Security Analysis of KEA Authenticated Key Exchange Protocol

Kristin Lauter[1] and Anton Mityagin[2]

[1] Microsoft Research, One Microsoft Way, Redmond, WA 98052
klauter@microsoft.com
[2] Department of Computer Science, University of California San Diego
9500 Gilman Dr., La Jolla, CA 92037
amityagin@cs.ucsd.edu

Abstract. KEA is a Diffie-Hellman based key-exchange protocol developed by NSA which provides mutual authentication for the parties. It became publicly available in 1998 and since then it was neither attacked nor proved to be secure. We analyze the security of KEA and find that the original protocol is susceptible to a class of attacks. On the positive side, we present a simple modification of the protocol which makes KEA secure. We prove that the modified protocol, called KEA+, satisfies the strongest security requirements for authenticated key-exchange and that it retains some security even if a secret key of a party is leaked. Our security proof is in the random oracle model and uses the Gap Diffie-Hellman assumption. Finally, we show how to add a key confirmation feature to KEA+ (we call the version with key confirmation KEA+C) and discuss the security properties of KEA+C.

1 Introduction

Authenticated Key Exchange. Generally, key exchange protocols allow 2 parties who share no secret information to compute a secret key via public communication. Authenticated key exchange (AKE) not only allows parties to compute the shared key but also ensures authenticity of the parties. A party can compute a shared key only if it is the one it claims to be. AKE protocols operate in a public key infrastructure and the parties use each other's public keys to construct a shared secret.

Natural Solution: Signed Diffie-Hellman. One possible solution for authenticated key exchange is to execute a Diffie-Hellman key exchange and to sign all the communication sent between the parties. Such an AKE protocol is sometimes referred to as Signed Diffie-Hellman. Let G be a group of prime order and denote by g a generator of G. Assume that the parties have secret/public keys for some digital signature scheme SIG and that parties know each other's registered public keys. Denote the signature of a message M under the secret key of a party $\mathbb{A}$ as $SIG_{\mathbb{A}}(M)$.

The protocol has 2 passes. First, an initiator $\mathbb{A}$ picks an ephemeral secret key x at random and sends to a responder $\mathbb{B}$ a tuple $\{g^x,\ SIG_{\mathbb{A}}(g^x, \mathbb{B})\}$. The responder $\mathbb{B}$ picks an ephemeral secret key y and replies with a tuple $\{g^y,\ SIG_{\mathbb{B}}(g^y, \mathbb{A})\}$.

Parties then verify each other's signatures and if accepted, compute a shared session key $K = g^{xy}$. The protocol is depicted in Figure 1. This protocol was

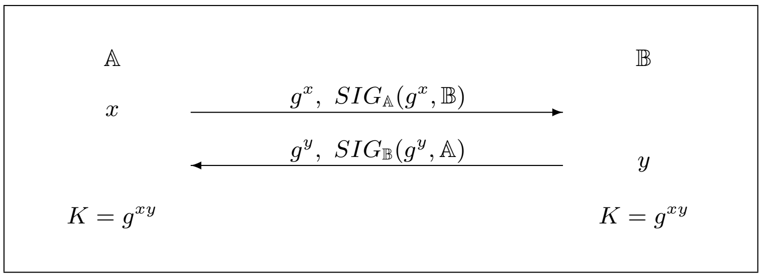
$\mathbb{A}$
$\mathbb{B}$
x
$g^x,\ SIG_{\mathbb{A}}(g^x, \mathbb{B})$
$g^y,\ SIG_{\mathbb{B}}(g^y, \mathbb{A})$
y
$K = g^{xy}$
$K = g^{xy}$

Fig. 1. Signed Diffie-Hellman authenticated key-exchange

formally analyzed by Shoup [17] and it is proven to be secure (we will discuss below in detail what security means) against an adversary who can reveal session keys of honest key-exchange sessions but who cannot reveal ephemeral secret keys.

It is worth noting that Signed Diffie-Hellman AKE can be broken if an adversary can reveal ephemeral secret keys of the parties. Exposure of ephemeral secret keys can occur in practical implementations of AKE protocols if ephemeral keys are precomputed or if they are stored in insecure storage. If an adversary $\mathbb{M}$ reveals an ephemeral secret key x used by $\mathbb{A}$ in some session with $\mathbb{B}$, then $\mathbb{M}$ can impersonate $\mathbb{A}$ to $\mathbb{B}$ by starting a session with $\mathbb{B}$ and sending the same tuple $\{g^x, SIG_{\mathbb{A}}(g^x, \mathbb{B})\}$. $\mathbb{B}$ will accept this tuple because the signature is valid and then $\mathbb{M}$ can compute a session key using the knowledge of x.

Security of Authenticated Key Exchange. For AKE protocols there are a surprisingly large number of possible attack scenarios and there is no single security definition. We sketch 3 security notions which seem to capture all possible attacks, and give their precise definitions in Section 2:

1. The main security requirement (we will call it AKE security) as introduced by Bellare and Rogaway [4] and further refined by Bellare, Pointcheval and Rogaway [3] and by Canetti and Krawczyk [9], considers a multi-party experiment with unauthenticated communication channels (called the AKE experiment). The adversary controls all the communication and can corrupt some of the parties. Moreover, the adversary selects honest parties to participate in key-exchange sessions. The adversary must select an uncorrupted session called a test session and then he is given a challenge, which is either the session key of the test session or a randomly selected key. The goal of the adversary is to distinguish between these 2 cases.

2. One of the properties not captured by AKE security is Perfect Forward Secrecy (PFS). Perfect Forward Secrecy says that an adversary in the AKE experiment who corrupted one of the parties (that is, revealed the long-term secret key), should not be able to reveal session keys of past sessions executed by that

party. Krawzcyk [11] shows that no 2-pass AKE protocol can achieve perfect forward secrecy. Alternatively, he presents a notion of weak perfect forward secrecy (wPFS). Weak perfect forward secrecy guarantees security only for those previous sessions executed without the adversary's intrusion.

3. The last security requirement is resistance to key compromise impersonation (KCI). An adversary who reveals a long-term secret key of some party $\mathbb{A}$ should be unable to impersonate other parties to $\mathbb{A}$ (still, an adversary can impersonate $\mathbb{A}$ to anyone else).

All these security notions can involve either a "weak" or a "strong" adversary: a weak adversary can reveal session keys of sessions executed by honest parties while a strong adversary can reveal both session keys and ephemeral secret keys. Both adversaries can also do total corruptions, i.e. take full control over honest parties. We assume that a certificate authority (CA), upon registering a public key, doesn't require a party to prove knowledge of the corresponding secret key. That is, a certificate authority will register arbitrary public keys presented by parties, even ones matching existing public keys of other parties. In contrast, proof of knowledge of the secret key is required by many existing AKE protocols, but these checks are rarely done in practice.

