Security Analysis of KEA Authenticated Key Exchange

Kristin Lauter
Microsoft Research
One Microsoft Way, Redmond, WA, 98052
klauter@microsoft.com

Anton Mityagin
University of California, San Diego
9500 Gilman Dr., La Jolla, CA, 92093
amityagin@cs.ucsd.edu

August 12, 2005

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. Finally, we show how to add a key confirmation feature to KEA+ (we call the version with key confirmation KEA+C) and discuss 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. A classical example of key exchange is the Diffie-Hellman protocol.

Authenticated key exchange (AKE) not only allows parties to compute a 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 typically use each other's public keys to construct a shared secret.

Natural Solution: Signed Diffie-Hellman. One possible solution to authenticated key exchange is 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 multiplicative group of prime order and denote by g a generator of G. Assume that all the parties have respective secret/public keys for some digital signature scheme SIG and that parties know each other's certified public keys. Denote the signature of a message M under a key of a party $\mathbb{A}$ as $SIG_{\mathbb{A}}(M)$.

The protocol has 2 rounds of communication. First, an initiator $\mathbb{A}$ picks an ephemeral secret key x at random and sends to a responder $\mathbb{B}$ a tuple $\{g^x,\ SIG_{\mathbb{A}}(g^x \| \mathbb{B})\}$. A value g^x is called an ephemeral public key of the initiator. 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 formally analyzed by Shoup [11] 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.

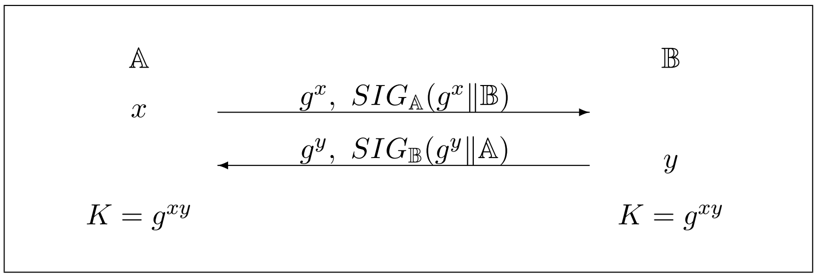

$\mathbb{A}$
$\mathbb{B}$
x
$g^x,\ SIG_{\mathbb{A}}(g^x\|\mathbb{B})$
$g^y,\ SIG_{\mathbb{B}}(g^y\|\mathbb{A})$
y
$K = g^{xy}$
$K = g^{xy}$

Figure 1: Signed Diffie-Hellman authenticated key-exchange

It is worth noting that Signed Diffie-Hellman AKE can be easily 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 public keys are precomputed or if ephemeral keys are stored in insecure storage. Let an adversary $\mathbb{M}$ reveal an ephemeral secret key x used by $\mathbb{A}$ in some session with $\mathbb{B}$. From that point on, $\mathbb{M}$ can impersonate $\mathbb{A}$ to $\mathbb{B}$ by 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 $\mathbb{M}$ 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 precise definitions in Section 2:

1. The main security requirement (we will call it AKE security or Canetti-Krawczyk security) as introduced by Bellare and Rogaway [2] and further refined by Canetti and Krawczyk [6] considers a multi-party experiment with unauthenticated communication channels. 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. After participating in this experiment, an adversary must select an uncorrupted session called a test session. An adversary is then 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 from an 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 [8] shows that no 2-round AKE protocol can achieve perfect forward secrecy. Alternatively, he presents a notion of weak perfect forward secrecy (wPFS). Weak perfect forward secrecy only guarantees security for 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.

The Bellare-Rogaway and Canetti-Krawczyk definitions are rather alike and both involve the above indistinguishability experiment. Besides minor technical details, the main difference is that Canetti-Krawczyk introduces a strong adversary, allows active corruptions and also allows an adversary to continue an experiment after receiving a challenge key.

