An extended abstract of this paper appears in the proceedings of CRYPTO'19. This is the full version.

Highly Efficient Key Exchange Protocols with Optimal Tightness

Enabling real-world deployments with theoretically sound parameters

Katriel Cohn-Gordon[1], Cas Cremers[2], Kristian Gjøsteen[3], Håkon Jacobsen[4], and Tibor Jager*[5]

[1]Independent Scholar, me@katriel.co.uk
[2]CISPA Helmholtz Center for Information Security, cremers@cispa.saarland
[3]NTNU - Norwegian University of Science and Technology, Trondheim, Norway, kristian.gjosteen@ntnu.no
[4]McMaster University, Hamilton, Canada jacobseh@mcmaster.ca
[5]Paderborn University, Paderborn, Germany, tibor.jager@upb.de

August 16, 2019

Abstract

In this paper we give nearly tight reductions for modern implicitly authenticated Diffie-Hellman protocols in the style of the Signal and Noise protocols, which are extremely simple and efficient. Unlike previous approaches, the combination of nearly tight proofs and efficient protocols enables the first real-world instantiations for which the parameters can be chosen in a theoretically sound manner, i.e., according to the bounds of the reductions. Specifically, our reductions have a security loss which is only *linear* in the number of users μ and *constant* in the number of sessions per user ℓ. This is much better than most other key exchange proofs which are typically *quadratic* in the product $\mu\ell$. Combined with the simplicity of our protocols, this implies that our protocols are more efficient than the state of the art when soundly instantiated.

We also prove that our security proofs are optimal: a linear loss in the number of users is unavoidable for our protocols for a large and natural class of reductions.

*Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823, and the Deutsche Forschungsgemeinschaft (DFG), project number 265919409.

Contents

1 Introduction

Securely deploying a key exchange protocol requires implementors to carefully choose concrete values for several parameters, such as group and key sizes. Here we abstract all these values into a single security parameter n. The question then becomes: how should n be selected? An answer is to select it based on formal reductionist arguments in the style of concrete security [8]. These arguments relate n as the security parameter of the protocol to the security parameter $f(n)$ of an assumed hard problem, such that breaking the protocol with parameter n would lead to an attack on the hard problem with parameter $f(n)$. We say a protocol is deployed in a *theoretically sound* way if n is selected such that the underlying problem is "hard enough" with parameter $f(n)$.

Unfortunately, the parameters of most key exchange protocols deployed today are actually *not* chosen in a theoretically sound way. This means that the formal security arguments are in reality vacuous since $f(n)$ is too small for the underlying problem to be hard. For example, existing security proofs for TLS [13, 24, 29] have a security loss which is quadratic in the total number of sessions, but the parameters chosen in practice do not account for this. If one aims for "128-bit security", and assumes 2^{30} users and up to 2^{30} sessions per user (very plausible for TLS), then a theoretically sound choice of parameters would have to provide at least "248-bit security". One particular consequence is that the algebraic groups used for the Diffie-Hellman (DH) key exchange would have to be of order $|\mathbb{G}| \approx 2^{496}$ instead of the common 128-bit-secure choice of $|\mathbb{G}| \approx 2^{256}$ used today. But since larger parameters typically lead to worse performance this is not done in practice. Thus, for TLS as actually deployed, the proofs do not really provide a meaningful security guarantee since they relate the hardness of breaking TLS to a DH instance which is too easy to solve.

It would be desirable if protocols could be instantiated in a theoretically sound way without sacrificing efficiency. This has led to the study of so-called *tight security*, in which one aims to construct proofs such that the gap between n and $f(n)$ is as small as possible. While there have been several recent advances in this field [3, 21], typically they trade tighter proofs for the use of more complex primitives and constructions—which themselves require more or larger keys. This leads to the perhaps counter-intuitive observation that the resulting protocols have a tighter security proof, but are substantially less efficient in practice. For example, the recent protocol of Gjøsteen and Jager [21] has a constant security loss, meaning that an attack on their protocol leads to an attack on decisional DH with essentially the same parameters. However, it is a signed DH protocol, and thus must be instantiated with a tightly-secure signature scheme. The solution used by Gjøsteen and Jager [21] requires a total of 17 exponentiations which can negate the efficiency savings from using a smaller group. In some sense they overshoot their target: they achieve tightness without reaching the actual goal of *efficient* theoretically sound deployment in practice.

In this work we instead aim between these two extremes—real-world protocols having very non-tight proofs on the one hand, and the more theoretical protocols having fully tight proofs on the other—and focus instead on the actual end-

goal of achieving efficient theoretically sound deployments in practice. Our constructions fall into the class of implicitly authenticated DH protocols, which often are more efficient than signed DH variants, and can additionally offer various forms of deniability. Implicitly authenticated key exchange protocols have been studied extensively in the literature, and in the past few years have also started to see deployments in the real world. Perhaps the most well-known example is the Signal protocol [40], which encrypts messages for WhatsApp's 1.5 billion users. Another example is the Noise protocol framework [38], whose so-called IK pattern powers the new Linux kernel VPN Wireguard [18]. Similar protocols in the literature include KEA+ [32] and UM [26].

