulator simply chooses the "r components" of Bob's private key by picking random $k^1_B, k^2_B \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and setting $r^1_B = g^{k^1_B}, r^2_B = g^{k^2_B}$. We observe that in this case $\mathcal{B}$ is not able to compute the corresponding s^1_B, s^2_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key. Moreover the simulator sets $ctr \leftarrow ctr + 2$ and store $\overline{H_1}[B, r^1_B] = h_{ctr-1}, \overline{H_1}[B, r^2_B] = h_{ctr}$.

Simulating sessions First we describe how to simulate sessions different from the test session. Here the main point is that the adversary is allowed to ask session-key queries and thus the simulator must be able to produce the correct session key for each of these sessions. The simulator has full information about all the users' secret keys except Bob. Therefore $\mathcal{B}$ can easily simulate all the protocol sessions that do not include Bob, and answer any of the attacker's queries about these sessions. Hence we concentrate on describing how $\mathcal{B}$ simulates interactions with Bob.
Assume that Bob has a session with Charlie (whose identity is the string C). If Charlie is an uncorrupted party this means that $\mathcal{B}$ will generate the messages on behalf of him. In this case $\mathcal{B}$ knows Charlie's secret key and also has chosen his ephemeral exponents. Thus it is trivial to see that $\mathcal{B}$ has enough information to compute the correct session key. The case when the adversary presents a message $\langle C, r^1_C, r^2_C, u^1_C, u^2_C \rangle$ to Bob as coming from Charlie is more complicated. Here is where $\mathcal{B}$ makes use of the oracle $\mathsf{2DH}(y_1, y_2, \cdot, \cdot, \cdot)$ to answer a session-key query about this session. The simulator replies with a message $\langle B, r^1_B, r^2_B, u^1_B = g^{t^1_B}, u^2_B = g^{t^2_B} \rangle$ where t^1_B and t^2_B are chosen by $\mathcal{B}$. Recall that the session key is

$$K = H(z_{11}, z_{12}, z_{21}, z_{22}, \omega_{11}, \omega_{12}, \omega_{21}, \omega_{22}).$$

Since the simulator knows t^1_B, t^2_B, k^1_B and k^2_B it can check if

$$\overline{H_2}[z_{11}, z_{12}, z_{21}, z_{22}, \omega_{11}, \omega_{12}, \omega_{21}, \omega_{22}] = Z$$

such that all the ω_{ij} have the right form (notice that $\mathcal{B}$ can compute them since it knows t^1_B and t^2_B) and $\mathsf{2DH}(y_1, y_2, u^1_C g^{s^1_C}, \overline{z_{11}}, \overline{z_{12}}) =$

"yes" and $\mathsf{2DH}(y_1, y_2, u_C^2 g^{s_C^2}, \overline{z_{21}}, \overline{z_{22}}) =$ "yes" where $\overline{z_{ij}}$'s are computed as follows:

$$\overline{z_{11}} = \frac{z_{11}}{(u_C^1 g^{s_C^1})^{(k_B^1 + t_B^1) H_1(B, r_B^1)^{-1}}} = g^{(s_C^1 + t_C^1) x_1},$$

$$\overline{z_{12}} = \frac{z_{12}}{(u_C^1 g^{s_C^1})^{(k_B^2 + t_B^2) H_1(B, r_B^2)^{-1}}} = g^{(s_C^1 + t_C^1) x_2},$$

$$\overline{z_{21}} = \frac{z_{21}}{(u_C^2 g^{s_C^2})^{(k_B^1 + t_B^1) H_1(B, r_B^1)^{-1}}} = g^{(s_C^2 + t_C^2) x_1}$$

and $\overline{z_{22}} = \frac{z_{12}}{(u_C^2 g^{s_C^2})^{(k_B^2 + t_B^2) H_1(B, r_B^2)^{-1}}} = g^{(s_C^2 + t_C^2) x_2}$.

If $\mathcal{B}$ finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \xleftarrow{\$} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query to H_2, if the queried tuple z satisfies the equation above it sets $\overline{H_2}[z] = \zeta$ and answers with ζ. This makes oracle's answers consistent.

In addition observe that the simulator can easily answer to $\mathsf{state\ reveal}$ queries as it chooses the fresh exponents on its own.

Simulating the test session Let $\langle B, \rho_B^1, \rho_B^2, u_B^1 = g^{t_B^1}, u_B^2 = g^{t_B^2} \rangle$ be the message from Bob to Alice sent in the test session. We notice that such message may be sent by the adversary who is trying to impersonate Bob. $\mathcal{B}$ responds with the message $\langle A, r_A^1, r_A^2, u_A^1 = V, u_A^2 = V^e, \rangle$ (where $e \xleftarrow{\$} \mathbb{Z}_q$) as coming from Alice. Finally $\mathcal{B}$ provides $\mathcal{A}$ with a random session key.

Run until $\mathcal{A}$ halts and outputs its decision bit

If $\overline{H_1}[B, \rho_B^1] = \perp$ and $\overline{H_1}[B, \rho_B^2] = \perp$ then set $ctr \leftarrow ctr{+}2$ and $\overline{H_1}[B, \rho_B^1] = h_{ctr-1}, \overline{H_1}[B, \rho_B^2] = h_{ctr}$

If $bad = true$ then return $(0, \perp)$

Let $i \in \{1, \ldots, 2Q\}$ such that $H_1(B, \rho_B^1) = h_i$ and $H_1(B, \rho_B^2) = h_{i+1}$

Let $Z = H_2(z_{11}, z_{12}, z_{21}, z_{22}, \omega_{11}, \omega_{12}, \omega_{21}, \omega_{22})$ be the correct session key for the test session.

If $\mathcal{A}$ has success into distinguishing Z from a random value it must necessarily query the correct tuple to the random oracle H_2. This means that $\mathcal{B}$ can efficiently find such tuple in the table $\overline{H_2}$ using the Strong-DH oracle.

Compute $\tau_1 = \frac{z_{11}}{\omega_{11}(u_B^1 \rho_B^1 y_1^{h_i})^{s_A^1}} = (\rho_B^1)^v W_1^{h_i}$ and $\tau_2 = \frac{z_{12}}{\omega_{12}(u_B^2 \rho_B^2 y_2^{h_{i+1}})^{s_A^1}} = (\rho_B^2)^v W_2^{h_{i+1}}$

Return $(i, (\tau_1, \tau_2, h_i, h_{i+1}))$

Let IG be the algorithm that generates a random Diffie-Hellman tuple $(q, \mathbb{G}, g, U_1, U_2, V)$ and $acc_{\mathcal{B}}$ be the accepting probability of $\mathcal{B}$. Then we have that:

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \Pr[bad = true].$$

The probability that $bad = true$ is the probability that the adversary has guessed the "right" r's for a corrupted party ID before corrupting it, in one of the H_1 oracle queries beforehand. Since r is uniformly distributed the probability of guessing it is $1/q$, and since the adversary makes at most $2Q$ queries to H_1 and corrupts at most Q_c parties (and $q > 2^\ell$) we have that

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \frac{Q_c(2Q)}{2^\ell}.$$

which is still non-negligible, since ϵ is non-negligible.

Once we have described the algorithm $\mathcal{B}$ we can now show how to build a solver algorithm S that can exploit $F_{\mathcal{B}}$, the forking algorithm associated with the above $\mathcal{B}$.

The algorithm S plays the role of a CDH solver under the Strong-2DH Assumption. It receives as input a CDH tuple $(q, \mathbb{G}, g, U_1, U_2, V)$ where $U_1 = g^{u_1}, U_2 = g^{u_2}, V = g^v$ and u_1, u_2, v are random exponents in $\mathbb{Z}_q$. S is also given access to a decision oracle $\mathsf{2DH}(U_1, U_2, \cdot, \cdot, \cdot)$ that on input $(\hat{V}, \hat{W}_1, \hat{W}_2)$ answers "yes" if $(U_1, \hat{V}, \hat{W}_1)$ and $(U_2, \hat{V}, \hat{W}_2)$ are a valid DH tuples .