KEA Protocol. KEA authenticated key exchange [10] 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 valid certified 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. Received ephemeral public keys should be verified for group membership in G. Then each of the parties computes g^{ay} and g^{bx} and computes a session key K by applying a hash function F to $g^{ay} \oplus g^{bx}$.[1] The original description specifies F to be a certain function built on the SKIPJACK block cipher [10]. Figure 2 depicts the protocol.

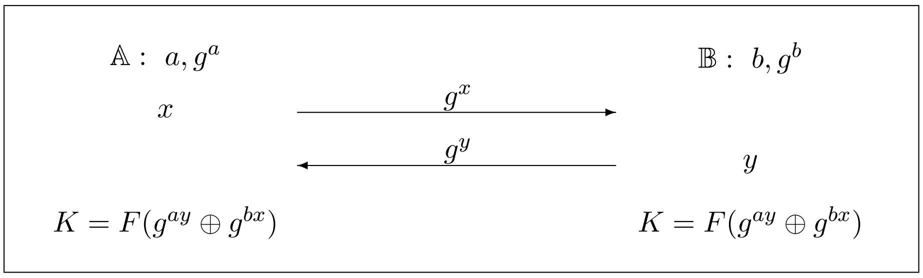


Figure 2: Original KEA protocol

The design of KEA closely resembles Protocol 4 from Blake-Wilson et al. [3]. 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.

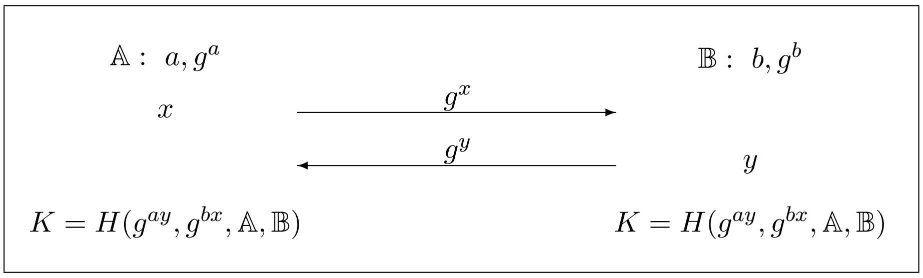
Attacks on KEA. We observe that the security of KEA can be violated if an adversary can register arbitrary public keys. If an adversary $\mathbb{M}$ registers a public key g^a of some honest party $\mathbb{A}$ as its own public key, then $\mathbb{M}$ can intercept a key-exchange session between $\mathbb{A}$ and some other honest party $\mathbb{B}$ and at the same time start a session between $\mathbb{M}$ and $\mathbb{B}$. Now $\mathbb{M}$ forwards the ephemeral public key g^x from $\mathbb{A}$ to $\mathbb{B}$ and the 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, but they participate in two different key-exchange sessions. To succeed in the experiment, $\mathbb{M}$ reveals the session key of one of the sessions and announces the other session as a test session. The demonstrated attack breaks AKE security against a weak adversary (who can only reveal session keys).

Security fix: KEA+. We present a modified version of the KEA protocol, called KEA+, which makes it resistant to the above attacks, and we prove that KEA+ satisfies the strongest possible 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.

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

[1] In the case when G is a subgroup of the multiplicative group of a finite field, the XOR operation is often replaced by modular addition.

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 3 depicts actions performed by the parties.



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

Figure 3: 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 or under a stronger Pairing Diffie-Hellman (PDH) assumption. The former assumption states intractability of the Computational Diffie-Hellman (CDH) problem provided a solver is given access to a Decisional Diffie-Hellman (DDH) oracle, while the latter assumption means hardness of CDH, where a solver is given access to a bilinear pairing oracle. We find this 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.

The reason for having 2 reductions (to GDH and to PDH) lies in the concrete security of the reductions. We present a concrete security analysis and find that a reduction to GDH is fairly inefficient while a reduction to PDH is tight.