We will give a security proof for three simple instances of this class, very close to Signal's basic design. In and of itself this isn't particularly noteworthy. What *is* noteworthy, however, is the *tightness* of these proofs. Unlike any other proofs for protocols as simple and efficient as ours, our proofs only incur a security loss which is *linear* in the number of users μ and *constant* in the number of sessions per user ℓ. This is in stark contrast to most other key exchange proofs that are typically *quadratic* in at least one of these parameters, and most of the time quadratic even in their product $\mu\ell$.

Our contributions. Our contributions revolve around three protocols which all aim for high practical efficiency when instantiated with theoretically sound parameters. The first protocol, which we call Π, is a simple and clean implicitly authenticated DH protocol very close to Signal, Noise-KK, KEA+ and UM, and provides weak forward secrecy. In protocol Π users exchange a single group element and perform four group exponentiations to establish a session key. Protocol Π—specified precisely in Section 4—aims for maximal efficiency under the strong DH assumption.

The other two protocols, which can be seen as variants of protocol Π, are designed to avoid the strong DH assumption of Π. The first protocol, which we call Π_{Twin}, adapts the "twinning" technique of Cash et al. [15] to protocol Π, and needs four more exponentiations. The second, which we call Π_{Com}, additionally adapts the "commitment" technique of Gjøsteen and Jager [21], and only needs two more exponentiations than protocol Π. On the other hand, it requires one more round of communication. Both Π_{Twin} and Π_{Com} are slightly more costly than protocol Π, but in return require only the standard CDH and DDH assumptions.

Common to all our protocols is that they are simple and conventional, with no heavyweight cryptographic machinery. They exchange ephemeral keys and derive a session key from the combination of static-ephemeral, ephemeral-static and ephemeral-ephemeral DH values via a hash function H. In our proofs H will be a random oracle.

Our first core contribution is thus to give new reductions for all these protocols with a linear loss $L = O(\mu)$ in the random oracle model. This is better than almost all known AKE protocols. As we will see, even though the loss is not constant, our protocols are so efficient that they perform better than both fully-

tight protocols as well as the most efficient non-tight AKEs[1]. In contrast to previous works, our proofs enable theoretically sound deployment of conventional protocols while maintaining high efficiency.

Our second core contribution is to show that the $O(\mu)$ tightness loss is essentially optimal for the protocols considered in this paper, at least for "simple" reductions. A "simple" reduction runs a *single* copy of the adversary only *once*. To the best of our knowledge, all known security reductions for AKE protocols are either of this type or use the forking lemma (which of necessity leads to a non-tight proof). Hence, to give a tighter security proof, one would have to develop a completely new approach to prove security.

The lower-bound proof will be based on the *meta-reduction* techniques described by Bader et al. [4]. However, these techniques are only able to handle tight reductions from *non-interactive* assumptions, while our first protocol is based on the *interactive* strong DH assumption. Therefore we develop a new variant of the approach, which makes it possible to also handle the strong DH assumption.

Finally, we prove that our protocols can be enhanced to also provide explicit entity authentication by adding key-confirmation messages, while still providing tight security guarantees. To do so, we generalise a theorem of Yang [43] in two ways: we apply it to N-message protocols for $N > 2$, and we give a tight reduction to the multi-user versions of the underlying primitives.

To summarise:

1. We give three protocols with linear-loss security reductions, making them faster than both fully-tight protocols and the most efficient non-tight ones when instantiated in a theoretically sound manner for reasonable numbers of users and sessions.

2. We prove optimality of linear loss for our protocols under "simple" reductions.

3. We tightly extend our protocols with key confirmation messages to provide explicit entity authentication.

Related work. We briefly touch upon some other protocols with non-quadratic security loss. KEA+ [32] achieves $L = O(\mu\ell)$ under the Gap-DH assumption, and where the reduction for pairing-friendly curves takes $O(t \log t)$ time. However, for non-pairing-friendly curves the reduction takes $O(t^2)$ time. Moreover, KEA+ also does not achieve weak forward secrecy in a modern model: only one side's long term key can be corrupted.

The first AKE protocols with L *independent* of μ and ℓ were described by Bader et al. [3] at TCC 2015. They describe two protocols, one with constant security loss $L = O(1)$ and another with loss $L = O(\kappa)$ linear in the security parameter. Both protocols make use of rather heavy cryptographic building

[1] When instantiated with theoretically sound parameters under reasonable assumptions on μ and ℓ in modern deployment settings.

blocks, such as tree-based signature schemes, Groth-Sahai proofs [22], and cryptographic pairings, and are therefore not very efficient.

As already mentioned, Gjøsteen and Jager [21] recently described a more practical protocol, which essentially is a three-message variant of "signed Diffie-Hellman". Even though their protocol uses a rather complex signature scheme to achieve tightness (a single key exchange requires 17 exponentiations and the exchange of in total 16 group elements/exponents), when instantiated with theoretically sound parameters it turns out to be more efficient than even plain signed DH with ECDSA, at least for large-scale deployments. Unlike [3], the security analysis in [21] is in the random oracle model [10] since the paper aims at maximal practical efficiency.