Algorithm $S^{\mathsf{2DH}(U_1,U_2,\cdot,\cdot,\cdot)}(q, \mathbb{G}, g, U_1, U_2, V)$
$(b, \sigma, \sigma') \xleftarrow{\$} F_{\mathcal{B}}^{\mathsf{2DH}(U_1,U_2,\cdot,\cdot,\cdot)}(q, \mathbb{G}, g, U_1, U_2, V)$
If $b = 0$ then return 0 and halt
Parse σ as $(\tau_1, \tau_2, h_1, h_2)$ and σ' as $(\tau_1', \tau_2', h_1', h_2')$
Return $W_1 = (\tau_1/\tau_1')^{(h_1 - h_1')^{-1}}, W_2 = (\tau_2/\tau_2')^{(h_2 - h_2')^{-1}}$

If the forking algorithm $F_{\mathcal{B}}$ has success, this means that there exist random coins γ, an index $J \geq 1$ and $h_1, \ldots, h_{2Q}, h_J', \ldots, h_{2Q}' \in \mathbb{Z}_q$ with $h_1 = h_J \neq h_J' = h_1'$ and $h_2 = h_{J+1} \neq h_{J+1}' = h_2'$ such that: the first execution of $\mathcal{B}(q, \mathbb{G}, g, U_1, V, h_1, \ldots, h_{2Q}; \gamma)$ outputs $\tau_1 = (\rho_B^1)^v W_1^{h_1}$ and $\tau_2 = (\rho_B^2)^v W_2^{h_2}$ where $\overline{H_1}[B, \rho_B^1] = h_1$ and $\overline{H_1}[B, \rho_B^2] = h_2$; the second execution of $\mathcal{B}(q, \mathbb{G}, g, U_1, U_2, V, h_1, \ldots, h_{J-1}, h_J', \ldots, h_{2Q}'; \rho)$ outputs $\tau_1' = (\rho_{B'}^{'1})^v W_1^{h_1'}$ and $\tau_2' = (\rho_{B'}^{'2})^v W_2^{h_2'}$ where $\overline{H_1}[B', \rho_{B'}^{'1}] = h_1'$ and $\overline{H_1}[B', \rho_{B'}^{'2}] = h_2'$. Since the two executions of $\mathcal{B}$ are the same until the response to the J-th query to H_1, then we must have $B = B'$, $\rho_B^1 = \rho_{B'}^{'1}$ and $\rho_B^2 = \rho_{B'}^{'2}$. It is worth noting that responses to $H_1(B, r_B^1)$ and $H_1(B, r_B^2)$ are always answered with consecutive values h_{ctr} and h_{ctr+1} respectively. Thus it is easy to see that S achieves its goal by computing $W_1 = (\tau_1/\tau_1')^{\frac{1}{h_1 - h_1'}} = g^{u_1 v}$ and $W_2 = (\tau_2/\tau_2')^{\frac{1}{h_2 - h_2'}} = g^{u_2 v}$.

Finally, by the General Forking Lemma, we have that if $\mathcal{A}$ has non-negligible advantage into breaking the security of $\mathsf{2IB\text{-}KA}$, then S's success probability is also non-negligible.

5.1 Forward secrecy

The id-based key agreement protocol $\mathsf{2IB\text{-}KA}$ described in the previous section satisfies weak forward secrecy as proven in the following theorem.

Theorem 7. *Under the 2DH Assumption, if we model H_1 and H_2 as random oracles then the protocol* 2IB-KA *has weak forward secrecy.*

Proof. For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that is able to break the weak forward secrecy of the protocol 2IB-KA with non-negligible advantage ϵ. Let n be an upper bound to the number of sessions of the protocol run by $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by the adversary to the random oracles H_1, H_2 respectively. Then we show how to build a simulator S that uses $\mathcal{A}$ to solve the 2DH problem with probability at least ϵ/nQ_2. S receives as input a tuple $(q, \mathbb{G}, g, U_1, U_2, V)$ where $U_1 = g^{u_1}, U_2 = g^{u_2}, V = g^v$ and u_1, u_2, v are random exponents in $\mathbb{Z}_q$. The simulator plays the role of the CDH solver and its goal it to compute the values $W_1 = g^{u_1 v}$ and $W_2 = g^{u_2 v}$.

SETUP. S sets up a simulated execution of the protocol, with simulated KGC, users and sessions. First of all S defines the public parameters of the protocol simulating the KGC. So it chooses random $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$ and sets $y_1 = g^{x_1}, y_2 = g^{x_2}$. Then it provides the adversary with input $(q, \mathbb{G}, g, y_1, y_2)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r) S chooses a random $e_{ID} \xleftarrow{\$} \mathbb{Z}_q$ and sets $H_1(ID, r) = e_{ID}$. Similar work is done for H_2.

Since S knows the master secret key (x_1, x_2), it can simulate the KGC in full, and give secret keys to all the parties in the network, including answering private key queries from the adversary. At the beginning of the game S guesses the test session and its peers Alice and Bob.

SIMULATING PROTOCOL SESSIONS. Sessions different from the test session are easily simulated since S knows all the information needed to compute the session keys and answer any query (including session key and state reveal queries) from the adversary.

SIMULATING THE TEST SESSION. We now show how to simulate the test session in order to extract W_1, W_2 from the adversary. Since in this game the adversary is assumed to be passive during the test session, the parties (i.e. the simulator in this case) choose the messages exchanged in this session.

Let $(A, r_A^1, r_A^2, s_A^1, s_A^2), (B, r_B^1, r_B^2, s_B^1, s_B^2)$ be the identity information and the secret keys of Alice and Bob respectively (S knows these values). The simulator sets Alice's message as $\langle A, r_A^1, r_A^2, u_A^1 = U_1, u_A^2 = U_2 \rangle$ while the one from Bob is $\langle B, r_B^1, r_B^2, u_B^1 = V, u_B^2 = V^d \rangle$ (for random $d \xleftarrow{\$} \mathbb{Z}_q$). S is implicitly setting $t_A^1 = u_1, t_A^2 = u_2, t_B^1 = v, t_B^2 = vd$. In this case the correct session key contains $\omega_{11} = g^{u_1 v}, \omega_{21} = g^{u_2 v}$. Since H_2 is modeled as a random oracle, if $\mathcal{A}$ has success into distinguishing Z from a random value, it must have queried H_2 on the correct input. Thus S can choose a random value among all the queries that it received from the adversary. Since the number of queries Q_2 is polynomially bounded, S can find the correct $\omega_{11} = W_1, \omega_{21} = W_2$ with non-negligible probability ϵ/nQ_2. This completes the proof of this case[7].

[7] We could give the simulator access to the Strong-2DH oracle 2DH, and then S could use it to "test" all queries to H_2 to find the correct W_1, W_2. The reduction would be

6 Comparisons with other IB-KA Protocols

In this section we compare IB-KA with other id-based KA protocols from the literature. In particular, we consider the protocol by Chen and Kudla [13] (SCK-2) (which is a modification of Smart's [36]) and two protocols proposed very recently by Boyd *et al.* [7] (BCNP1, BCNP2).

For our efficiency comparisons we consider a security parameter of 128 and implementations of SCK-2, BCNP1 and BCNP2 with Type 3 pairings [8], which are the most efficient pairings for this kind of security level (higher than 80). Our protocol is assumed to be implemented in an elliptic curves group $\mathbb{G}$ with the same security parameter. In this scenario elements of $\mathbb{G}$ and $\mathbb{G}_1$ need 256 bit to be represented, while 512 bits are needed for $\mathbb{G}_2$ elements and 3072 bits for an element of $\mathbb{G}_T$.