KEA Protocol. KEA authenticated key exchange [15] was designed by NSA in 1994 and originally its design was kept secret. It was declassified and became available to the public in 1998. KEA involves 2 parties, $\mathbb{A}$ and $\mathbb{B}$, with respective secret keys a and b and public keys g^a and g^b. We assume that parties know each other's registered public keys. The protocol first executes a standard Diffie-Hellman communication: parties select ephemeral secret keys x and y at random and exchange ephemeral public keys g^x and g^y. Then each party computes g^{ay} and g^{bx} and computes a session key K by applying a hash function F to $g^{ay} \oplus g^{bx}$. The original description of KEA specifies F to be a certain function built on the SKIPJACK block cipher [15]. The design of KEA closely resembles Protocol 4 from Blake-Wilson et al. [5]. They suggest computing a session key as $H(g^{ay}, g^{bx})$, where H is a cryptographic hash function. Blake-Wilson et al. conjectured (without proof) the security of their protocol provided H is modeled by a random oracle.

Attacks on KEA. We observe that AKE security of KEA (even against a weak adversary) can be violated if an adversary can register arbitrary public keys. Consider the following adversary $\mathbb{M}$. $\mathbb{M}$ registers a public key g^a of some honest party $\mathbb{A}$ as $\mathbb{M}$'s own public key. Then $\mathbb{M}$ intercepts a key-exchange session between $\mathbb{A}$ and some other honest party $\mathbb{B}$ and at the same time starts a session between $\mathbb{M}$ and $\mathbb{B}$. Now $\mathbb{M}$ forwards ephemeral public key g^x from $\mathbb{A}$ to $\mathbb{B}$ and ephemeral public key g^y from $\mathbb{B}$ to $\mathbb{A}$. Since $\mathbb{M}$ has the same public key as $\mathbb{A}$, both $\mathbb{A}$ and $\mathbb{B}$ will compute identical session keys, however they participate in two different key-exchange sessions. $\mathbb{B}$ participates in a session with $\mathbb{M}$ while $\mathbb{A}$ participates in a session with $\mathbb{B}$. Finally, $\mathbb{M}$ reveals a session key of one of the sessions and announces the other session as a test session. Given a challenge key, $\mathbb{M}$ compares it to the revealed key. If they are the same, $\mathbb{M}$ decides that the challenge is a correct key for the test session and if different, $\mathbb{M}$ decides that the challenge key was chosen at random. The demonstrated attack breaks AKE security against a weak adversary (who can only reveal session keys). This attack is often called as Unknown Key Share (UKS) attack.

One possible counter-measure to the above attack is not to allow 2 parties to have the same public key, and this check can be done by a certificate authority. We note that this counter-measure also wouldn't work. In the previous attack's scenario, an adversary can pick any exponent k, register a public key g^{ak} and instead of sending g^y as a response to $\mathbb{A}$, send a value g^{yk}. This way, both $\mathbb{A}$ and $\mathbb{B}$ will again have the same session key $H(g^{ayk} \oplus g^{bx})$.

Security fix: KEA+. We present a modified version of the KEA protocol, called KEA+, which is resistant to the above attacks. We prove that no such attacks on KEA+ are possible and that KEA+ satisfies the strongest known security requirement. The main idea behind KEA+ is to incorporate parties' identities in the computation of a session key. Interestingly, this simple feature of the protocol turns out to be crucial in the security analysis and avoids the proof-of-possession requirement.

The KEA+ protocol proceeds as follows. First, parties $\mathbb{A}$ and $\mathbb{B}$ randomly select ephemeral secret keys x and y and exchange ephemeral public keys g^x and g^y. Then parties verify that the received ephemeral public keys are in the group G and compute a session key K as $H(g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$, where H can be an arbitrary cryptographic hash function. In the security analysis we model H by a random oracle. Figure 2 depicts actions performed by the parties. We note that verifying that the ephemeral public keys are in the group G is essential for the security of the protocol. Otherwise, the protocol is vulnerable to a so-called "small subgroup" attack.

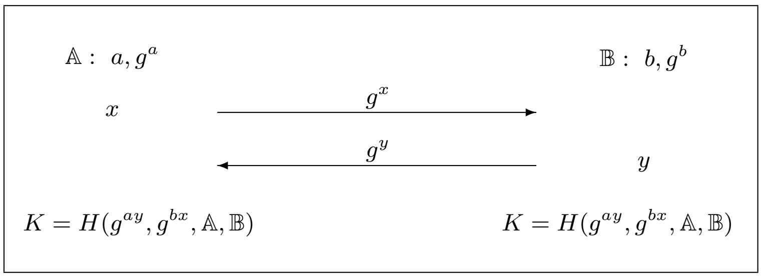
$\mathbb{A}:\ a, g^a$
$\mathbb{B}:\ b, g^b$
x
g^x
g^y
y
$K = H(g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$K = H(g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$

Fig. 2. New KEA+ protocol

We prove that KEA+ protocol satisfies AKE security, weak perfect forward secrecy and security against KCI attacks. All these results involve a strong adversary who can reveal ephemeral secret keys of the parties as well as session keys. The results hold under either the standard Gap Diffie-Hellman (GDH) assumption in a group G, as defined by Okamoto and Pointcheval [16], or under a stronger Pairing Diffie-Hellman (PDH) assumption. The latter assumption means hardness of the computational Diffie-Hellman problem, where a solver is given access to a bilinear pairing oracle. The reason for having two reductions (one to GDH and one to PDH) lies in the concrete security analysis. The reduction to PDH achieves better concrete security compared to the reduction to GDH.

We stress that KEA+ does not require parties to prove possession of secret keys upon key registration. Parties can register arbitrary public keys, even ones matching somebody else's keys. Moreover, an adversary can register keys for corrupted parties at any time in the experiment. Our security results imply that these powers do not allow the adversary to break the security of KEA+.

Key Confirmation: KEA+C. The 2-pass KEA+ protocol is optimized for communication and has exactly the same communication as the original Diffie-Hellman protocol. While satisfying the strongest security requirement, it doesn't provide delivery guarantees which might be desirable for some applications. Namely, KEA+ doesn't provide assurance that the other party actually completed the session. To address this issue, we add one more pass of communication to KEA+ to obtain a protocol called KEA+C, or KEA+ with key confirmation.

KEA+C involves a message authentication code to construct a confirmation message. KEA+C achieves a key confirmation property [11], namely it assures that the other party is able to compute the session key. As well, KEA+C satisfies the full perfect forward secrecy requirement lacking in KEA+. Finally, results of Canetti and Krawczyk [9] imply that KEA+C satisfies Universally Composable security defined by [6], which ensures that KEA+C can be securely executed concurrently with arbitrary other protocols.