2 Background

In this section we recap some background and standard definitions. Let $\mathbb{G}$ be a cyclic group of prime order p with generator g.

Diffie-Hellman Problems. The computational and decisional Diffie-Hellman problems are natural problems related to breaking the Diffie-Hellman protocol.

Definition 1. Consider the following experiment involving an adversary $\mathcal{A}$. The experiment samples $x, y \xleftarrow{\$} \mathbb{Z}_p$ and starts $\mathcal{A}(g^x, g^y)$. The advantage of $\mathcal{A}$ in solving the *computational Diffie-Hellman* problem is defined as

$$\mathrm{Adv}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{A}) := \Pr\left[\mathcal{A}(g^x, g^y) = g^{xy}\right]$$

Definition 2. Consider the following experiment involving an adversary $\mathcal{A}$. The experiment samples $x, y, z \xleftarrow{\$} \mathbb{Z}_p$ and tosses a coin $b \xleftarrow{\$} \{0,1\}$. If $b = 1$ then it sets $Z := g^{xy}$, while if $b = 0$ then it sets $Z = g^z$. We define the advantage of $\mathcal{A}$ in solving the *decisional Diffie-Hellman* problem as

$$\mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{A}) := \left|\Pr\left[\mathcal{A}(g^x, g^y, Z) = b\right] - 1/2\right|$$

Let $\mathsf{DDH}(g^x, g^y, g^z)$ be an oracle that returns 1 if and only if $xy = z$. The *gap* Diffie-Hellman problem asks to solve the computational Diffie-Hellman problem, given access to the oracle $\mathsf{DDH}(\cdot,\cdot,\cdot)$. The *strong* Diffie-Hellman problem is related to the gap Diffie-Hellman problem, except that the adversary now gets a less capable oracle where the first input is fixed, i.e., $\mathsf{stDH}_x(\cdot,\cdot) = \mathsf{DDH}(g^x,\cdot,\cdot)$.

Definition 3. Consider the following experiment involving an adversary $\mathcal{A}$. The experiment samples $x, y \xleftarrow{\$} \mathbb{Z}_p$ and starts $\mathcal{A}^{\mathsf{stDH}_x(\cdot,\cdot)}(g^x, g^y)$. The advantage of $\mathcal{A}$ in solving the *strong Diffie-Hellman* problem is defined as

$$\mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathsf{stDH}_x(\cdot,\cdot)}(g^x, g^y) = g^{xy}\right].$$

One may wonder to which extent the number of oracle queries to the strong DH oracle affects the concrete security of this assumption. That is, how does the security of strong DH degrade with the number of queries to the stDH oracle? We are not aware of any concrete attacks that exploit the oracle to solve the CDH problem more efficiently than other algorithms for CDH. In particular, in many elliptic curves with practical bilinear pairings it is reasonable to assume hardness of CDH, even though the bilinear pairing is a much stronger tool than a strong DH oracle.

A crucial technique in any tight proof utilizing one of the Diffie-Hellman problems is rerandomisation. With this technique a single Diffie-Hellman problem instance is turned into many in such a way that an answer to any one of them can be turned into an answer to the original instance (see e.g., [7, Lemma 1] for how rerandomisation can be used with the DDH problem, and [21, Thm. 1] for the CDH problem). We will apply rerandomisation many times in our proofs.

The Strong Twin Diffie-Hellman Problem. The strong twin Diffie-Hellman problem was introduced by Cash, Kiltz, and Shoup [15] at EUROCRYPT 2008. It is closely related to the standard computational Diffie-Hellman (CDH) problem, except that it "twins" certain group elements, in order to enable an efficient "trapdoor-DDH" test that makes it possible to simulate a strong-CDH oracle. This makes it possible to show that the twin-DH problem is *equivalent* to the standard CDH problem. Let $\mathsf{twinDH}_{x_0,x_1}(Y, Z_0, Z_1)$ be an oracle which returns 1 if and only if $\mathsf{DDH}(g^{x_0}, Y, Z_0) = 1$ *and* $\mathsf{DDH}(g^{x_1}, Y, Z_1) = 1$.

Definition 4. Consider the following experiment involving an adversary $\mathcal{A}$. The experiment samples $x_0, x_1, y \xleftarrow{\$} \mathbb{Z}_p$ and starts $\mathcal{A}^{\mathsf{twinDH}_{x_0,x_1}(\cdot,\cdot,\cdot)}(g^{x_0}, g^{x_1}, g^y)$. The advantage of $\mathcal{A}$ in solving the *strong twin Diffie-Hellman* problem is defined as

$$\mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathsf{twinDH}_{x_0,x_1}(\cdot,\cdot,\cdot)}(g^{x_0}, g^{x_1}, g^y) = (g^{x_0 y}, g^{x_1 y})\right]$$

The following theorem was proven by Cash, Kiltz, and Shoup [15, Theorem 3].