We estimate the computational cost of all the protocols using the costs per operation for Type 3 pairings given by Chen *et al.* in [12]. The bandwidth cost is expressed as the amount of data in bits sent by each party to complete a session of the protocol[9].

According to the work of Chen *et al.* [12] SCK-2 is the most efficient protocol with a proof of security in the CK model for all types of pairings. It is proved secure using random oracles under the Bilinear Diffie-Hellman Assumption and requires one round of communication with only one group element sent by each party. To be precise, we point out that the protocol of Boyd *et al.* (BMP) [8] would appear computationally more efficient than SCK-2, but unfortunately it works only in type 1 and type 4 pairings and is proven secure only in symmetric pairings. BCNP1 and BCNP2 are generic constructions based on any CCA-secure IB-KEM. When implemented (as suggested by the authors of [7]) using one of the IB-KEMs by Kiltz [24], Kiltz-Galindo [25] or Gentry [21] they lead to a two-pass single-round protocol with (CK) security in the standard model. BCNP2 provides weak FS and resistance to KCI attacks, while BCNP1 satisfies only the former property.

The results are summarized in Table 1 assuming protocols BCNP1 and BCNP2 to be implemented with Kiltz's IB-KEM (the most efficient for this application according to the work of Boyd *et al.* [7]). We defer to the original papers of SCK-2 [13] and BCNP1, BCNP2 [7] for more details about these costs. As described in the table, our protocol has a reasonable bandwidth requirement and achieves the best computational efficiency among the other id-based KA protocols.

Comparison with PKI-based protocols. We also compare our protocol to MQV [27], and its provably secure version HMQV [26], which is the most

tighter (removing the factor of Q_2^{-1} from the success probability) but would require the Strong-2DH Assumption also in this case.

[8] This classification of pairing groups into several types is provided by Galbraith *et al.* in [19].

[9] We do not consider the identity string sent with the messages as it can be implicit and, in any way, appears in all the protocols.

	weak FS	KCI	Standard model	Efficiency	
				Bandwidth	Cost per party
BCNP1	✗	✓	✓	768	56
BCNP2	✓	✓	✓	1024	59
SCK-2	✓	✓	✗	256	43
IB-KA	✓	✓	✗	512	6

Table 1. Comparisons between IB-KA protocols.

efficient protocol in the public-key setting. When comparing our protocol to a PKI-based scheme, like MQV, we must also consider the additional cost of sending and verifying certificates.

We measure the computation costs of the protocols in terms of the number of exponentiations in the underlying group needed to compute the session key. If the exponentiations is done with an exponent that is half the length of the group size, then obviously we count it as $1/2$ exponentiation. Also if an exponentiation is done over a fixed basis, we apply precomputation schemes to speed up the computation, e.g. [20].

Our protocol requires each party to send a single message consisting of two group elements. To compute the session key, the parties perform 2 full exponentiations over variable basis, and one half exponentiation over a fixed basis[10]. For our security parameter, following [20], the latter half exponentiation can be computed with less than 20 group multiplications, with a precomputation table of moderate size.

In MQV, each party sends a single message consisiting of one group element, and performs 1.5 exponentiations to compute the session key. Moreover, in HMQV certificates are sent and verified. Here we distinguish two cases: the certificate is based either on an RSA signature, or on a discrete-log signature, e.g. Schnorr's.

In the RSA case, a short exponent e.g. $e = 2^{16} + 1$, is typically used, and the verification cost is basically equivalent to the cost of the half exponentiation with precomputation in our protocol above. Therefore in this case, MQV is faster, but by a mere half exponentiation. The price to pay however is a massive increase in bandwidth to send the RSA signature (i.e. 3072 bits), and the introduction of the RSA Assumption in order to prove security of the entire scheme.

If we use a Schnorr signature for the certificate, then MQV require sending two more group elements, and therefore its bandwidth requirement is already worse than our protocol (by one group element). The parties then must compute one full and one half exponentiation, both with fixed basis[11] to verify the certificate. This extra computational cost can be compared to an additional half exponentiation, making the computation requirement of MQV with Schnorr certificates equivalent to that of our protocol.

In conclusion, when comparing our protocol with MQV with certificates we find that our protocol: (i) has comparable computational cost; (ii) has better

[10] Indeed since the input to the hash function H_1 is randomized, we can set its output length to be half of the length of the group size.

[11] Though different basis, which means that in order to apply precomputation techniques, the parties need to maintain two tables.

bandwidth (by far in the case of RSA certificates) and (iii) simplifies protocol implementation by removing entirely the need to manage certificates and to interact with a PKI[12].

7 Security analysis of related protocols

As an additional contribution of the paper, in this section we present a formal security analysis of two id-based KA protocols that use techniques that inspired our work: the first by Gunther [22] and the second by Saeednia [32] (which is is an improvement of the previous one). In particular we show variants of these protocols that allow to prove their security in the CK model while only an intuition of security was stated in the original works [22, 32].

7.1 Gunther's protocol

We present a slightly different variant of Gunther's protocol [22] which we prove secure under the Gap-DH and KEA assumptions.

The Knowledge of Exponent Assumption (KEA) was first stated by Damgård in [15] and later discussed in [3, 23]. Let $\mathbb{G}$ be a group of prime order q with generator g. Then we say that KEA holds over $\mathbb{G}$ if: for any efficient algorithm $\mathcal{A}$ that on input (g, g^a) outputs a pair (B, C) such that $C = B^a$ there exists an efficient "extractor" algorithm $\mathcal{A}'$ that given the same input of $\mathcal{A}$ outputs (B, C, b) such that $C = B^a$ and $B = g^b$.

The modified protocol is summarized in Figure 2. We recall that the session key in the original protocol was just $z_1 z_2 z_3$ and the key generation process computed the hash only on the identity string $H(ID)$. So what we change is: to hash the session key and include the value r_{ID} when hashing the identity. Since the key derivation process is essentially an El Gamal signature on the identity string, the latter modification follows what Pointcheval and Stern proposed in [31] to prove the security of the El Gamal signature scheme.

The following theorem proves the security of the protocol.

Theorem 8. *If H_1 and H_2 are modeled as random oracles and the Gap-DH and KEA assumptions hold, then Gunther's protocol is a secure identity-based key agreement protocol.*

Proof. For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that has non-negligible advantage ϵ into breaking Gunther's protocol, then we show how to build a solver algorithm S for the CDH problem.

In our reduction we will proceed into two steps. First, we describe an intermediate algorithm $\mathcal{B}$ (i.e. the simulator) that interacts with the protocol adversary $\mathcal{A}$ and returns a side output σ. Second, we will show how to build an algorithm

[12] In the above, we did not account for the cost of verifying group membership for the elements sent by the parties, which is necessary both in the case of MQV and our protocol, and is the same in both protocols.

Gunther's protocol

Setting: A Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q together with a random generator $g \in \mathbb{G}$ and an exponent $x \xleftarrow{\$} \mathbb{Z}_q$. KGC publishes $\mathbb{G}, q, g, y = g^x$ and two hash functions H_1, H_2. It distributes to each user with identity U a private key (r_U, s_U) computed as follows: $r_U = g^k, s_U = k^{-1}(H_1(U, r_U) - xr_U) \bmod q$ for random $k \xleftarrow{\$} \mathbb{Z}_q$.

Key agreement: A and B choose ephemeral private exponents t_A, w_A and t_B, w_B, respectively.