History and Related Work. Defining security of authenticated key exchange dates back to the work Bellare and Rogaway [4] from 1993. Following work of Bellare, Pointcheval and Rogaway [3] and Shoup [17], the current security definition was formulated by Canetti and Krawczyk [9]. We refer the reader to [7] for a comparison and a discussion of existing security definitions for authenticated key exchange.

To date, a great number of AKE protocols have been proposed and many of them were subsequently broken. Currently, there exist a number of protocols that satisfy AKE security against adversaries who cannot reveal ephemeral secret keys (weak adversaries), and only a few protocols which are secure against strong adversaries. AKE protocols proved to be secure against strong adversaries include SIG-DH from [9], SIGMA [10] and HMQV [11].

We compare our KEA+ protocol with the recent HMQV protocol [11], which combines great efficiency with the highest security level. KEA+ and HMQV are both proven to achieve AKE security, security against KCI and wPFS[3]. However, the security of HMQV relies on the knowledge of the exponent assumption[4] [2] and doesn't have a concrete security analysis. As noted by Menezes [14], the

[3] In fact, the wPFS requirement from [11] is stronger than ours. They allow an adversary to reveal long-term keys of both parties, while we only allow revealing the long-term key of at most one of the parties.

[4] We remark that in the analysis of HMQV this assumption is only needed to ensure security against strong adversaries (who can reveal ephemeral secret keys).

concrete security reduction of [11] appears to be inefficient. Our security proof doesn't employ the knowledge of exponent assumption and provides a tight security reduction (under the Pairing Diffie-Hellman assumption). Our protocol requires the same number of exponentiations as HMQV (although one of the exponentiations in HMQV involves half-size exponents).

After submitting our paper we discovered the parallel independent work of Kudla and Paterson [13]. They use very similar techniques to prove the security of a modification of Protocol 4 from Blake-Wilson et al [5], which can be viewed as the KEA+ protocol where identities of the parties are excluded from the key computation. We want to highlight some differences between our work and theirs. First, their protocol is vulnerable to the UKS attack. This attack is not captured by their security analysis, as the security model of [13] requires that all parties (even ones controlled by the adversary) do key-generation properly. Second, they only prove security against weak adversaries (which cannot reveal ephemeral keys) and their security proof doesn't contain a concrete security analysis. Finally, we discuss a key-confirmation property and analyze the security of our KEA+C protocol.

2 Definitions

Notation. All protocols in the paper use a mathematical group G of a known prime order q where the Diffie-Hellman problem is computationally infeasible. The group G can be implemented either as a multiplicative subgroup of a finite field or as a group of points on an elliptic curve. We denote by g a generator of G and write the group operation in a multiplicative manner.

Throughout the paper, we will apply hash functions and signature schemes to lists of several arguments. In these cases, we write function arguments separated by commas, for example $H(X, Y, Z)$. Doing that, we assume that we have a collision-free encoding which maps lists of arguments to binary strings. Also, we assume that parties' identities are arbitrary binary strings.

Gap Diffie-Hellman (GDH). A computational Diffie-Hellman (CDH) problem is, given g^x and g^y (for randomly chosen x and y) to compute g^{xy}. A Decisional Diffie-Hellman (DDH) Oracle DDH takes input a triple $(g^x, g^y, Z) \in G^3$ and outputs 1 if $Z = g^{xy}$ and 0 otherwise. The Gap-Diffie-Hellman [16] problem is the CDH problem, where the solver algorithm is additionally given access to a DDH oracle. The advantage of such a solver $\mathbb{M}$, denoted as $\mathbf{Adv}^{\mathrm{GDH}}(\mathbb{M})$, is $\mathbb{M}$'s winning probability in the CDH problem. We say that G satisfies the Gap-Diffie-Hellman (GDH) assumption if no feasible adversary exists to solve the CDH problem, even provided with a DDH-oracle. Gap Diffie-Hellman is a standard cryptographic assumption which was used to establish the security of several key agreement protocols [1, 18, 12].

Pairing Diffie-Hellman (PDH). Let G' be another mathematical group of the same order as G with efficiently computable group operation. A function $e : G \times G \to G'$ is a bilinear pairing if it is non-degenerate and if for any pair $g^a, g^b \in G$, $e(g^a, g^b) = e(g, g)^{ab}$. A pairing oracle P associated with the pairing function e and the group G' takes two elements $X, Y \in G$ and returns $e(X, Y)$. The Pairing Diffie-Hellman problem is the CDH problem, where the solver is additionally given access to the pairing oracle P. The advantage $\mathbf{Adv}^{\mathrm{PDH}}(\mathbb{M})$ of a PDH solver $\mathbb{M}$ is the probability of $\mathbb{M}$ solving the CDH problem. We say that G satisfies the PDH assumption if no feasible adversary exists to solve the CDH problem provided with an arbitrary PDH-oracle.

In the groups which have a bilinear pairing, PDH problem is equivalent to the original CDH problem. As well, one can consider PDH problem in the groups where no efficient pairing operation is known. We find the Pairing Diffie-Hellman assumption to be as justified as GDH since the only known way to compute DDH in groups where CDH is hard is via a pairing function.

AKE Security. The AKE experiment involves multiple honest parties and an adversary $\mathbb{M}$ connected via an unauthenticated network. The adversary selects parties to execute key-exchange sessions and selects an order of the sessions. It can also corrupt some of the parties. An adversary has full control over the communications and he can delay/cancel/modify any message.

There is a special party, $\mathbb{CA}$, called the certificate authority, who registers the public keys of the parties. We model a $\mathbb{CA}$ as a trusted directory. The $\mathbb{CA}$ registers arbitrary keys (even those matching keys of other parties) with the only restriction that no party can have more than one registered public key. In the beginning of the AKE experiment all honest parties generate their public keys and register them with the $\mathbb{CA}$. The adversary can register public keys of adversary-controlled parties at any time in the experiment, even during the execution of an AKE session. That is, the adversary is allowed to mount the Unknown Key Share attack and related attacks.

To start an AKE session, the adversary activates an honest party and specifies that party's role in the exchange (initiator or responder) and the identity of the other participant. We identify an AKE session by a 4-tuple $(\mathbb{A}, \mathbb{B}, \mathit{role}, \mathit{Comm})$, where $\mathbb{A}$ is the executing party, $\mathbb{B}$ is the other party, $\mathit{role} \in \{\mathit{initiator}, \mathit{responder}\}$ is $\mathbb{A}$'s role in the protocol and Comm consists of all messages sent and received by $\mathbb{A}$. We stress that an AKE session is executed by a single party: since all communication is controlled by an adversary, a party executing a session cannot know for sure whom it is talking to. We call the session which is supposed to be executed by the other party as the matching AKE session. For example the session $(\mathbb{A}, \mathbb{B}, \mathit{initiator}, \mathit{Comm})$ matches $(\mathbb{B}, \mathbb{A}, \mathit{responder}, \mathit{Comm})$ and vice versa. A party completes the session when it receives the last message from the other party and computes the session key.