Key Confirmation: KEA+C. The 2-round 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 practical applications. Namely, KEA+ doesn't provide assurance that the other party completed a session with the same session key. To address this issue, we add one more round 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 [8], 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 [6] imply that KEA+C satisfies Universally Composable security defined by [4], 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 [2] from 1993. Following work of Bellare, Canetti and Krawczyk [1] and Shoup [11], the current security definition was formulated by Canetti and Krawczyk [6]. We refer a reader to [5] 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 achieve a weaker form of security and only a few of them which are proved to satisfy to the strongest security requirement. AKE protocols proved to achieve the strongest security include SIG-DH from [6], SIGMA [7] and HMQV [8].

We want to compare our KEA+ protocol with a recent HMQV protocol [8], which combines great efficiency with the highest security level. KEA+ and HMQV are proven to satisfy the same security notions, which are AKE security, wPFS and KCI. However, security of HMQV relies on the knowledge of the exponent assumption (KEA1) and doesn't provide a satisfactory concrete security analysis. As noted by Menezes [9], the concrete security reduction of [8] appears to be inefficient. Our security proof doesn't employ the KEA1 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 exponentiations in HMQV uses half-size exponents).

We concentrate on AKE protocols in the 2-party public key scenario and we don't consider related scenarios for authenticated key exchange such as identity-based AKE, password-based AKE and three-party or multi-party AKE.

2 Definitions

Mathematical Group. 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 a group operation in a multiplicative manner.

Computational Diffie-Hellman (CDH). The Computational Diffie-Hellman function $CDH(\cdot,\cdot)$ takes as input 2 elements of G: $X = g^x$ and $Y = g^y$. The value $CDH(X,Y)$ is defined as g^{xy}. The Computational Diffie-Hellman problem is defined by an experiment between a challenger and an adversary $\mathbb{M}$, where a challenger picks random elements $X, Y \in G$, hands them to $\mathbb{M}$, who in turn returns a value $Z \in G$ (which is the adversary's guess for $CDH(X,Y)$). The advantage of $\mathbb{M}$ in this experiment is defined to be

$$\mathbf{Adv}^{\mathrm{CDH}}(\mathbb{M}) \;=\; Pr[Z = CDH(X,Y)],$$

where the probability is taken over a random choice of X, Y and over random coins of $\mathbb{M}$.

Gap Diffie-Hellman (GDH). A Decisional Diffie-Hellman (DDH) Oracle O for a group G and generator g is an oracle which takes as input a triple $(g^x, g^y, Z) \in G^3$. O outputs 1 if $Z = g^{xy}$ and 0 otherwise. The Gap-Diffie-Hellman problem is defined by the CDH experiment where the solver algorithm is additionally given access to a DDH oracle O. The advantage of such a GDH solver $\mathbb{M}$, denoted as $\mathbf{Adv}^{\mathrm{GDH}}(\mathbb{M})$, is $\mathbb{M}$'s winning probability in a CDH experiment. 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.

Pairing Diffie-Hellman (PDH). We define Pairing Diffie-Hellman assumption in a similar manner to GDH. Let G' be another mathematical group of order q with efficiently computable group operation. We will say that a function $e : G \times G \to G'$ is a bilinear pairing if it is non-degenerate and if for any $g^a, g^b \in G$ the following holds:

$$e(g^a, g^b) = e(g,g)^{ab}.$$

A pairing oracle P associated with some pairing function e and some group G' takes two elements $X, Y \in G$ and returns $e(X,Y)$. The Pairing Diffie-Hellman problem is defined by a CDH experiment, where the solver algorithm is additionally given access to some pairing oracle P. The advantage $\mathbf{Adv}^{\mathrm{GDH}}(\mathbb{M})$ of a PDH solver $\mathbb{M}$ is the probability of $\mathbb{M}$ winning in this CDH experiment. We say that G satisfies the PDH assumption if no feasible adversary exists to solve the CDH problem provided with an arbitrary PDH-oracle. [2]

We observe that the PDH assumption is stronger than GDH. One can use a PDH oracle P to solve the DDH problem as follows. Given $X = g^a, Y = g^b, Z = g^c \in G$ one needs to check if $\mathrm{P}(X, Y) = \mathrm{P}(g, Z)$. The equality holds iff $e(g, g)^{ab} = e(g, g)^c$, and since $e(g, g)$ is an element of order q (this follows from the pairing being non-degenerate), this is equivalent to $ab = c$ modulo q.