Theorem 1. *Let $\mathcal{A}$ be a strong twin DH adversary that makes at most Q queries to oracle $\mathcal{O}$ and runs in time $t_{\mathcal{A}}$. Then one can construct a DH adversary $\mathcal{B}$ that runs in time $t_{\mathcal{A}} \approx t_{\mathcal{B}}$ such that*

$$\mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{B}) + Q/p.$$

3 AKE Security Model

In this section we define our game-based key exchange security model. It is based on the real-or-random ("RoR") security definition of Abdalla, Fouque, and Pointcheval [2], and incorporates the extension of Abdalla, Benhamouda, and MacKenzie [1] to capture forward secrecy. The central feature of the RoR-model is that the adversary can make *many* Test-queries, and that all queries are answered with a "real" or "random" key based on the *same* random bit b.

We prefer to work in the RoR-model because it automatically lends itself to tight composition with protocols that use the session keys of the key exchange protocol. For security models where there is only a single Test-query, or where each Test-query is answered based on an individual random bit [3, 21], such a composition is not automatically tight.

Although we mainly consider key exchange protocols with *implicit* authentication in this paper, we show in Section 8 how they can easily be upgraded to also have *explicit* authentication by adding key-confirmation messages to the protocols. Again, the advantage of working in the RoR-model is that it allows us to do this transformation tightly.

Execution Environment. We consider μ parties $1, \ldots, \mu$. Each party i is represented by a set of ℓ oracles, $\{\pi_i^1, \ldots, \pi_i^\ell\}$, where each oracle corresponds to a session, i.e., a single execution of a protocol role, and where $\ell \in \mathbb{N}$ is the maximum number of protocol sessions per party. Each oracle is equipped with a randomness tape containing random bits, but is otherwise deterministic. Each oracle π_i^s has access to the long-term key pair (sk_i, pk_i) of party i and to the public keys of all other parties, and maintains a list of internal state variables that are described in the following:

- Pid_i^s ("peer id") stores the identity of the intended communication partner.
- $\Psi_i^s \in \{\emptyset, \texttt{accept}, \texttt{reject}\}$ indicates whether oracle π_i^s has successfully completed the protocol execution and "accepted" the resulting key.
- k_i^s stores the session key computed by π_i^s.
- sent_i^s contains the list of messages sent by π_i^s in chronological order.
- recv_i^s contains the list of messages received by π_i^s in chronological order.
- $\mathsf{role}_i^s \in \{\emptyset, \texttt{init}, \texttt{resp}\}$ indicates π_i^s's role during the protocol execution.

For each oracle π_i^s these variables are all initialized to the empty string $\emptyset$. The computed session key is assigned to the variable k_i^s if and only if π_i^s reaches the `accept` state, that is, we have $k_i^s \neq \emptyset \iff \Psi_i^s = \texttt{accept}$.

Partnering. To define when two oracles are supposed to derive the same session key we use a variant of matching conversations. In addition to agreement on their message transcripts, they should also agree upon each other's identities and have compatible roles (one being the initiator the other the responder). We remark that our protocol messages consist only of group elements and deterministic functions of them. This means that they are not vulnerable to the "no-match" attacks of Li and Schäge [34].

Definition 5 (Origin-oracle)**.** An oracle π_j^t is an *origin-oracle* for an oracle π_i^s if $\Psi_j^t \neq \emptyset$, $\Psi_i^s = \texttt{accept}$, and the messages sent by π_j^t equal the messages received by π_i^s, i.e., if $\mathsf{sent}_j^t = \mathsf{recv}_i^s$.

Definition 6 (Partner oracles)**.** We say that two oracles π_i^s and π_j^t are *partners* if (1) each is an origin-oracle for the other; (2) each one's identity is the other one's peer identity, i.e., $\mathsf{Pid}_i^s = j$ and $\mathsf{Pid}_j^t = i$; and (3) they do not have the same role, i.e., $\mathsf{role}_i^s \neq \mathsf{role}_j^t$.

Attacker Model. The adversary $\mathcal{A}$ interacts with the oracles through queries. It is assumed to have full control over the communication network, modeled by a Send query which allows it to send arbitrary messages to any oracle. The adversary is also granted a number of additional queries that model the fact that various secrets might get lost or leaked. The queries are described in detail below.

- $\mathsf{Send}(i, s, m)$: This query allows $\mathcal{A}$ to send an arbitrary message m to oracle π_i^s. The oracle will respond according to the protocol specification and its current internal state. To start a new oracle, the message m takes a special form:

  $\langle \texttt{START}\colon \mathsf{role}, j\rangle$; this initialises π_i^s in the role role, having party P_j as its intended peer. Thus, this query sets $\mathsf{Pid}_i^s := j$ and $\mathsf{role}_i^s := \mathsf{role}$. Moreover, if π_i^s is started in the initiator role ($\mathsf{role} = \texttt{init}$), then it outputs the first message of the protocol.

- $\mathsf{RevLTK}(i)$: For $i \leq \mu$, this query allows the adversary to learn the long-term private key sk_i of user i. After this query i is said to be *corrupted*.