An adversary can corrupt honest parties as well as reveal session information. When an adversary corrupts a party (often referred to as a Corrupt query), he learns the long-term secret key of that party and gets full control of that party from that moment on. Revealing session information (often referred to as a Reveal query) only affects a single AKE session. We distinguish between 2 reveal scenarios. First, an adversary can learn only a session key of a completed session. We call it a session key reveal and we call an adversary who only makes session key reveals (in addition to total corruptions) a "weak" adversary. A second type of adversary, called a "strong" adversary, is also allowed to reveal an ephemeral secret key of a party executing a session.

We say that a completed session is "clean" if this session as well as its matching session (if it exists) is not corrupted (neither session key nor ephemeral secret key were revealed by $\mathbb{M}$) and if none of the participating parties is corrupted.

Eventually an adversary should select a clean completed session ($\mathbb{A}$, $\mathbb{B}$, *role*, *Comm*), which is called a test session. A challenger tosses a coin to obtain $b \in \{0, 1\}$; if $b = 0$ he sets K_C to be the session key of the test session and otherwise he sets K_C to be a random string of the same length. A challenger gives the challenge K_C to the adversary. After receiving the challenge, the adversary continues the experiment, but is not allowed to corrupt the test session nor any of the parties involved in the test session. The experiment ends when the adversary outputs a guess bit b'.

The advantage of the adversary $\mathbb{M}$ participating in the above AKE experiment against AKE protocol Π is defined as

$$\mathbf{Adv}_{\Pi}^{\mathrm{AKE}}(\mathbb{M}) = Pr[b = b'] - \frac{1}{2}.$$

We say that an AKE protocol is secure if no feasible AKE adversary has more than a negligible advantage in the AKE experiment.

Perfect Forward Secrecy (PFS). The Perfect Forward Secrecy property of an AKE protocol guarantees that an adversary who corrupts a party cannot gain any information about session keys of previous AKE sessions. We formally define PFS by modifying the AKE experiment as follows. Now we allow the adversary to corrupt at most one of the two participants of the test session after the test session is completed. As in the original AKE experiment, the adversary must distinguish between the session key of the test session and a random key.

Krawczyk [11] observed that no 2-pass AKE protocol can achieve full PFS in a presence of strong adversaries. To address forward secrecy of 2-pass protocols, he suggests a relaxed notion, called weak PFS (wPFS). Weak PFS only guarantees security of those AKE sessions executed without active adversarial intrusion. We define weak PFS by limiting the set of clean sessions to only those executed without active adversarial intrusion. That is, the adversary is only allowed to forward communications in the test session and its matching session and is not allowed to cancel or modify them.

We remark that our definitions of PFS and wPFS are weaker than the ones by Krawczyk [11]. Krawczyk's definitions allow an adversary to corrupt both participants of the test session, while our definition only allows corruption of at most one of the participants.

Security against Key Compromise Impersonation (KCI). KCI security considers a scenario when an adversary reveals a long-term secret key of some party $\mathbb{A}$ without corrupting $\mathbb{A}$ (that is, without taking full control over $\mathbb{A}$). Note that in this case an adversary can impersonate $\mathbb{A}$ to anyone else. KCI security guarantees that an adversary should be unable to impersonate other parties to $\mathbb{A}$.

We define KCI security by the following modification of the AKE experiment. We allow an adversary to make a new type of corruption: to reveal a long-term secret key of a party without taking control over it. Now, a test session is allowed to be a clean session, where the party running the session had its long-term secret key revealed. Still, an adversary is not allowed to corrupt or reveal the long-term secret key of the other party.

3 Security of KEA+

AKE SECURITY OF KEA+. We show that the KEA+ protocol with a hash function modeled as a random oracle satisfies AKE security against a strong adversary under the GDH or PDH assumptions in a group G.

REDUCTION TO A FORGING ATTACK. Assume by contradiction that there exists some efficient adversary $\mathbb{M}$ against the KEA+ protocol. Let $(\mathbb{A}, \mathbb{B}, \mathit{initiator}, X, Y)$ be a test session in some AKE experiment. Let A be the public key of $\mathbb{A}$ and B be the public key of $\mathbb{B}$. Denote by $CDH(\cdot, \cdot)$ the computational Diffie-Hellman function. We observe that since the session key of a test session is computed as a hash value of a 4-tuple $\{CDH(A, Y), CDH(B, X), \mathbb{A}, \mathbb{B}\}$, the adversary $\mathbb{M}$ has only 2 ways to distinguish K from a random string:

1. Forging attack. At some point $\mathbb{M}$ queries H on the tuple
$$\sigma = (CDH(A, Y), CDH(B, X), \mathbb{A}, \mathbb{B}).$$
2. Key-replication attack. $\mathbb{M}$ succeeds in forcing the establishment of a session that has the same signature (and subsequently, the same session key) as the test session. In this case $\mathbb{M}$ can learn the test-session key by simply making a reveal query on the session with the same key, without having to learn the value of the test signature.

We denote a 4-tuple $\sigma = (g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$ as the "signature" of a key exchange session. Recall that the key for the test session is the value of a random oracle H on the test signature σ. Since H is a truly random function, an adversary has only 2 ways of learning $H(\sigma)$: $\mathbb{M}$ can either query σ to H himself or σ can be queried to H by some honest party and $\mathbb{M}$ can reveal $H(\sigma)$ by corrupting that party. Otherwise, $\mathbb{M}$ cannot distinguish information-theoretically between $H(\sigma)$ and a random string. Note that these cases correspond to a forging attack and a key-replication attack respectively. If $\mathbb{M}$ doesn't mount either of these attacks, then it cannot win the experiment with probability any better than 1/2.

Let's see that a key-replication attack is impossible. In that case, if an adversary finds some session with the same signature σ as the test session, then this session must be executed by the same 2 parties, $\mathbb{A}$ and $\mathbb{B}$. Let the ephemeral public keys of this session be X' and Y'. Since the session has the same signature as the test session, $CDH(A, Y')$ must be equal to $CDH(A, Y)$ and $CDH(B, X')$ – equal to $CDH(B, X)$. This implies that $X' = X$ and $Y' = Y$, and thus the sessions must be identical.

We're left to show impossibility of a forging attack. We are going to show that given an efficient forging adversary against KEA+, we can construct an adversary which efficiently solves the GDH problem. We first establish a reduction to GDH and then show how to modify it to obtain an improved reduction to PDH.