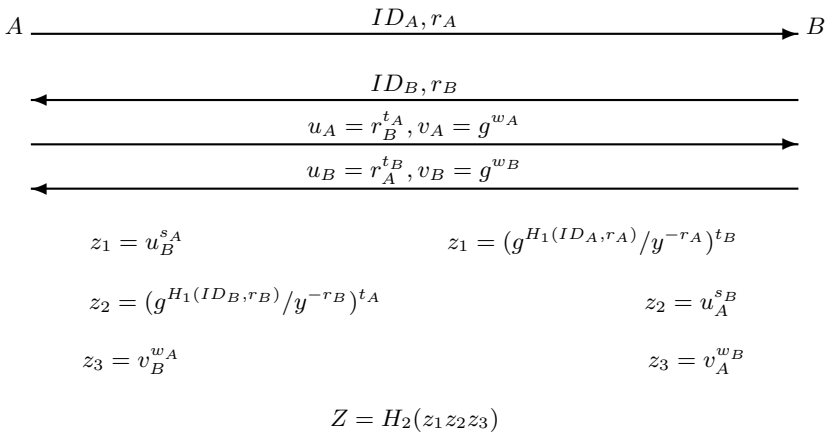

Fig. 2. A and B share session key Z.

S that exploits $F_{\mathcal{B}}$, the forking algorithm associated with $\mathcal{B}$, to solve the CDH problem under the Gap-DH Assumption.

$\mathcal{B}$ receives as input a tuple $(\mathbb{G}, g, U, V)$, where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$, and a set of random elements $h_1, \ldots, h_Q \in \mathbb{Z}_q$. The simulator is also given access to a DH oracle $\mathsf{DH}(\cdot, \cdot, \cdot)$ that on input (U, V, W) answers "yes" if (U, V, W) is a valid DDH tuple . The side output of $\mathcal{B}$ is $\sigma \in \mathbb{G}^2 \times \mathbb{Z}_q$ or $\perp$. Let n be an upper bound to the number of sessions of the protocol run by the adversary $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by $\mathcal{A}$ to the random oracles H_1, H_2 respectively. Moreover, let Q_c be the number of users corrupted by $\mathcal{A}$ and $Q = Q_1 + Q_c + 1$.

Algorithm $\mathcal{B}^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V, h_1, \ldots, h_Q)$
Initialize $ctr \leftarrow 0; bad \leftarrow false$; empty tables $\overline{H_1}, \overline{H_2}$;
Run $\mathcal{A}$ on input $(\mathbb{G}, g, y = U)$ as the public parameters of the protocol and simulates the protocol's environment for $\mathcal{A}$ as follows:
Guess the test session by choosing at random the user (let us call him Bob) and the order number of the test session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

H_2 **queries** On input a value z:
If $\overline{H_2}[z] = \perp$: choose a random string $Z \in \{0,1\}^\ell$ and store $\overline{H_2}[z] = Z$
Return $\overline{H_2}[z]$ to $\mathcal{A}$

H_1 **queries** On input (ID, r):
If $\overline{H_1}[ID, r] = \perp$, then $ctr \leftarrow ctr + 1; \overline{H_1}[ID, r] = h_{ctr}$
Return $\overline{H_1}[ID, r]$ to $\mathcal{A}$

Party Corruption When $\mathcal{A}$ asks to corrupt party $ID \neq B$, then:
$ctr \leftarrow ctr + 1; d \xleftarrow{\$} \mathbb{Z}_q; r = g^{h_{ctr}} y^d; s = -rd^{-1}$
If $\overline{H_1}[ID, r] \neq \perp$ then $bad \leftarrow true$
Store $\overline{H_1}[ID, r] = h_{ctr} s$ and return (r, s) as ID's private key.
For the case of Bob, the simulator simply chooses the r_B component of Bob's private key by picking a random $k_B \xleftarrow{\$} \mathbb{Z}_q$ and setting $r_B = g^{k_B}$ and $\overline{H_1}[B, r_B] = h_{ctr}$. We observe that in this case $\mathcal{B}$ is not able to compute the corresponding s_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key.

Simulating sessions First we describe how to simulate sessions different from the test session. Here the main point is that the adversary is allowed to ask session-key queries and thus the simulator must be able to produce the correct session key for each of these sessions. The simulator has full information about all the users' secret keys except Bob. Therefore $\mathcal{B}$ can easily simulate all the protocol sessions that do not include Bob, and answer any of the attacker's queries about these sessions. Hence we concentrate on describing how $\mathcal{B}$ simulates interactions with Bob.

Assume that Bob has a session with Charlie (whose identity is the string C). If Charlie is an uncorrupted party this means that $\mathcal{B}$ will generate the messages on behalf of him. In this case $\mathcal{B}$ knows Charlie's secret key and also has chosen his ephemeral exponents t_C, w_C. Thus it is trivial to see that $\mathcal{B}$ has enough information to compute the correct session key. The case when the adversary presents messages $\langle C, r_C \rangle, \langle u_C, v_C \rangle$ to Bob as coming from Charlie is more complicated. Here is where $\mathcal{B}$ makes use of the oracle $\mathsf{DH}(\cdot,\cdot,\cdot)$ to answer a session-key query about this session. The simulator replies with messages $\langle B, r_B \rangle, \langle u_B = g^{t_B}, v_B = g^{w_B} \rangle$ where t_B and w_B are chosen by $\mathcal{B}$. Recall that the session key is $H_2(z_1 z_2 z_3)$ with $z_1 = u_B^{s_C}$, $z_2 = u_C^{s_B}$ and $z_3 = g^{w_B w_C}$. Notice that z_2 is the Diffie-Hellman result of the values $u_C = r_B^{t_C}$ and $r_B^{s_B}$. Since the simulator can compute $r_B^{s_B} = g^{H_1(B, r_B) - x r_B}$, it can check if $\overline{H_2}[z] = Z$ where $\mathsf{DH}(r_B^{s_B}, u_C, \bar{z}) =$ "yes" and $\bar{z} = z / z_1 z_3$. If $\mathcal{B}$ finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \xleftarrow{\$} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query z to H_2, if z satisfies the equation above it sets $\overline{H_2}[z] = \zeta$ and answers with ζ. This makes oracle's answers consistent.

In addition observe that the simulator can easily answer to state reveal queries as it chooses the fresh exponents on its own.

Simulating the test session Let $\langle B, \rho_B \rangle, \langle u_B = r_A^{t_B}, v_B = g^{w_B} \rangle$ be the messages from Bob to Alice sent in the test session. We notice that such message may be sent by the adversary who is trying to impersonate Bob. In this case $\mathcal{A}$ may use a value $\rho_B = g^{\lambda_B}$ of its choice as the public component of Bob's private key (i.e. different than $r_B = g^{k_B}$ which $\mathcal{B}$ simulated and for which it knows k_B). $\mathcal{B}$ responds with the messages $\langle A, r_A \rangle, \langle u_A = V, v_A = g^{w_A} \rangle$ as coming from Alice. Finally $\mathcal{B}$ provides $\mathcal{A}$ with a random session key.

Run until $\mathcal{A}$ halts and outputs its decision bit
If $\overline{H_1}[B, \rho_B] = \perp$ then set $ctr \leftarrow ctr + 1$ and $\overline{H_1}[B, \rho_B] = h_{ctr}$
If $bad = true$ then return $(0, \perp)$
Let $i \in \{1, \ldots, Q\}$ such that $H_1(B, \rho_B) = h_i$
Let $Z = H_2(z_1 z_2 z_3)$ be the correct session key for the test session where $z_1 = u_B^{s_A}, z_2 = \rho_B^{(h_i - x\rho_B)t_A}$ and $z_3 = v_B^{w_A}$.
If $\mathcal{A}$ has success into distinguishing Z from a random value it must necessarily query the correct value $z = z_1 z_2 z_3$ to the random oracle H_2. This means that $\mathcal{B}$ can efficiently find z in the table $\overline{H_2}$ using the Gap-DH oracle.
Compute $\tau = \frac{z}{z_1 z_3} = V^{h_i/\lambda_B} W^{-\rho_B/\lambda_B}$
Return $(i, (\tau, h_i, \rho_B))$

Let IG be the algorithm that generates a random Diffie-Hellman tuple $(\mathbb{G}, g, U, V)$ and $acc_{\mathcal{B}}$ be the accepting probability of $\mathcal{B}$. Then we have that:

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \Pr[bad = true].$$

For the same argument given in Section 4 we have that

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \frac{Q_c(Q)}{2^\ell}.$$

which is still non-negligible, since ϵ is non-negligible.