- $\mathsf{RegisterLTK}(i, pk_i)$: For $i > \mu$, this query allows the adversary to register a new party i with public key pk_i. We do not require that the adversary knows the corresponding private key. After the query the pair (i, pk_i) is distributed to all other parties. Parties registered by $\mathsf{RegisterLTK}$ are corrupted by definition.

- $\mathsf{RevSessKey}(i, s)$: This query allows the adversary to learn the session key derived by an oracle. That is, query $\mathsf{RevSessKey}(i, s)$ returns the contents of k_i^s. Recall that we have $k_i^s \neq \emptyset$ if and only if $\Psi_i^s = \texttt{accept}$. After this query π_i^s is said to be *revealed*.

Note that unlike, e.g., [12, 14], we do not allow the adversary to learn the sessions' ephemeral randomness.

Security experiment. To define the security of a key exchange protocol we want to evaluate the attacker's knowledge of the session keys. Formally, we have an AKE security game, played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where the adversary can issue the queries defined above. Additionally, it is given access to a special Test query, which, depending on a secret bit b chosen by the challenger, either returns real or random keys. The goal of the adversary is to guess b.

- $\mathsf{Test}(i,s)$: If $\Psi_i^s \neq \texttt{accept}$, return $\perp$. Else, return k_b, where $k_0 = k_s^i$ and $k_1 \xleftarrow{\$} \mathcal{K}$ is a random key. If a Test query is repeated in the case $b = 1$, the same random key is returned. After the query, oracle π_i^s is said to be *tested*.

The adversary can issue many Test queries, to different oracles, but all are answered using the *same* bit b.

The AKE security game, denoted $G_\Pi(\mu, \ell)$, is parameterized by the protocol Π and two numbers μ (the number of honest parties) and ℓ (the maximum number of protocol executions per party), and is run as follows.

1. $\mathcal{C}$ begins by drawing a random bit $b \xleftarrow{\$} \{0,1\}$, then generates μ long-term key pairs $\big\{(sk_i, pk_i) \bigm| i \in [1,\ldots,\mu]\big\}$, and initializes the collection of oracles $\big\{\pi_i^s \bigm| i \in [1,\ldots,\mu], s \in [1,\ldots,\ell]\big\}$.
2. $\mathcal{C}$ now runs $\mathcal{A}$, providing all the public keys $pk_1, \ldots, pk_\mu$ as input. During its execution, $\mathcal{A}$ may adaptively issue Send, RevLTK, $\mathsf{RevSessKey}$, $\mathsf{RegisterLTK}$ and Test queries any number of times and in arbitrary order. The only requirement is that all tested oracles remain *fresh* throughout the game (see Definition 7 below). Otherwise, the game aborts and outputs a random bit.
3. The game ends when $\mathcal{A}$ terminates with output b', representing its guess of the secret bit b. If not all test oracles are fresh, the security game outputs a random bit. If all test oracles are fresh and $b' = b$, it outputs 1. Otherwise, it outputs 0.

Definition 7 (Freshness)**.** An oracle π_i^s is *fresh*, written $\mathsf{fresh}(i,s)$, if:

(i) $\mathsf{RevSessKey}(i,s)$ has not been issued,

(ii) no query $\mathsf{Test}(j,t)$ or $\mathsf{RevSessKey}(j,t)$ has been issued, where π_j^t is a partner of π_i^s, and

(iii) Pid_i^s was:

  (a) not corrupted before π_i^s accepted if π_i^s has an origin-oracle, and

  (b) not corrupted at all if π_i^s has no origin-oracle.

Definition 8 (Winning events)**.** We define the following three winning events on game $G_\Pi(\mu, \ell)$.

(i) Event $\mathsf{break}_{\mathrm{Sound}}$ occurs if there exist two partner oracles π_i^s and π_j^t with $k_i^s \neq k_j^t$. In other words, there are two partner oracles which compute different session keys.

(ii) Event $\mathsf{break}_{\mathrm{Unique}}$ occurs if for some oracle π_i^s there exist distinct oracles π_j^t and $\pi_{j'}^{t'}$ such that π_i^s is a partner oracle to *both* π_j^t and $\pi_{j'}^{t'}$. In other words, there exists an oracle with more than one partner oracle.

(iii) Let $\mathsf{guess}_{\mathrm{KE}}$ be the output of game $G_{\Pi}(\mu, \ell)$. We define $\mathsf{break}_{\mathrm{KE}}$ to be the event $\mathsf{guess}_{\mathrm{KE}} = 1$.

Definition 9 (AKE Security)**.** An attacker $\mathcal{A}$ breaks the security of protocol Π, if at least one of $\mathsf{break}_{\mathrm{Sound}}$, $\mathsf{break}_{\mathrm{Unique}}$, or $\mathsf{break}_{\mathrm{KE}}$ occurs in $G_{\Pi}(\mu, \ell)$. The *advantage* of the adversary $\mathcal{A}$ against AKE security of Π is

$$\mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{A}) = \max\left\{\Pr\left[\mathsf{break}_{\mathrm{Sound}}\right], \Pr\left[\mathsf{break}_{\mathrm{Unique}}\right], |\Pr\left[\mathsf{break}_{\mathrm{KE}}\right] - 1/2|\right\}.$$

We say that $\mathcal{A}$ $(\epsilon_{\mathcal{A}}, t, \mu, \ell)$-*breaks* Π if its running time is t and $\mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{A}) \geq \epsilon_{\mathcal{A}}$. The running time of $\mathcal{A}$ includes the running time of the security experiment (see [21, Remark 1]).