Security against a Simplistic Adversary. First we show how the reduction works in the simplistic case of a certain (very limited) adversary and then proceed to the general case. Assume that the AKE experiment only involves 2 honest parties $\mathbb{A}$ and $\mathbb{B}$ and that the adversary $\mathbb{M}$ passively observes a single AKE session executed by these parties and selects it as a test session. In this case the reduction to the GDH problem is natural: given a GDH challenge (X_0, Y_0) the GDH solver $\mathbb{S}$ runs the AKE experiment with parties $\mathbb{A}$ and $\mathbb{B}$ and the adversary $\mathbb{M}$. $\mathbb{S}$ sets the first challenge value X_0 to be the long-term public key of $\mathbb{A}$ and selects keys for $\mathbb{B}$ in the usual way. When $\mathbb{A}$ and $\mathbb{B}$ execute a test session, $\mathbb{A}$ picks a random x and sends g^x to $\mathbb{B}$, while $\mathbb{B}$ responds with Y_0. Note that a view of $\mathbb{M}$ in this simulated AKE experiment is distributed identically to a view of $\mathbb{M}$ in a true AKE experiment and thus $\mathbb{M}$ wins with the same probability. As we justified earlier, if $\mathbb{M}$ wins, he should query H a signature $\sigma = (CDH(X_0, Y_0), g^{bx}, \mathbb{A}, \mathbb{B})$. Note that in this case σ contains $CDH(X_0, Y_0)$, which is a solution to the original CDH problem.

Idea of the General-Case Reduction. The idea of the reduction is very similar to the simple case with the difference that $\mathbb{S}$ selects at random a party $\mathbb{A}$ (to put a first challenge value in $\mathbb{A}$'s public key) and a session executed by $\mathbb{A}$ and some other party $\mathbb{B}$ (to put a second challenge value in $\mathbb{B}$'s ephemeral public key). The complication that arises in the general case is how to handle session-corrupt queries involving the selected party $\mathbb{A}$. Since $\mathbb{S}$ doesn't know a secret key for $\mathbb{A}$'s public key, it cannot compute a signature (nor a session key) for such a session. We handle this case by picking a session key at random without computing a signature. Then $\mathbb{S}$ uses the DDH oracle to test if $\mathbb{M}$ queries H with a signature for such a session and if "yes", returns the previously selected session key. We proceed with a formal description and analysis of the reduction.

Construction of a GDH Solver $\mathbb{S}$. Let $\mathbb{M}$ be an AKE adversary against KEA+. Consider the following GDH adversary $\mathbb{S}$:

$\mathbb{S}$ takes input a pair $(X_0, Y_0) \in G^2$. $\mathbb{S}$ is also given access to a DDH oracle DDH. $\mathbb{S}$ creates an AKE experiment which includes a number of honest parties and an adversary $\mathbb{M}$. We assume that the experiment involves at most n parties and that each party participates in at most k AKE sessions. $\mathbb{S}$ randomly selects one of the honest parties (say, this is a party $\mathbb{A}$) and sets the public key of $\mathbb{A}$ to be X_0. All the other parties compute their keys normally. $\mathbb{S}$ picks a number i_k at random from $\{1, \ldots, k\}$ and initializes the counter at $i = 1$ (i counts sessions that A participates in). $\mathbb{S}$ runs an AKE experiment with adversary $\mathbb{M}$ and handles queries made by $\mathbb{M}$ as follows:

1. When $\mathbb{M}$ queries a hash function H on a string v, return the value of Hsim(v). The procedure Hsim$(\cdot)$ which simulates a random oracle H is described later on.

2. When $\mathbb{M}$ starts a session $(\mathbb{B}, \mathbb{C}, role)$ between parties $\mathbb{B}$ and $\mathbb{C}$ both different from a selected party $\mathbb{A}$, $\mathbb{S}$ follows the protocol for KEA+. Denote $\mathbb{B}$'s secret key as b, $\mathbb{B}$'s public key as $B = g^b$ and $\mathbb{C}$'s public key as C. If $role = initiator$, $\mathbb{B}$ picks a random exponent x, returns $X = g^x$, waits for the reply Y and computes a session key $K = \text{HSIM}(Y^b, C^x, \mathbb{B}, \mathbb{C})$. If $role = responder$, $\mathbb{B}$ waits for $\mathbb{C}$'s initiating message X, picks a random exponent y, replies with g^y and computes a session key $K = \text{HSIM}(C^y, X^b, \mathbb{C}, \mathbb{B})$.
3. When $\mathbb{M}$ starts a session $(\mathbb{A}, \mathbb{C}, role)$ (here $\mathbb{A}$ is the special party whose public key is a GDH challenge X_0), $\mathbb{S}$ cannot follow the protocol since it doesn't know a secret for $\mathbb{A}$'s public key. Denote $\mathbb{C}$'s public key as C. If $\mathbb{A}$ is an initiator, it picks a random exponent x, sends g^x to $\mathbb{C}$ and waits for the reply Y. Now it sets a session key to be $\text{HSPEC}(1, Y, C^x, \mathbb{A}, \mathbb{C})$, see the description of the procedure HSPEC below. If $\mathbb{A}$ is the responder, it waits for an initiating message X, picks a random exponent y, replies with g^y and computes a session key $K = \text{HSPEC}(2, X, C^y, \mathbb{C}, \mathbb{A})$.
4. When $\mathbb{M}$ starts a session $(\mathbb{B}, \mathbb{A}, role)$ for some party $\mathbb{B}$, where the second party is the selected party $\mathbb{A}$, $\mathbb{S}$ first checks if $i = i_k$. If "no", $\mathbb{S}$ increments the counter i and behaves according to the rule for Query 2. If the check succeeds, $\mathbb{S}$ declares $(\mathbb{B}, \mathbb{A}, role)$ to be a "special session". In a special session, $\mathbb{B}$ outputs a message Y_0 (which is the second part of the GDH challenge) and doesn't compute a session key.
5. When $\mathbb{M}$ makes a session key-reveal or ephemeral secret key-reveal query against some session (different from the special session), $\mathbb{S}$ returns to $\mathbb{M}$ a session key or an ephemeral secret key for this session (which was computed previously in Queries 2, 3 or 4). If $\mathbb{M}$ tries to reveal a session key or an ephemeral secret key of the special session, $\mathbb{S}$ declares failure and stops the experiment.
6. When $\mathbb{M}$ makes a corruption on some party $\mathbb{C}$ (different from $\mathbb{A}$ and $\mathbb{B}$), $\mathbb{S}$ returns the secret key of $\mathbb{C}$ as well as ephemeral secret keys of all current AKE sessions executed by $\mathbb{C}$ and gives $\mathbb{M}$ full control over $\mathbb{C}$. If $\mathbb{M}$ tries to corrupt $\mathbb{A}$ or $\mathbb{B}$ (after a special session is selected), $\mathbb{S}$ declares failure.