Once we have described the algorithm $\mathcal{B}$ we show how to build an algorithm S' that exploits $F_{\mathcal{B}}$, the forking algorithm associated with the above $\mathcal{B}$. Then we will show another algorithm S that solves CDH under the Gap-DH Assumption.

Algorithm $S'^{\mathsf{DH}(\cdot,\cdot,\cdot)}(\mathbb{G}, g, U, V)$
$(b, \sigma, \sigma') \stackrel{\$}{\leftarrow} F_{\mathcal{B}}^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V)$
If $b = 0$ then return 0 and halt
Parse σ as (τ, h, ρ) and σ' as (τ', h', ρ')
Compute $V' = V^{\lambda_B^{-1}} = (\tau/\tau')^{(h-h')^{-1}}$ and output (V', ρ, τ, h).

If the forking algorithm $F_{\mathcal{B}}$ has success, this means that there exist random coins γ, an index $J \geq 1$ and $h_1, \ldots, h_Q, h'_J, \ldots, h'_Q \in \mathbb{Z}_q$ with $h = h_J \neq h'_J = h'$ such that: the first execution of $\mathcal{B}(\mathbb{G}, g, U, V, h_1, \ldots, h_Q; \gamma)$ outputs

$\tau = g^{vh/\lambda} W^{-\rho/\lambda}$ where $\overline{H_1}[B, \rho] = h$; the second execution of $\mathcal{B}(\mathbb{G}, g, U, V, h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_Q; \gamma)$ outputs $\tau' = g^{vh'/\lambda'} W^{-\rho'/\lambda'}$ such that $\overline{H_1}[B', \rho'] = h'$. Since the two executions of $\mathcal{B}$ are the same until the response to the J-th query to H_1, then we must have $B = B'$ and $\rho = \rho'$ (and $\lambda = \lambda'$). In other words we have an algorithm that given in input a pair (g, g^v) is returning in output $(V', \rho = g^{\lambda})$ such that $V' = g^{v/\lambda}$. If the KEA assumption holds then there exists an extractor algorithm that given the same input (g, g^v) outputs (V', ρ, λ). Therefore we can run such algorithm to get λ. We can define S as the algorithm that runs the corresponding extractor algorithm of S' on its same input and gets $(V', \rho, \lambda, \tau, h)$. Finally S can compute

$$W = (\frac{\tau}{V^{\lambda/h}})^{-\rho^{-1}} = g^{uv}.$$

By the General Forking Lemma, we have that if $\mathcal{A}$ has non-negligible advantage into breaking the security of Gunther's protocol, then the probability that S has success is also non-negligible.

Vulnerability to reflection attack In this section we show that Gunther's protocol is vulnerable to the reflection attack. We recall that this attack occurs when an adversary tries to impersonate a party, e.g. Bob to Bob himself. In the case of Gunther's protocol we can restrict this attack to the case when an adversary presents to Bob the first message containing Bob's identity B and the key r_B. In particular, we do not consider the case in which the adversary uses a value $r'_B \neq r_B$ because one can imagine that the honest Bob (who knows his secret key r_B) refuses the connections from himself with $r'_B \neq r_B$.

In this scenario, when (the honest) Bob generates $u_B = g^{t_B}, v_B = g^{w_B}$ and the adversary sends $u'_B = g^{t'_B}, v'_B = g^{w'_B}$ the session key will be $(r_B^{s_B})^{t_B + t'_B} g^{w_B w'_B}$. Thus an adversary, after seeing the message from Bob, can set $u'_B = g^t / u_B$ and $v'_B = g^{w'_B}$ and then is able to compute the session key $H(\bar{z})$ where $\bar{z} = (r_B^{s_B})^t \cdot v_B^{w'_B} = (g^{H(B, r_B)} y^{-r_B})^t \cdot v_B^{w'_B}$.

Other security properties of Gunther's protocol Following an argument similar to that used for protocol IB-KA in Section 4.1, it is possible to show that Gunther's protocol is resistant to KCI attacks. Moreover we prove the following theorem to show that it satisfies weak forward secrecy.

Theorem 9. *Let $\mathcal{A}$ be a PPT adversary that is able to break the weak forward secrecy of Gunther's protocol with advantage ϵ. Let n be the an upper bound to the number of sessions of the protocol run by $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by the adversary to the random oracles H_1, H_2 respectively. Then we can solve the CDH problem with probability at least $\epsilon/(nQ_2)$.*

In the following we show how to build an efficient algorithm S that can solve the CDH problem.

S receives as input a tuple $(\mathbb{G}, q, g, U = g^u, V = g^v)$ and wants to compute $W = g^{uv}$. First S simulates the KGC setting up the public parameters of the protocol. It chooses a random $x \xleftarrow{\$} \mathbb{Z}_q$ and sets $y = g^x$. Then it provides the adversary with input $(\mathbb{G}, q, g, y)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r_{ID}) S chooses a random $e_{ID} \xleftarrow{\$} \mathbb{Z}_q$ and sets $H_1(ID, r_{ID}) = e_{ID}$. Similar work is done for H_2.

The adversary is allowed to ask the KGC for the secret keys of users of its choice and thus S must be able to simulate the key derivation process. As one can notice, when the adversary asks for the secret key of a user, the simulator is always able to respond, since it has chosen the master secret key x by itself.

At the beginning of the game S guesses the test session and its holder (let us call him Bob). Also let Alice be the other party of the session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

Without loss of generality we assume that the test session is at Bob (thus the corresponding matching session is at Alice). Since we are in the case when the adversary is passive during the execution of the protocol, the simulator chooses the messages exchanged in the test session.

Let $(A, r_A, s_A), (B, r_B, s_B)$ be the identity informations and the secret keys of Alice and Bob respectively. The simulator uses these values to create the first two messages between the parties. To generate the other ones S chooses random $t_A, t_B \xleftarrow{\$} \mathbb{Z}_q$ and sets $\langle u_A = r_B^{t_A}, v_A = U \rangle$ and $\langle u_B = r_A^{t_B}, v_B = V \rangle$. Thus S is implicitly setting $w_A = u, w_B = v$. Since H_2 is modeled as a random oracle, if the adversary has success into distinguishing the real session key from a random value, it must have queried H_2 on the correct input $\bar{z} = u_B^{s_A} u_A^{s_B} g^{uv}$. Thus S can choose a random value among all the queries that it received from the adversary and then extract $W = \bar{z}/(u_B^{s_A} u_A^{s_B})$ from it. In conclusion S can find W with non-negligible probability ϵ/nQ_2. This completes the proof of this case.

Remark 1. If we would assume the simulator having access to a Gap-DH oracle, S might use the oracle to test, for all queries z made by the adversary, if $DH(U, V, z_3) =$ "yes" (where z_3 is computed as $z/z_1 z_2$) and then output z_3 for which the test is satisfied. In this case the security of Gunther's protocol would reduce to the Gap-DH Assumption instead of CDH, but we would not have the Q_2 loss factor.