Security properties. The core aspects of the security properties in our model are captured by the $\mathsf{break}_{\mathrm{KE}}$ event, combined with the adversary's capabilities and the restrictions imposed on them through the freshness predicate.

The freshness clauses (i) and (ii) imply that we only exclude the reveal of session keys for tested oracles as well as their partners. This encodes both (a) key independence if the revealed key is different from the session key: knowing some keys must not enable computing other keys, as well as (b) implicitly ensuring agreement on the involved parties, since sessions that compute the same session key but disagree on the parties would not be partnered, and reveal the Test session's key.

Our freshness clause (iii) encodes *weak forward secrecy*: the adversary can learn the peer's long-term key after the tested oracle accepted, but only if it has been passive in the run of the oracle [28]. Krawczyk [28] presented a generic attack that shows that a class of implicitly authenticated two-message protocols cannot achieve full forward secrecy, but only weak forward secrecy. We remark that this is a very standard class of protocols, including e.g. (H)MQV and NAXOS, and that adding a key confirmation message leads to full forward secrecy [28]. Another property captured by our model is resistance to *key-compromise impersonation* attacks. Recall that KCI attacks are those where the adversary uses a party A's own private long-term key to impersonate other users towards A. This is (implicitly) encoded by the absence of any adversary restrictions on learning the private long-term key of a test-oracle itself. Additionally, the $\mathsf{break}_{\mathrm{Unique}}$ event captures the resistance to replay attacks. The $\mathsf{break}_{\mathrm{Sound}}$ event ensures that two parties that execute the protocol together in the absence of an attacker (or at least a passive one), compute the same session key.

Some recent protocols also offer *post-compromise security*, in which the communication partner π_j^t may be corrupted before π_i^s has accepted. However, in this work we consider only stateless protocols, which cannot achieve this goal [16].

4 Protocol Π

In this section we prove our first main result: a nearly tight proof for a protocol we call Π, which is extremely efficient both in terms of communication as well

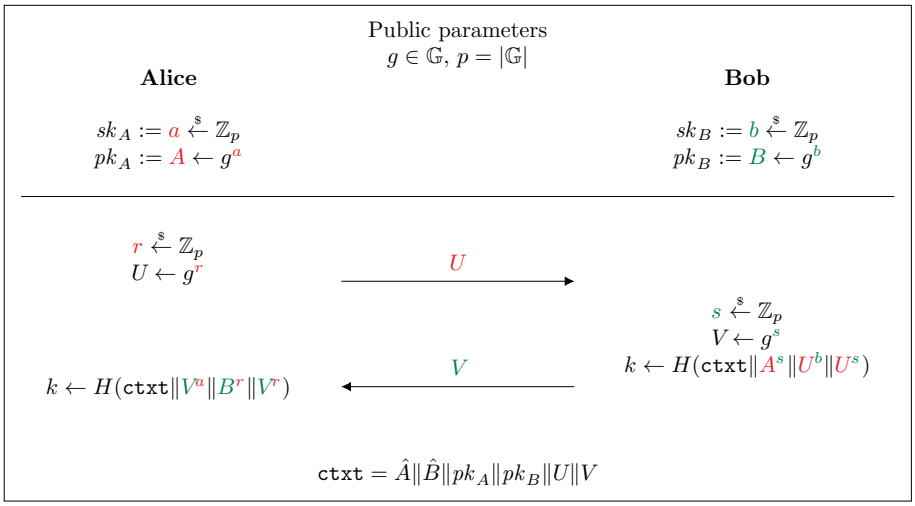

Figure 1: Protocol Π. The session key is derived from the combination of the parties' static-ephemeral, ephemeral-static, and ephemeral-ephemeral DH values.

as computational complexity. Protocol Π, defined in Fig. 1, uses a mix of static-ephemeral and ephemeral-ephemeral Diffie-Hellman key exchanges to the derive its session key. Specifically, the two protocol participants exchange ephemeral Diffie-Hellman shares g^r and g^s for random r, s, and then compute a session key from three Diffie-Hellman shared secrets (static-ephemeral, ephemeral-static, ephemeral-ephemeral) as well as identities and a transcript. Note that protocol Π is very close to the Noise-KK pattern [38].