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 (as we showed above). It is interesting that despite all the similarities between GDH and PDH, GDH is widely adopted while we were unable to find any PDH-like assumptions mentioned in the literature.

Functions Which Take Multiple Arguments. 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. Sometimes we use parties' identities as some of the inputs: we assume that identities are represented by binary strings. In the security analysis we model a hash function as a random oracle and treat a hash function which takes multiple arguments as a random oracle which takes multiple arguments.

AKE Security. The security of an AKE protocol is defined by a multi-party experiment. We give a brief description of the model here and refer the reader to Canetti and Krawczyk [6] for a more comprehensive description.

The AKE experiment involves multiple parties connected via an unauthenticated network and an adversary $\mathbb{M}$. All the parties are interactive probabilistic Turing machines and the parties represent participants of a key-exchange system. An adversary is responsible for activating the parties ($\mathbb{M}$ selects parties to execute key-exchange sessions and an order of the sessions) and as well it can corrupt some of the parties. An adversary has full control over communication and he can delay/cancel/modify any message.

There is a special party, $\mathbb{CA}$, called the certificate authority, who certifies the public keys of the parties. Presented with a public key and an identity of a party, $\mathbb{CA}$ issues a certificate on the key and the identity. From that moment on, that key is considered to be registered. We model a $\mathbb{CA}$ as a trusted directory. Any party at any time can ask the $\mathbb{CA}$ for a list of all parties with their certified public keys; as well, any party can ask the $\mathbb{CA}$ to certify any given key. We assume that $\mathbb{CA}$ registers arbitrary keys (even those matching keys of other parties) and that no party can have more than one certified public key.

An adversary can activate a key exchange session between any two parties. To start a session, an adversary activates an honest party and specifies that party's role in the exchange (which can be either an initiator or a 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 will sometimes omit a communication Comm from a description of a session if we only want to indicate the parties executing a session but not messages they sent to each other. We stress that an AKE session is executed by a single party: since communication is controlled by an adversary, a party executing a session cannot know for sure whom it is talking to. We denote a matching AKE session by a session which is supposed to be executed by the other party: for example, a session $(\mathbb{A}, \mathbb{B}, \mathit{initiator}, \mathit{Comm})$ matches $(\mathbb{B}, \mathbb{A}, \mathit{responder}, \mathit{Comm})$

[2]We note that a valid solver should solve CDH given an arbitrary group G' and an arbitrary pairing oracle. For example, for every group there exists some PDH oracle which allows computation of discrete logarithms and thus solves the CDH problem: consider the pairing function $e(g^a, g^b) = ab \bmod q$.

and vice versa. We say that a party completes the session when it has received the last message from the other party and computes a session key.

An adversary can corrupt honest parties as well as reveal session information. When an adversary makes a corruption of some party (in the literature this is 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 (this is 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 make ephemeral secret key reveals of any sessions. He can reveal ephemeral secret keys of a session both if the session is being executed or if it is already completed.

We say that a completed session is "clean" if this session as well as its matching session is not corrupted and if none of the participating parties is corrupted. A session can be clean even if the adversary actively participated in the session and modified the communications. For example, if an adversary starts a session with $\mathbb{A}$ pretending to be $\mathbb{B}$, this session is clean as long as the adversary doesn't reveal $\mathbb{A}$'s secret keys nor the session key (computed by $\mathbb{A}$). 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 $b \in \{0,1\}$; 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 hands the challenge K_C to the adversary who is supposed to output a bit b' (which is the adversary's guess for b). After receiving a challenge key, the adversary continues the experiment but he is not allowed to corrupt the test sessions nor any of the parties involved in the test session. The experiment ends when an adversary outputs a guess bit b'.