7.2 Saeednia's protocol

Saeednia proposed in [32] a variant of Gunther's protocol that allows to reduce to 2 the number of messages exchanged by the parties. The idea of Saeednia was basically to use a different equation for computing the El Gamal signature to generate users' keys. Here we propose a variant of Saeednia's protocol that can be proved secure in the CK model under the Gap-DH assumption. The modified protocol is summarized in Figure 3.

Saeednia's protocol

Setting: A Key Generation Center (KGC) chooses a group $\mathbb{G}$ of prime order q together with a random generator $g \in \mathbb{G}$ and an exponent $x \xleftarrow{\$} \mathbb{Z}_q$. KGC publishes $\mathbb{G}, q, g, y = g^x$ and two hash functions H_1, H_2. It distributes to each user with identity U a private key (r_U, s_U) computed as follows: $r_U = g^k, s_U = kH_1(U, r_U) + xr_U \bmod q$ for random $k \xleftarrow{\$} \mathbb{Z}_q$.

Key agreement: A and B choose ephemeral private exponents t_A and t_B, respectively.

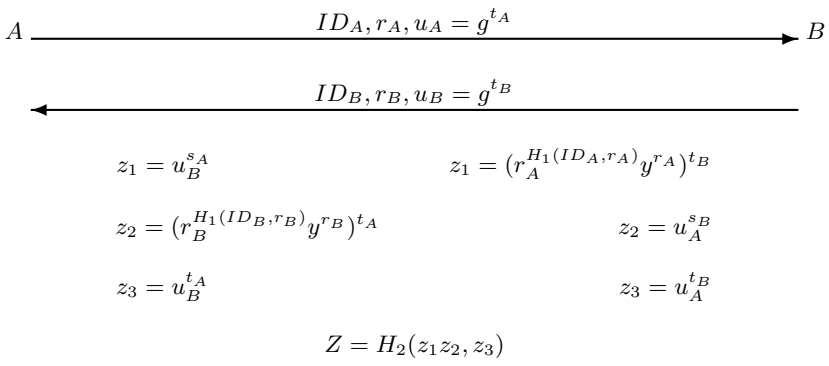

Fig. 3. A and B share session key Z.

We did almost the same modifications proposed for Gunther's protocol in Section 7.1, namely adding the value r when hashing the identity and hashing the session key. We recall that the session key in the original version of the protocol is the value $z_1 z_2 z_3$ where z_3 is needed to obtain (weak) FS. In our variant we include z_3 in the hash of the session key as $H_2(z_1 z_2, z_3)$.

The following theorem proves the security of the modified Saeednia's protocol.

Theorem 10. *If H_1 and H_2 are modeled as random oracles and the Gap-DH assumption holds, then Saeednia's protocol is a secure identity-based key agreement protocol.*

Proof. For sake of contradiction let us suppose there exists a PPT adversary $\mathcal{A}$ that has non-negligible advantage ϵ into breaking Saeednia's protocol, then we show how to build a solver algorithm S for the CDH problem.

In our reduction we will proceed into two steps. First, we describe an intermediate algorithm $\mathcal{B}$ (i.e. the simulator) that interacts with the protocol adversary $\mathcal{A}$ and returns a side output σ. Second, we will show how to build an algorithm S that exploits $F_{\mathcal{B}}$, the forking algorithm associated with $\mathcal{B}$, to solve the CDH problem under the Gap-DH Assumption.

$\mathcal{B}$ receives as input a tuple $(\mathbb{G}, g, U, V)$, where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$, and a set of random elements $h_1, \ldots, h_Q \in \mathbb{Z}_q$. The

simulator is also given access to a DH oracle $\mathsf{DH}(\cdot,\cdot,\cdot)$ that on input (U,V,W) answers "yes" if (U,V,W) is a valid DDH tuple . The side output of $\mathcal{B}$ is $\sigma \in \mathbb{G}^2 \times \mathbb{Z}_q$ or $\perp$. Let n be an upper bound to the number of sessions of the protocol run by the adversary $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by $\mathcal{A}$ to the random oracles H_1, H_2 respectively. Moreover, let Q_c be the number of users corrupted by $\mathcal{A}$ and $Q = Q_1 + Q_c + 1$.

Algorithm $\mathcal{B}^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V, h_1, \ldots, h_Q)$

Initialize $ctr \leftarrow 0; bad \leftarrow false$; empty tables $\overline{H_1}, \overline{H_2}$;

Run $\mathcal{A}$ on input $(\mathbb{G}, g, y = U)$ as the public parameters of the protocol and simulates the protocol's environment for $\mathcal{A}$ as follows:

Guess the test session by choosing at random the user (let us call him Bob) and the order number of the test session. If n is an upper bound to the number of all the sessions initiated by $\mathcal{A}$ then the guess is right with probability at least $1/n$.

H_2 **queries** On input a pair (z, z_3):

If $\overline{H_2}[z, z_3] = \perp$: choose a random string $Z \in \{0,1\}^\ell$ and store $\overline{H_2}[z, z_3] = Z$

Return $\overline{H_2}[z, z_3]$ to $\mathcal{A}$

H_1 **queries** On input (ID, r):

If $\overline{H_1}[ID, r] = \perp$, then $ctr \leftarrow ctr + 1; \overline{H_1}[ID, r] = h_{ctr}$

Return $\overline{H_1}[ID, r]$ to $\mathcal{A}$

Party Corruption When $\mathcal{A}$ asks to corrupt party $ID \neq B$, then:

$ctr \leftarrow ctr + 1; e \stackrel{\$}{\leftarrow} \mathbb{Z}_q; r = g^e y^{h_{ctr}}; s = -erd^{-1}$

If $\overline{H_1}[ID, r] \neq \perp$ then $bad \leftarrow true$

Store $\overline{H_1}[ID, r] = -rh_{ctr}^{-1}$ and return (r, s) as ID's private key.

For the case of Bob, the simulator simply chooses the r_B component of Bob's private key by picking a random $k_B \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and setting $r_B = g^{k_B}$. Moreover it sets $\overline{H_1}[B, r_B] = h_{ctr}$. We observe that in this case $\mathcal{B}$ is not able to compute the corresponding s_B. However, since Bob is the user guessed for the test session, we can assume that the adversary will not ask for his secret key.

Simulating sessions First we describe how to simulate sessions different from the test session. Here the main point is that the adversary is allowed to ask session-key queries and thus the simulator must be able to produce the correct session key for each of these sessions. The simulator has full information about all the users' secret keys except Bob. Therefore $\mathcal{B}$ can easily simulate all the protocol sessions that do not include Bob, and answer any of the attacker's queries about these sessions. Hence we concentrate on describing how $\mathcal{B}$ simulates interactions with Bob.

Assume that Bob has a session with Charlie (whose identity is the string C). If Charlie is an uncorrupted party this means that $\mathcal{B}$ will generate the messages on behalf of him. In this case $\mathcal{B}$ knows Charlie's secret key and also has chosen his ephemeral exponent t_C. Thus it is trivial to see that $\mathcal{B}$ has enough information to compute the

correct session key. The case when the adversary present a message $\langle C, r_C, u_C\rangle$ to Bob as coming from Charlie is more complicated. Here is where $\mathcal{B}$ makes use of the oracle $\mathsf{DH}(\cdot,\cdot,\cdot)$ to answer a session-key query about this session. The simulator replies with the message $\langle B, r_B, u_B = g^{t_B}\rangle$ where t_B is chosen by $\mathcal{B}$. Recall that the session key is $H_2(z_1 z_2, z_3)$ with $z_1 = u_B^{s_C}$, $z_2 = u_C^{s_B}$ and $z_3 = g^{u_B u_C}$. Notice that z_2 is the Diffie-Hellman result of the values u_C and g^{s_B}, where $g^{s_B} = r_B^{H_1(B,r_B)} y^{r_B}$. Since the simulator can compute z_1 and $z_3 = u_C^{t_B}$, it can check if $\overline{H_2}[z, z_3] = Z$ where $\mathsf{DH}(g^{s_B}, u_C, \bar{z}) =$ "yes" and $\bar{z} = z/z_1$. If $\mathcal{B}$ finds a match then it outputs the corresponding Z as session key for Bob. Otherwise it generates a random $\zeta \xleftarrow{\$} \{0,1\}^\ell$ and gives it as response to the adversary. Later, for each query (z, z_3) to H_2, if z satisfies the equation above it sets $\overline{H_2}[z, z_3] = \zeta$ and answers with ζ. This makes oracle's answers consistent.
In addition observe that the simulator can easily answer to state reveal queries as it chooses the fresh exponents on its own.