When $\mathbb{M}$ stops, $\mathbb{S}$ goes over all random oracle queries made by $\mathbb{M}$ and checks (using a DDH oracle DDH) if any of them includes the value of $CDH(X_0, Y_0)$. If "yes", return $CDH(X_0, Y_0)$ to the GDH challenger. If "no", $\mathbb{S}$ declares failure.

Function $\text{HSIM}(Z_1, Z_2, \mathbb{B}, \mathbb{C})$. This function implements a random oracle on valid signatures of the KEA+ protocol. The function proceeds as follows:

– If the value of the function on that input has been previously defined, return it.
– If not defined, go over all the previous calls to $\text{HSPEC}(\cdot)$ and for each previous call of the form $\text{HSPEC}(i, Y, Z, \mathbb{B}', \mathbb{C}') = v$ check if

$$\mathbb{B} = \mathbb{B}', \quad \mathbb{C} = \mathbb{C}', \quad Z = Z_{3-i} \text{ and } \text{DDH}(X_0, Y, Z_i) = 1.$$

  If all these conditions hold, return v.

– If not found, pick a random w from $\{0,1\}^l$, define $\textsc{Hsim}(Z_1, Z_2, \mathbb{B}, \mathbb{C}) = w$ and return w.

Function $\textsc{Hspec}(i, Y, Z, \mathbb{B}, \mathbb{C})$. Informally, $\textsc{Hspec}$ implements a random oracle on signatures which are not known to $\mathbb{S}$. Specifically, the input corresponds to a signature $(Z_1, Z_2, \mathbb{B}, \mathbb{C})$, where $Z_i = CDH(X_0, Y)$ (here X_0 is a part of the GDH challenge) and $Z_{3-i} = Z$. This signature is not known to $\mathbb{S}$ since $\mathbb{S}$ cannot compute $CDH(X_0, Y)$. The function proceeds as follows:

– If the value of the function on that input has been previously defined, return it.
– If not defined, go over all the previous calls to $\textsc{Hsim}(\cdot)$ and for each previous call of the form $\textsc{Hsim}(Z_1, Z_2, \mathbb{B}', \mathbb{C}') = v$ check if

$$\mathbb{B} = \mathbb{B}', \quad \mathbb{C} = \mathbb{C}', \quad Z = Z_{3-i} \text{ and } \mathrm{DDH}(X_0, Y, Z_i) = 1.$$

  If all these conditions hold, return v.
– If the check failed for all the calls, pick a random w from $\{0,1\}^l$, define $\textsc{Hspec}(i, Y, Z, \mathbb{B}, \mathbb{C})$ to be w and return w.

Analysis of $\mathbb{S}$. The the running time of $\mathbb{S}$ is the time needed to run an AKE experiment and $\mathbb{M}$ plus the time needed to handle H-queries. Each call to $\textsc{Hsim}$ or $\textsc{Hspec}$ requires $\mathbb{S}$ to pass over all the previously made queries. Thus, time needed to handle H-queries is proportional to a squared number of queries. Since the number of H-queries is upper-bounded by the running time of $\mathbb{M}$, we can bound the running time of $\mathbb{S}$ by $O(t^2)$, where t is the running time of $\mathbb{M}$.

We are now going to show that if $\mathbb{M}$ doesn't corrupt $\mathbb{A}$ and doesn't reveal a session key or an ephemeral secret key for the special session, then the simulation of an AKE experiment is perfect. That is, the view of $\mathbb{M}$ in the experiment run by $\mathbb{S}$ is identically distributed to the view of $\mathbb{M}$ in an authentic experiment. To be precise, the view of $\mathbb{M}$ consists of public keys of all the parties, secret keys of the corrupted parties, ephemeral public keys of all the sessions, ephemeral secret keys and session keys of the corrupted sessions and of the random oracle's responses.

We start by observing that secret/public key pairs of all honest parties except $\mathbb{A}$ are distributed correctly. A public key of $\mathbb{A}$ is also distributed correctly, however $\mathbb{S}$ doesn't know the secret key for it. By assumption, $\mathbb{M}$ doesn't corrupt $\mathbb{A}$ and thus $\mathbb{M}$ wouldn't notice that. Similarly, ephemeral secret/public values of all sessions except the test session are distributed as in the original protocol. The ephemeral public key Y_0 in the test session is also distributed correctly, although $\mathbb{S}$ doesn't know a secret for it. Again, we assume that $\mathbb{M}$ doesn't corrupt the test session and so $\mathbb{S}$ wouldn't have to reveal it.

The adversary can obtain the random oracle's responses either by querying H directly or by revealing session keys from honest parties. Without loss of generality, we can assume that the adversary queries a random oracle only on tuples of the form $(Z_1, Z_2, \mathbb{B}_1, \mathbb{B}_2)$, where $Z_1, Z_2 \in G$ and B_1 and B_2 are identities of some parties. To ensure that the simulation is perfect, we need to verify that

i) the oracle responses are selected at random and ii) if the same argument is queried several times, the same value is returned.

Recall that $\mathbb{S}$ handles two types of queries differently. Queries of the first type are fully specified 4-tuples and such queries are made both by $\mathbb{M}$ and by honest parties. They are handled by the function Hsim. Queries of the second type are made only by $\mathbb{A}$ and such queries have one of the components unspecified. That is, a value Z_i (for some $i = 1, 2$) is unknown and it is specified by $Y \in G$ such that $Z_i = CDH(X_0, Y)$. These queries are handled by Hspec. Note that distinct Hspec arguments correspond to distinct queries to H.

In our construction of Hsim and Hspec, a new random value of H is chosen every time the argument wasn't found in the record of previous queries. Thus, condition i) is satisfied and we only need to show that by querying the same argument several times, $\mathbb{M}$ always receives the same answers. If the same query is made for the second time either to Hsim or to Hspec, the same answer is returned. The only conflicts can arise if a query previously handled by Hsim is queried again to Hspec or vice versa. That is, Hsim was called on a tuple $(Z_1, Z_2, \mathbb{B}, \mathbb{C})$ and Hspec — on $(i, Y, Z, \mathbb{B}, \mathbb{C})$ where $Z_i = CDH(X_0, Y)$ and $Z_{3-i} = Z$. Note that one can check whether these queries correspond to identical signatures by checking that $Z_{3-i} = Z$ and that $\text{DDH}(X_0, Y, Z_i) = 1$. Whichever of the functions was called first, on the second call (to the other function) $\mathbb{S}$ will go over all previous calls to the first function and do such a check. If a match is found, the previously defined value is returned. This guarantees that condition ii) is also satisfied.