The above experiment run by a challenger is called an AKE-experiment. In turn, the adversary participating in an AKE-experiment is called an AKE-adversary. The advantage of adversary $\mathbb{M}$ in the above AKE-experiment against AKE protocol Π is defined as

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

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. Now we allow an adversary to select a test session which was clean at the time of completion of the session, but give the adversary the power to corrupt at most one of the parties after the session is completed. As in the original AKE experiment, an adversary must distinguish between the session key of the test session and a random key.

Krawczyk [8] observed that no 2-round AKE protocol can achieve full PFS. To address forward secrecy of 2-round protocols, they suggest a relaxed notion, called weak PFS (wPFS). Weak PFS only guarantees security of those AKE sessions executed without active adversarial intrusion. We defined full PFS by allowing an adversary to select a test session among some extra set of sessions. We define weak PFS by limiting this set of sessions to those executed without active adversarial intrusion. That is, an adversary is only allowed to forward communications in the test session and its matching session and not allowed to cancel or modify them.

Security against Key Compromise Impersonation (KCI) Attacks. 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 Attacks on KEA

We refer the reader to the introduction for a description of the original KEA protocol. 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 $\mathbb{A}$. We depict this attack in Figure 4. Since $\mathbb{M}$ has the same public key as $\mathbb{A}$,

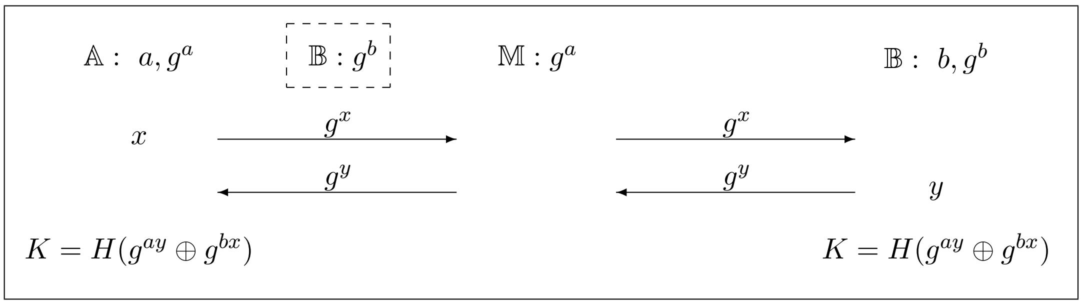


Figure 4: Attack on KEA protocol

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 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 compute a session key as $H(g^{ayk} \oplus g^{bx})$.

4 Security of KEA+

KEA+ Protocol. We assume that prior to the execution of the protocol, both participants know each other's certified valid public keys. Valid means that they are elements of a group G.

Let H be an arbitrary cryptographic hash function. In the security analysis we model H by a random oracle.

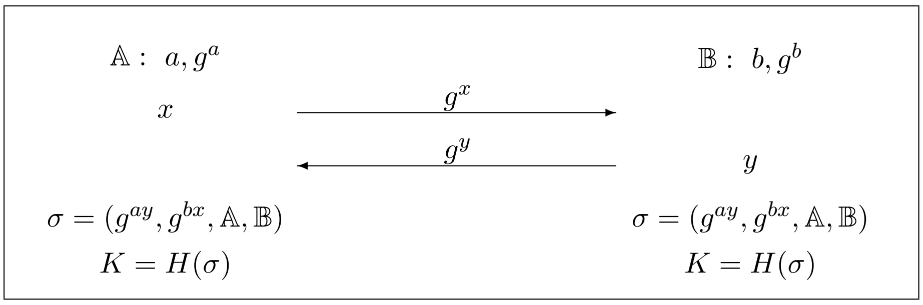

$\mathbb{A}:\ a, g^a$
$\mathbb{B}:\ b, g^b$
x
g^x
g^y
y
$\sigma = (g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$K = H(\sigma)$
$\sigma = (g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$K = H(\sigma)$

Figure 5: KEA+ protocol

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 $H(g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$. When a party completes an AKE session and computes a session key, it should erase all the intermediate information except possible an ephemeral secret key. Figure 5 depicts actions performed by the parties. We denote by a 4-tuple $\sigma = (g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$ a "signature" of a key exchange session. 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.