Simulating the test session Let $\langle B, \rho_B, u_B = g^{t_B}\rangle$ be the message from Bob to Alice sent in the test session. We notice that such message may be sent by the adversary who is trying to impersonate Bob. In this case $\mathcal{A}$ may use a value $\rho_B = g^{\lambda_B}$ of its choice as the public component of Bob's private key (i.e. different than $r_B = g^{k_B}$ which $\mathcal{B}$ simulated and for which it knows k_B). $\mathcal{B}$ responds with the message $\langle A, r_A, u_A = V\rangle$ as coming from Alice. Finally $\mathcal{B}$ provides $\mathcal{A}$ with a random session key.

Run until $\mathcal{A}$ halts and outputs its decision bit
If $\overline{H_1}[B, \rho_B] = \perp$ then set $ctr \leftarrow ctr + 1$ and $\overline{H_1}[B, \rho_B] = h_{ctr}$
If $bad = true$ then return $(0, \perp)$
Let $i \in \{1, \ldots, Q\}$ such that $H_1(B, \rho_B) = h_i$
Let $Z = H_2(z_1 z_2, z_3)$ be the correct session key for the test session where $z_1 = u_B^{s_A}, z_2 = g^{(\lambda_B h_i + x\rho_B)t_A}$ and $z_3 = u_B^{t_A}$.
If $\mathcal{A}$ has success into distinguishing Z from a random value it must necessarily query the correct value $(z_1 z_2, z_3)$ to the random oracle H_2. This means that $\mathcal{B}$ can efficiently find such a pair in the table $\overline{H_2}$ using the Gap-DH oracle.
Compute $\tau = \frac{z}{z_1} = V^{\lambda_B h_i} W^{\rho_B}$
Return $(i, (\tau, h_i, \rho_B))$

Let IG be the algorithm that generates a random Diffie-Hellman tuple $(\mathbb{G}, g, U, V)$ and $acc_{\mathcal{B}}$ be the accepting probability of $\mathcal{B}$. Then we have that:

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \Pr[bad = true].$$

For the same argument of Section 4 we have that

$$acc_{\mathcal{B}} \geq \frac{\epsilon}{n} - \frac{Q_c(Q)}{2^\ell}$$

which is still non-negligible, since ϵ is non-negligible.

Once we have described the algorithm $\mathcal{B}$ we can now show how to build a solver algorithm S that can exploit $F_{\mathcal{B}}$, the forking algorithm associated with the above $\mathcal{B}$.

The algorithm S plays the role of a CDH solver under the Gap-DH Assumption. It receives in input a CDH tuple $(\mathbb{G}, g, U, V)$ where $U = g^u, V = g^v$ and u, v are random exponents in $\mathbb{Z}_q$. S is also given access to a decision oracle $\mathsf{DH}(\cdot,\cdot,\cdot)$ that on input (U, V, W) answers "yes" if (U, V, W) is a valid DH tuple .

Algorithm $S^{\mathsf{DH}(\cdot,\cdot,\cdot)}(q, \mathbb{G}, g, U, V)$
$(b, \sigma, \sigma') \xleftarrow{\$} F_{\mathcal{B}}^{\mathsf{DH}(U,\cdot,\cdot)}(\mathbb{G}, g, U, V)$
If $b = 0$ then return 0 and halt
Parse σ as (τ, h, ρ) and σ' as (τ', h', ρ')
Compute $\omega = (\tau/\tau')^{(h-h')^{-1}}$ and output $W = (\frac{\tau}{\omega^h})^{\rho^{-1}}$.

If the forking algorithm $F_{\mathcal{B}}$ has success, this means that there exist random coins γ, an index $J \geq 1$ and $h_1, \ldots, h_Q, h'_J, \ldots, h'_Q \in \mathbb{Z}_q$ with $h = h_J \neq h'_J = h'$ such that: the first execution of $\mathcal{B}(\mathbb{G}, g, U, V, h_1, \ldots, h_Q; \gamma)$ outputs $\tau = V^{h\lambda} W^{\rho}$ where $\overline{H_1}[B, \rho] = h$; the second execution of $\mathcal{B}(\mathbb{G}, g, U, V, h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_Q; \gamma)$ outputs $\tau' = V^{h'\lambda'} W^{\rho'}$ where $\overline{H_1}[B', \rho'] = h'$. Since the two executions of $\mathcal{B}$ are the same until the response to the J-th query to H_1, then we must have $B = B'$ and $\rho = \rho'$ (and $\lambda = \lambda'$). Therefore it is easy to see that S compute $W = g^{uv}$.

By the General Forking Lemma, we have that if $\mathcal{A}$ has non-negligible advantage into breaking the security of Saeednia's protocol, then the probability that S has success is also non-negligible.

Other security properties of Saeednia's protocol Saeednia's protocol with the modifications presented above satisfies resistance to KCI and reflection attacks. To see this, it is possible to observe that the same arguments given in Section 4.1 for the IB-KA protocol apply to this case. In particular, resistance to reflection attacks can be proven under the Square-DH assumption as well, namely we can build an algorithm that computes g^{u^2} when given in input g, g^u.

Moreover we can prove the following theorem to show that the protocol has weak forward secrecy.

Theorem 11. *Let $\mathcal{A}$ be a PPT adversary that is able to break the weak forward secrecy of Saeednia's protocol with advantage ϵ. Let n be the an upper bound to the number of sessions of the protocol run by $\mathcal{A}$ and Q_1 and Q_2 be the number of queries made by the adversary to the random oracles H_1, H_2 respectively. Then we can solve the CDH problem with probability at least $\epsilon/(nQ_2)$.*

In the following we show how to build an efficient algorithm S that can solve the CDH problem.

S receives as input a tuple $(\mathbb{G}, q, g, U = g^u, V = g^v)$ and wants to compute $W = g^{uv}$. First S simulates the KGC setting up the public parameters of the

protocol. It chooses a random $x \xleftarrow{\$} \mathbb{Z}_q$ and sets $y = g^x$. Then it provides the adversary with input $(\mathbb{G}, q, g, y)$ and oracle access to H_1 and H_2. Since H_1 and H_2 are modeled as random oracles, S can program their output. For each input (ID, r_{ID}) S chooses a random $e_{ID} \xleftarrow{\$} \mathbb{Z}_q$ and sets $H_1(ID, r_{ID}) = e_{ID}$. Similar work is done for H_2.

The adversary is allowed to ask the KGC for the secret keys of users of its choice and thus S must be able to simulate the key derivation process. As one can notice, when the adversary asks for the secret key of a user, the simulator is always able to respond, since it has chosen the master secret key x by itself.

At the beginning of the game S guesses the test session and its holder (let us call him Bob). Also let Alice be the other party of the session. Sessions different from the test session are easily simulated since S knows all the informations needed to compute the session keys and answer to session key queries.

Without loss of generality we assume that the test session is at Bob (and thus the corresponding matching session is at Alice). Since we are in the case when the adversary is passive during the execution of the protocol, the simulator chooses the messages of the test session.