We showed that, provided $\mathbb{M}$ doesn't corrupt $\mathbb{A}$ or the special session, the simulation of the AKE experiment is perfect. Since the party $\mathbb{A}$ and the special session are chosen at random, a test session selected by $\mathbb{M}$ matches the special session with probability $1/nk$ (recall that n is the number of parties in the experiment and k is the maximal number of sessions any party can participate in). In this case, the simulation is perfect since $\mathbb{M}$ doesn't corrupt the test session. We know that a successful adversary must reveal the signature of the test session. Whenever $\mathbb{M}$ wins in the AKE experiment and the test session was guessed correctly, $\mathbb{S}$ reveals the signature of the test session which contains $CDH(X_0, Y_0)$, and therefore wins in the GDH experiment. To summarize the lengthy proof, for any AKE adversary $\mathbb{M}$ running in time t we constructed a GDH solver $\mathbb{S}$ which runs in time $O(t^2)$ such that

$$\mathbf{Adv}^{\mathrm{GDH}}(\mathbb{S}) \geq \frac{1}{nk}\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{KEA+}}(\mathbb{M}).$$

Improving Concrete Security Reduction. The above reduction transforms a time t AKE adversary to a GDH solver which runs in time $O(t^2)$ and makes $O(t^2)$ calls to a DDH oracle, which is fairly inefficient. We observe that given access to a pairing oracle, we can solve the CDH problem in time $O(t \log t)$ by making $O(t)$ calls to a pairing oracle.

The construction of the solver $\mathbb{S}$ remains the same except for the Hsim and Hspec functions. We create an array T and implement Hsim and Hspec as follows:

Function $\textsc{Hsim}(Z_1, Z_2, \mathbb{B}, \mathbb{C})$:

– Compute $\delta = (\mathrm{P}(g, Z_1), \mathrm{P}(g, Z_2), \mathbb{B}, \mathbb{C})$.
– Look up δ in T.
– If T contains a record (δ, v), return v.
– If not, pick w at random, add a record (δ, w) to T and return w.

Function $\textsc{Hspec}(i, Y, Z, \mathbb{B}, \mathbb{C})$.

– Compute $Z'_i = \mathrm{P}(X_0, Y)$, $Z'_{3-i} = \mathrm{P}(g, Z)$ and set $\delta = (Z'_1, Z'_2, \mathbb{B}, \mathbb{C})$.
– Look up δ in T.
– If T contains a record (δ, v), return v.
– If not, pick w at random, add a record (δ, w) to T and return w.

First, note that the queries to Hsim and Hspec which correspond to the same arguments to a random oracle will be mapped to the same values of δ. Thus a random oracle will be perfectly simulated and $\mathbb{S}$ will win the CDH experiment with the same probability as in the original proof.

Second, each call to Hsim or Hspec requires only one oracle call to P. Moreover, if T is implemented as a balanced search tree indexed by values of δ, each search and insert operation in T takes logarithmic time in the size of T. Thus the processing of each call to Hsim or Hspec takes at most $O(\log t)$ time, where t is the maximal running time of $\mathbb{M}$.

For any AKE adversary $\mathbb{M}$ running in time t we have a PDH solver $\mathbb{S}$ which runs in time $O(t \log t)$ and makes $O(t)$ oracle queries such that

$$\mathbf{Adv}^{\mathrm{PDH}}(\mathbb{S}) \geq \frac{1}{nk} \mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{KEA+}}(\mathbb{M}).$$

Weak PFS. We observe that our proof of AKE security can be modified to establish wPFS security of KEA+. Consider the same party $\mathbb{S}$ who runs an AKE experiment with an adversary $\mathbb{M}$. Consider the test session selected by $\mathbb{M}$ and its matching session. By the definition of wPFS, $\mathbb{M}$ did not cancel or modify communications sent between the parties involved in these sessions. The test session (as well as its matching session) must be clean at the time of completion. After the test session and its matching session are completed, $\mathbb{M}$ can corrupt either one of the involved parties but not both of them. Now consider that session, (out of the test session and its matching session), where the executing party can be corrupted and the other party is not corrupted. We observe that with probability $1/nk$ this session matches the special session $(\mathbb{B}, \mathbb{A}, \mathit{role})$, which is randomly selected by $\mathbb{S}$.

Since $\mathbb{S}$ knows the long-term secret key of the party $\mathbb{B}$ executing the special session, $\mathbb{S}$ can handle corruptions of $\mathbb{B}$ which are made after the test session is completed. When $\mathbb{M}$ launches a corruption of $\mathbb{B}$, $\mathbb{S}$ hands to $\mathbb{M}$ the long-term secret key of $\mathbb{B}$ and ephemeral secret keys of all current sessions being executed by $\mathbb{B}$. Since the test session is already completed, $\mathbb{B}$ will know all the ephemeral secret keys for the current session (provided that the test session matches the

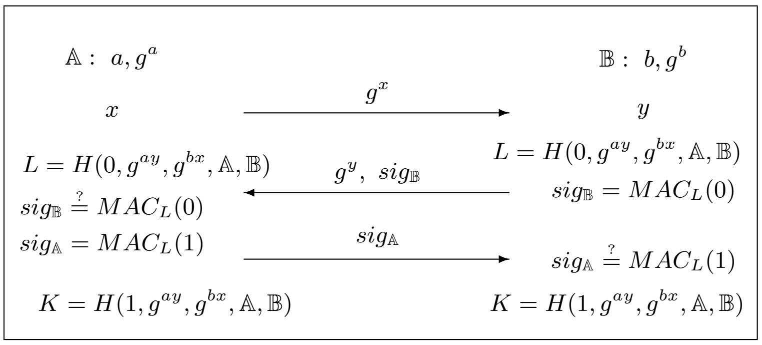
$\mathbb{A}:\ a, g^a$
$\mathbb{B}:\ b, g^b$
x
g^x
y
$L = H(0, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$g^y,\ sig_{\mathbb{B}}$
$L = H(0, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$sig_{\mathbb{B}} = MAC_L(0)$
$sig_{\mathbb{B}} \overset{?}{=} MAC_L(0)$
$sig_{\mathbb{A}} = MAC_L(1)$
$sig_{\mathbb{A}}$
$sig_{\mathbb{A}} \overset{?}{=} MAC_L(1)$
$K = H(1, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$K = H(1, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$

Fig. 3. KEA+C protocol

special session). Therefore, the simulation of an AKE experiment remains perfect and the GDH/PDH solver $\mathbb{S}$ has the same advantage.