AKE SECURITY OF KEA+. The rest of the section is devoted to the proof of 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 GDH or PDH assumptions in a group G.

REDUCTION TO A FORGING ATTACK. Assume by contradiction that there exists some efficient distinguishing adversary $\mathbb{M}$ against the KEA+ protocol. Let $(\mathbb{A}, \mathbb{B}, \textit{initiator}, X, Y)$ be a test session in some AKE experiment. Let A be the public key of $\mathbb{A}$ and B be the public key of $\mathbb{B}$. We observe that since the session key of a test session is computed as

$$K = H(CDH(A, Y), CDH(B, X), \mathbb{A}, \mathbb{B}),$$

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 test signature
$$\sigma = (CDH(A, Y), CDH(B, X), \mathbb{A}, \mathbb{B}).$$
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.

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 show that a key-replication attack is impossible. In that case, if an adversary finds some session with the same signature σ as in 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.

It is only 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 will 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 which includes 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 (by picking a random long-term secret key b and computing a long-term public key $B = g^b$). $\mathbb{S}$ simulates a random oracle H that records all $\mathbb{M}'s$ queries. When $\mathbb{A}$ and $\mathbb{B}$ execute a test session, $\mathbb{A}$ picks a random ephemeral secret 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 $\mathbb{M}$ will win with the same probability. As we justified earlier, if $\mathbb{M}$ wins, he should query H a signature $\sigma = (CDH(X_0, Y_0), g^{bx}, \mathbb{A}, \mathbb{B})$. This happens with probability at least $\mathbf{Adv}^{\mathrm{AKE}}_{\mathrm{KEA+}}(\mathbb{M})$. Note that in this case $\mathbb{M}$ computes CDH of the challenge values! If $\mathbb{M}$ makes such a query, $\mathbb{S}$ can identify it using a DDH oracle. That is, given an AKE adversary $\mathbb{M}$ we constructed a GDH adversary $\mathbb{S}$ which runs in the same time as $\mathbb{M}$ such that

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

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 suggest handling this case by picking a session key at random without computing a signature. Then $\mathbb{S}$ uses a 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 O. $\mathbb{S}$ 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 $\textsc{Hsim}(v)$. The procedure $\textsc{Hsim}(\cdot)$ 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 $K = \textsc{Hsim}(Y^b, C^x, \mathbb{B}, \mathbb{C})$. 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 $K = \textsc{Hsim}(C^y, X^b, \mathbb{C}, \mathbb{B})$.
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 $\textsc{Hspec}(1, Y, C^x, \mathbb{A}, \mathbb{C})$, see the description of the procedure $\textsc{Hspec}$ 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 $K = \textsc{Hspec}(2, X, C^y, \mathbb{C}, \mathbb{A})$.
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 O) 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 $\textsc{Hsim}(Z_1, Z_2, \mathbb{B}, \mathbb{C})$. 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 $\textsc{Hspec}(\cdot)$ and for each previous call of the form $\textsc{Hspec}(i, Y, Z, \mathbb{B}', \mathbb{C}') = v$ check if

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

  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 HSIM$(\cdot)$ and for each previous call of the form HSIM$(Z_1, Z_2, \mathbb{B}', \mathbb{C}') = v$ check if

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

  If all these conditions hold, return v.

- If the check failed for all the calls, pick a random w from $\{0,1\}^l$, define 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 HSIM or 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 $\mathrm{O}(X_0, Y, Z_i) = 1$. 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 maps 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 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 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 $(\mathbb{B}, \mathbb{A}, role)$, 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 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.