Theorem 2. *Consider the protocol* Π *defined in Fig. 1 where* H *is modeled as a random oracle. Let* $\mathcal{A}$ *be an adversary against the AKE security of* Π*. Then there exist adversaries* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *and* $\mathcal{B}_3$ *against strong Diffie-Hellman such that*

$$\mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{A}) \leq \mu \cdot \mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_1) + \mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_2) + \mu \cdot \mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_3) + \frac{\mu\ell^2}{p}.$$

The strong Diffie-Hellman adversaries all run in essentially the same time as $\mathcal{A}$*, and make at most as many queries to their strong DH-oracle as* $\mathcal{A}$ *makes to its hash oracle* H*.*

The proof of Theorem 2 is structured as a sequence of games running variations on the security experiment, with the first game identical to the experiment. We bound the difference in the probability of the event that the experiment outputs 1 in each game. As a side effect, along the way we also get a bound on $\mathsf{break}_{\mathrm{Unique}}$. Then we argue that the probability that the experiment outputs 1 is 1/2 in the final game, which gives us a bound on $\mathsf{break}_{\mathrm{KE}}$. Since the scheme has perfect correctness, the theorem follows.

To achieve this result in the final game, we shall have our oracles choose session keys at random, without reference to secret keys or messages. Obviously,

we have to ensure consistency with what the adversary can learn. This means that we have to make sure that partnered oracles both choose the same key (Game 2); that keys the adversary should be able to compute on his own are the same as chosen by the oracle (Game 2), and that corruptions of long-term keys that enable the adversary to compute session keys on his own return results consistent with previous RevSessKey-queries (Game 3 and 5).

The general technique we use is to have our session oracles refrain from computing the input to the key derivation function H (i.e., the RO), but instead check to see if the adversary somehow computes it. Namely, the reduction can check if the adversary ever submits the correct input to H by using the strong DH oracle provided.

We call an oracle *honest* (at some point) if the user it belongs to has not yet been corrupted (at that point). There are five types of oracles that we will have to deal with in separate ways, and the first four are essentially fresh oracles:

- (I) initiator oracles whose response message comes from a responder oracle, which has the same `ctxt` (i.e., they agree on the message transcript and participant identities and public keys) and which is honest when the response is received;
- (II) other initiator oracles whose intended peer is honest until the oracle accepts;
- (III) responder oracles whose initial message comes from an initiator, which has the same `ctxt` up to the responder message (thus agreeing on the first message and participant identities and public keys) and which is honest when the response is received;
- (IV) other responder oracles whose intended peer is honest until the oracle accepts; and
- (V) oracles whose intended peer is corrupted.

Note that at the time an initiator oracle starts, we cannot know if it will be of type I or II. However, we will know what type it is when it is time to compute the oracle's session key. We also remark that types I and III correspond to case (iii)a in the definition of freshness. Types II and IV correspond to case (iii)b.

In the following, let S_j denote the event that the experiment in Game j outputs 1.

Game 0. Our starting point Game 0 is the security experiment defining AKE security. We have that

$$\Pr\left[\mathsf{break}_{\mathrm{KE}}\right] = \Pr[S_0]. \tag{1}$$

We begin with an administrative step to avoid pathologies where honest players choose the same random nonces.

Game 1. In this game, we abort if two initiator oracles or two responder oracles ever arrive at the same `ctxt`. The probability of this happening can be upper-bounded by the probability of two oracles for the same peer choosing the same random exponents, and we get that

$$|\Pr[S_1] - \Pr[S_0]| \leq \frac{\mu\ell^2}{p}. \tag{2}$$

We also note that the event in this game that corresponds to $\mathsf{break}_{\mathrm{Unique}}$ cannot happen in this game. It follows that

$$\Pr[\mathsf{break}_{\mathrm{Unique}}] \leq \frac{\mu\ell^2}{p}. \tag{3}$$

4.1 Preparing Oracles

Our goal in this game is to change every oracle so that it no longer computes the input to the key derivation hash H, but instead checks if the adversary computes this input and adapts accordingly. This is essential for later games, since it allows us to replace every use of the private key with queries to a strong DH oracle.

Game 2. In this game, we modify how our oracles determine their session keys. Note that at the point in time where an initiator oracle determines its session key, we know its type exactly.

For a type III, IV, or V responder oracle π_i^d with $\texttt{ctxt} = \hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V$, having private key $sk_i = b$ and random exponent s, and where the public key of its initiator peer is $pk_j = A$, the game does the following to determine π_i^d's session key k. First, it checks to see if any H-queries $\hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V\|W_1\|W_2\|W_3$ have been made satisfying

$$W_1 = A^s \qquad W_2 = U^b \qquad W_3 = U^s. \tag{4}$$

If any such query is found k is set to the corresponding hash value. Otherwise, the session key is chosen at random. And if such a hash query happens later, the hash value is set to the chosen session key.

A type I initiator oracle will simply use the key from the corresponding responder oracle.

For a type II or V initiator oracle π_i^d with $\texttt{ctxt} = \hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V$, having private key $sk_i = a$ and random exponent r, and where the public key of its responder peer is $pk_j = B$, the game does the following to determine π_i^d's session key k. First, it checks to see if any oracle queries $\hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V\|W_1\|W_2\|W_3$ have been made satisfying

$$W_1 = V^a \qquad W_2 = B^r \qquad W_3 = V^r. \tag{5}$$

If any such query is found, k is set to the corresponding hash value. Otherwise, the session key is chosen at random. And if such a hash query happens later, the hash value is set to the chosen session key.