KCI Security. The same proof of AKE security can be used to show that KEA+ also satisfies KCI security. The only difference is that now $\mathbb{S}$ has to handle long-term secret key reveals made by $\mathbb{M}$. Since $\mathbb{S}$ knows the long-term secret keys of all the parties other than $\mathbb{A}$, $\mathbb{S}$ can answer all such long-term secret key reveals anytime. We note that in the event that the special session matches the test session, $\mathbb{M}$ is not allowed to reveal the long-term secret key of $\mathbb{A}$. Therefore, in this case the simulation remains perfect and the GDH/PDH solver $\mathbb{S}$ has the same advantage in a CDH experiment.

4 Key Confirmation: KEA+C

Protocol Description. We assume that both parties know each other's registered public keys. Let H be an arbitrary cryptographic hash function and MAC be an arbitrary message authentication code.

The KEA+C protocol is illustrated in Figure 3. First, $\mathbb{A}$ selects a random ephemeral secret key x and sends an ephemeral public key g^x to $\mathbb{B}$. In turn, $\mathbb{B}$ verifies that $g^x \in G$, selects a random ephemeral secret key y and computes a verification key $L = H(0, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$. $\mathbb{B}$ then sends back to $\mathbb{A}$ an ephemeral public key g^y together with a key confirmation value $sig_{\mathbb{B}} = MAC_L(0)$. On receipt of the tuple $(g^y, sig_{\mathbb{B}})$, the party $\mathbb{A}$ first verifies that $g^y \in G$ and if accepted, computes a verification key $L = H(0, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$, checks that $sig_{\mathbb{B}}$ is valid, sends to $\mathbb{B}$ a key confirmation value $sig_{\mathbb{A}}$ and computes a session key $K = H(1, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$. Finally, $\mathbb{B}$ verifies the validity of $sig_{\mathbb{A}}$ and if accepted, computes a session key $K = H(1, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$. The session key K should be used as a shared key between the parties while the confirmation key L as well as all the intermediate information (except possibly ephemeral secret keys) should be erased immediately after completion of a session. We remark that despite the visible similarity, the keys K and L are computationally independent. In a practical implementation, one might alternatively derive them from a 4-tuple

$(g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$ by applying 2 independent hash functions. When a hash function is modeled by a random oracle $H(0, \cdot)$ and $H(1, \cdot)$ are independent random oracles.

Security Analysis. We show that KEA+C has key confirmation, AKE security against a strong adversary, full PFS, KCI security and is also secure in the Universally Composable model as defined by Canetti and Krawczyk [9].

First of all, we observe that repeating the proof of security for KEA+ we obtain the same security guarantees for KEA+C, namely AKE security against a strong adversary, weak PFS and KCI security. Universally Composable security [6, 9] ensures that a key-exchange protocol can securely run concurrently with arbitrary other applications. In fact, UC-security of KEA+C automatically follows from the result of Canetti and Krawczyk [9]. They establish UC security of authenticated key exchange provided that the protocol satisfies AKE security and also enjoys the so-called "ACK property". The latter requires that at the time when the initiator party outputs its session key, the other party's state can be "simulated" given only the session key and public information in the protocol. We observe that Claim 15 in [9] implies that KEA+C has this property, thus establishing UC security of KEA+C. Finally, we observe that the full Perfect Forward Secrecy property follows from UC security.

Acknowledgements

The work for this paper was done while the second author was visiting Microsoft Research. The authors thank Josh Benaloh, Brian LaMacchia, Gideon Yuval and anonymous reviewers for helpful comments and suggestions.

References

1. M. Abdalla, O. Chevassut and D. Pointcheval, *One-Time Verifier-Based Encrypted Key Exchange*, Public Key Cryptography — PKC '05, pp. 47–64, Springer-Verlag, 2005
2. M. Bellare, A. Palacio, *The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols*, Advances in Cryptology — CRYPTO '04, pp. 273–289, Springer-Verlag, 2004
3. M. Bellare, D. Pointcheval, P. Rogaway, *Authenticated Key Exchange Secure Against Dictionary Attacks*, Advances in Cryptology — Eurocrypt '00, pp. 139–155, Springer-Verlag, 2000
4. M. Bellare and P. Rogaway, *Entity Authentication and Key Distribution*, Advances in Cryptology — CRYPTO '93, pp. 110–125, Springer-Verlag, 1993
5. S. Blake-Wilson, D. Johnson, and A. Menezes, *Key Agreement Protocols and their Security Analysis*, 6th IMA International Conference on Cryptography and Coding, LNCS 1355, pp. 30-45, Springer-Verlag, 1997
6. R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science, IEEE Computer Society, 2001

7. K.-K. R. Choo, C. Boyd and Y. Hitchcock, *Examining Indistinguishability-Based Proof Models for Key Establishment Protocols*, to appear in Advances in Cryptology — Asiacrypt '05, Springer-Verlag, 2005
8. I. R. Jeong, J. Katz, D. H. Lee, *One-Round Protocols for Two-Party Authenticated Key Exchange*, ACNS '04, 2004
9. R. Canetti and H. Krawczyk, *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*, Advances in Cryptology — EUROCRYPT '01, pp. 453–474, Springer-Verlag, 2001
10. H. Krawczyk, *SIGMA: The "SIGn-and-MAc" Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols*, Advances in Cryptology — CRYPTO '03, LNCS 2729, pp. 400–425, Springer-Verlag, 2003
11. H. Krawczyk, *HMQV: A High-Performance Secure Diffie-Hellman Protocol*, Advances in Cryptology — CRYPTO '05, LNCS 3621, pp. 546–566, Springer-Verlag, 2005
12. M. Jakobsson and D. Pointcheval, *Mutual Authentication for Low-Power Mobile Devices*, Financial Cryptography '01, pp. 178–195, Springer-Verlag, 2001
13. C. Kudla and K. G. Paterson, *Modular Security Proofs for Key Agreement Protocols*, Advances in Cryptology — ASIACRYPT '05, pp. 549–565, Springer-Verlag, 2005
14. A. Menezes, *Another look at HMQV*, IACR Eprint archive, `http://eprint.iacr.org/2005/205`, 2005
15. NIST, *SKIPJACK and KEA Algorithm Specification*, `http://csrc.nist.gov/encryption/skipjack/skipjack.pdf`, 1998
16. T. Okamoto and D. Pointcheval, *The Gap Problems: A New Class of Problems for the Security of Cryptographic Schemes*, Public Key Cryptology — PKC '01, LNCS 1992, pp. 104–118, Springer-Verlag, 2001
17. V. Shoup, *On Formal Models for Secure Key Exchange*, Theory of Cryptography Library, `http://www.shoup.net/papers/skey.ps`, 1999
18. Y. S. T. Tin, C. Boyd and J. M. González Nieto, *Provably Secure Mobile Key Exchange: Applying the Canetti-Krawczyk Approach*, ACISP '03, pp. 166–179, Springer-Verlag, 2003