Let $(A, r_A, s_A), (B, r_B, s_B)$ be the identity informations and the secret keys of Alice and Bob respectively. The simulator sets Alice's message as $(A, r_A, u_A = U)$ while the one from Bob is $(B, r_B, u_B = V)$. S is implicitly setting $t_A = u, t_B = v$. Since H_2 is modeled as a random oracle, if the adversary has success into distinguishing the real session key from a random value, it must have queried H_2 on the correct input $(z = u_B^{s_A} u_A^{s_B}, z_3 = g^{uv})$. Thus S chooses a random value among all the queries that it received from the adversary. Since the number of queries Q_2 is polynomially bounded, the simulator can find $z_3 = W$ with non-negligible probability ϵ / nQ_2. This completes the proof of this case.

Remark 2. If we would assume the simulator having access to a Gap-DH oracle, S might use the oracle to test, for all queries (z, z_3) made by the adversary, if $DH(U, V, z_3) =$ "yes" and then output z_3 for which the test is true. In this case the security of Saeednia's protocol would reduce to the Gap-DH Assumption instead of CDH, but we would not have the Q_2 loss factor.

Acknowledgements

The authors would like to thank Gregory Neven for suggesting the use of the General Forking Lemma.

References

1. M. Abdalla, M. Bellare and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. *In proceedings of CT-RSA 2001*, LNCS vol. 2020, pp. 143-158.
2. M. Bellare and G. Neven. New Multi-Signature Schemes and a General Forking Lemma. *In proceedings of the 13th Conference on Computer and Communications Security – ACM CCS 2006*, ACM Press, 2006.

3. M. Bellare and A. Palacio. The Knowledge-of-Exponent Assumptions and 3-round Zero-Knowledge Protocols. *Advances in Cryptology – CRYPTO 2004*, LNCS vol. 3152.
4. Dan Boneh, Xavier Boyen. Short Signatures without Random Oracles. *Advances in Cryptology – Eurocrypt 2004*, LNCS vol. 3027.
5. Dan Boneh, Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. *SIAM J. Comput.* 32(3): 586-615 (2003) (Also in CRYPTO 2001.)
6. Colin Boyd, Kim-Kwang Raymond Choo. Security of Two-Party Identity-Based Key Agreement. Mycrypt 2005: 229-243
7. Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, Kenneth G. Paterson. Efficient One-Round Key Exchange in the Standard Model. *In proceedings of ACISP 2008*, LNCS vol. 5107, pp. 69-83
8. Colin Boyd, Wenbo Mao, Kenneth G. Paterson. Key Agreement Using Statically Keyed Authenticators. *In proceedings of ACNS 2004*, LNCS vol. 3089, pp. 248-262
9. R. Canetti, H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. *Advances in cryptology – EUROCRYPT 2001*, LNCS vol. 2045, pp. 453-474
10. R. Canetti, H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Advances in cryptology – EUROCRYPT 2002*, LNCS vol. 2332, pp. 337-351
11. D. Cash, E. Kiltz and V. Shoup. The Twin Diffie-Hellman Problem and Applications *Advances in cryptology – EUROCRYPT 2008*, LNCS vol. 4965, pp. 127-145.
12. L. Chen, Z. Cheng, Nigel P. Smart. Identity-based key agreement protocols from pairings. Int. J. Inf. Sec. 6(4): 213-241 (2007)
13. L. Chen and C. Kudla. Identity Based Authenticated Key Agreement Protocols from Pairings In 16th IEEE Computer Security Foundations Workshop - CSFW 2003, pages 219-233. IEEE Computer Society Press, 2003.
14. Q. Cheng and C. Ma. Ephemeral Key Compromise Attack on the IB-KA protocol. Cryptology Eprint Archive, Report 2009/568. http://eprint.iacr.org/2009/568.
15. I. Damgård. Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks. *Advances in Cryptology – CRYPTO'91*, LNCS vol. 576.
16. W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 1976, vol. 22, n. 6 , pp. 644-654
17. A. Fiat and A. Shamir How to Prove Yourself: Practical Solutions of Identification and Signature Problems. *Advances in cryptology – CRYPTO 1986*, LNCS vol. 263, pp. 186-194
18. D. Fiore and R. Gennaro. Making the diffie-hellman protocol identity-based. In *Proceedings of CT-RSA 2010*. LNCS. Springer-Verlag. to appear – also in http://eprint.iacr.org/2009/174.
19. S.D. Galbraith, K.G. Paterson and N.P. Smart. Pairings for Cryptographers. *Cryptology ePrint Archive, Report 2006/165*, 2006. `http://eprint.iacr.org`.
20. C.H. Lim and P.J. Lee More Flexible Exponentiation with Precomputation In *Crypto'94*, pp.95–107, LNCS no. 839.
21. C. Gentry. Practical Identity-Based Encryption Without Random Oracles. *Advances in cryptology – proceedings of EUROCRYPT 2006*, 2006, LNCS vol. 4004, pp. 494-510
22. Gunther, C.G. An Identity-Based Key-Exchange Protocol. *Advances in cryptology – proceedings of EUROCRYPT 1989*, 1989, LNCS, vol. 434, pp. 29-37.
23. S. Hada and T. Tanaka. On the Existence of 3-round Zero-Knowledge Protocols. *Advances in Cryptology – CRYPTO 1998*, LNCS vol. 1462

24. E. Kiltz. Direct Chosen-Ciphertext Secure Identity-Based Encryption in the Standard Model with short Ciphertexts. Cryptology Eprint Archive, Report 2006/122. http://eprint.iacr.org/2006/122.
25. E. Kiltz and D. Galindo. Direct Chosen-Ciphertext Secure Identity-Based Key Encapsulation Without Random Oracles. Cryptology Eprint Archive, Report 2006/034. http://eprint.iacr.org/2006/034.
26. Hugo Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. *Advances in cryptology – CRYPTO 2005*, LNCS vol. 3621, pp. 546-566
27. 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.
28. U. Maurer and S. Wolf. Diffie-Hellman oracles. *Advances in cryptology – CRYPTO 1996*, LNCS vol. 1109, pp. 268-282
29. E. Okamoto. Key Distribution Systems Based on Identification Information. In Advances in Cryptology, Crypto 1987, pp. 194-202. LNCS Vol. 293/1988.
30. E. Okamoto and K. Tanaka. Key Distribution System Based on Identification. Information. IEEE Journal on Selected Areas in Communications, 7(4):481-485, May 1989.
31. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361-396 (2000).
32. S. Saeednia. Improvement of Gunther's identity-based key exchange protocol. *Electonics Letters* vol. 36, Issue 18, 31 Aug 2000, pp. 1535 - 1536
33. R. Sakai, K. Ohgishi and M. Kasahara. Cryptosystems based on pairing. *In Symposium on Cryptography and Information Security*, Okinawa, Japan, 2000.
34. Adi Shamir Identity-Based Cryptosystems and Signature Schemes *Advances in Cryptology – Proceedings of CRYPTO '84*, 1985, pp. 47-53
35. C.P. Schnorr. Efficient identification and signatures for smart cards. *Advances in Cryptology – CRYPTO '89*, 1989, LNCS vol. 435, pp. 239-252
36. N. P. Smart. An identity-based authenticated key-agreement protocol based on the Weil pairing. *Electronics letters*, 2002, vol. 38, pp.630-632.
37. D.K. Smetters, G. Durfee. Domain-based Administration of Identity-Based Cryptosystems for Secure E-Mail and IPSEC. In: SSYM 2003: Proceedings of the 12th Conference on USENIX Security Symposium, p. 15. USENIX Association (2003)
38. Y. Wang. Efficient Identity-Based and Authenticated Key Agreement Protocol. Cryptology ePrint Archive, Report 2005/108, 2005. http://eprint.iacr.org/2005/108/.