5 Key Confirmation: KEA+C

Protocol Description. We assume that both parties know each other's certified valid public keys. Valid means that they are elements of the group G. Let H be an arbitrary cryptographic hash function and MAC be an arbitrary message authentication code.

The KEA+C protocol is illustrated in Figure 6. 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, 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 [6].

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 [4, 6] 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 [6]. They establish UC security of authenticated

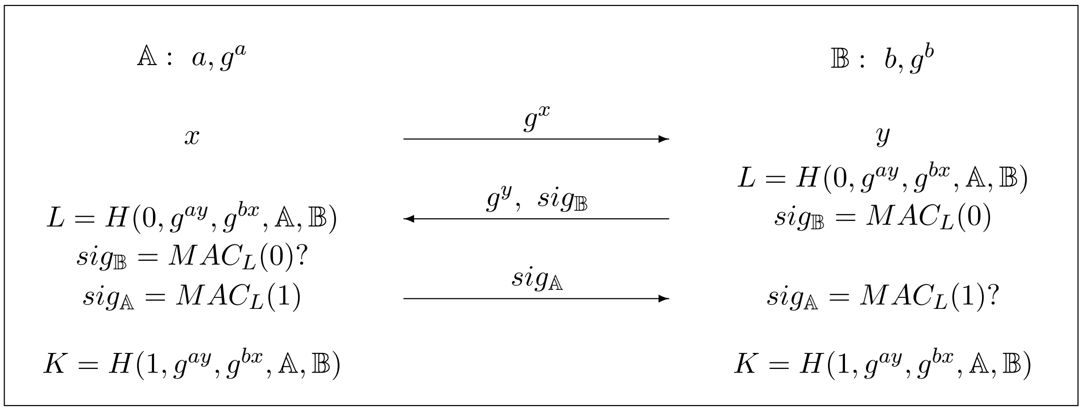

$\mathbb{A}:\ a, g^a$
$\mathbb{B}:\ b, g^b$
x
g^x
y
$L = H(0, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$sig_{\mathbb{B}} = MAC_L(0)$
$g^y,\ sig_{\mathbb{B}}$
$L = H(0, g^{ay}, g^{bx}, \mathbb{A}, \mathbb{B})$
$sig_{\mathbb{B}} = MAC_L(0)?$
$sig_{\mathbb{A}} = MAC_L(1)$
$sig_{\mathbb{A}}$
$sig_{\mathbb{A}}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

Figure 6: KEA+C protocol

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 [6] 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

This work was done while the second author was visiting Microsoft Research. The authors want to thank Josh Benaloh, Brian LaMacchia and Gideon Yuval for helpful discussions and comments.

References

[1] M. Bellare, R. Canetti and H. Krawczyk, *A modular approach to the design and analysis of authentication and key exchange protocols*, STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM Press, 1998

[2] M. Bellare and P. Rogaway, *Entity Authentication and Key Distribution*, Advances in Cryptology - CRYPTO'93, pp. 110-125, Springer-Verlag, 1993

[3] 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. 3045, Springer-Verlag, 1997

[4] 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

[5] 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

[6] R. Canetti and H. Krawczyk, *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*, EUROCRYPT'01 Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques, pp. 453–474, Springer-Verlag, 2001

[7] 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

[8] H. Krawczyk, *HMQV: A High-Performance Secure Diffie-Hellman Protocol*, to appear at CRYPTO'05, 2005

[9] A. Menezes, *Another look at HMQV*, IACR Eprint archive, `http://eprint.iacr.org/2005/205`, 2005

[10] NIST, *SKIPJACK and KEA Algorithm Specification*, 1998, `http://csrc.nist.gov/encryption/skipjack/skipjack.pdf`.

[11] V. Shoup, *On Formal Models for Secure Key Exchange*, Theory of Cryptography Library, 1999, `http://www.shoup.net/papers/skey.ps`