The only potential change in this game is at which point in time the key derivation hash oracle value is first defined, which is unobservable. It follows that

$$\Pr[S_2] = \Pr[S_1]. \tag{6}$$

4.2 Type IV Responder Oracles

Game 3. In this game type IV oracles choose their session key at random, but do not modify the hash oracle unless the intended peer is corrupted. If the adversary corrupts the intended peer i of a type IV oracle π_j^d, having private key $sk_j = b$, random exponent s, and chosen key k, then from that point in time, any query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V\|pk_i^s\|U^b\|U^s$$

to the key derivation hash oracle H will result in the hash value k.

Unless one of these queries happen before user i is corrupted, the only change is at which point in time the key derivation hash oracle value is first defined, which is unobservable. Let F be the event that a query as above happens before the corresponding long-term key is corrupted. Then

$$|\Pr[S_3] - \Pr[S_2]| \leq \Pr[F].$$

Let F_i be the same event as F, but with the intended peer being user i. We then have that $\Pr[F] = \sum_i \Pr[F_i]$.

Next, consider the event E_i which is that for some type IV oracle as above, any query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V\|W_1\|W_2\|W_3 \qquad\qquad W_1 = pk_i^s = V^a \tag{7}$$

to the key derivation hash oracle H happens before user i is corrupted. Then $\Pr[F_i] \leq \Pr[E_i]$.

We shall now bound the probability of the event E_i by constructing an adversary against strong Diffie-Hellman. This adversary will embed its DH challenge in some user i's public key and type IV oracle responses for oracles whose intended peer is user i, and recover the solution to its DH challenge from the hash query in event E_i.

Strong Diffie-Hellman adversary $\mathcal{B}_1$. The algorithm $\mathcal{B}_1$ takes as input a DH challenge $(X, Y) = (g^x, g^y)$ and outputs a group element Z. It has access to a strong Diffie-Hellman oracle $\mathsf{stDH}_x(\cdot,\cdot)$.

Reduction $\mathcal{B}_1$ runs Game 2 with the following changes: it chooses i uniformly at random and sets user i's public key to $pk_i = X$ (and thus implicitly sets i's private key to the unknown value x). For type IV oracles whose intended peer is user i, $\mathcal{B}_1$ sets $V = Y \cdot g^\rho$, with ρ random (an independent random value ρ is chosen for each oracle). If the adversary corrupts user i, the reduction $\mathcal{B}_1$ aborts. For other users, the reduction simply returns the private key, as in Game 2.

We need to recognise hash queries of the form (4) and (5) that involve user i, as well as queries of the form (7). For (4), where user i acts in the responder role, we know the oracle's random exponent s, so we only need to recognise if W_2 is U raised to user i's private key, which can be done by checking if $\mathsf{stDH}_x(U, W_2) = 1$.

For (5), where user i is the initiator, we know the oracle's random exponent r, so we only need to recognise if W_1 is V raised to user i's private key, which can be done by checking if $\mathsf{stDH}_x(V, W_1) = 1$.

Finally, for (7), we need to recognise if a group element W_1 is V raised to user i's private key, which can be done by checking if $\mathsf{stDH}_x(V, W_1) = 1$. When we recognise a query of the form (7), since we know that $V = Y \cdot g^{\rho}$, we output

$$Z = W_1 X^{-\rho} = V^x X^{-\rho} = Y^x g^{\rho x} g^{-x\rho} = Y^x.$$

In other words, our adversary $\mathcal{B}_1$ succeeds whenever E_i would happen in Game 2. Furthermore, E_i in Game 2 can only happen before user i is corrupted, so whenever E_i would happen in Game 2, $\mathcal{B}_1$ would not have aborted.

We get that

$$\mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_1) \geq \frac{1}{\mu} \sum_i \Pr[E_i] \geq \frac{1}{\mu} \sum_i \Pr[F_i] = \frac{1}{\mu} \Pr[F],$$

from which it follows that

$$|\Pr[S_3] - \Pr[S_2]| \leq \Pr[F] \leq \mu \cdot \mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_1). \tag{8}$$

4.3 Type III Responder Oracles

Game 4. In this game type III responder oracles choose their session key at random, and do not modify the key derivation hash oracle.

Consider a type III responder oracle for user j with private key $sk_j = b$, random exponent s, and intended initiator peer i. Unless the adversary ever makes a hash query of the form

$$\hat{i} \| \hat{j} \| pk_i \| pk_j \| U \| V \| W_1 \| W_2 \| W_3 \qquad\qquad W_3 = U^s, \tag{9}$$

this change is unobservable. Call this event F, thus

$$|\Pr[S_4] - \Pr[S_3]| \leq \Pr[F]. \tag{10}$$

We shall bound the probability of F by constructing an adversary against strong Diffie-Hellman. This adversary will embed its challenge in type I or II initiator oracles' message, as well as in type III responder oracles' message. It will recover the solution to its DH challenge from the hash query in event F.

Strong Diffie-Hellman adversary $\mathcal{B}_2$. The algorithm $\mathcal{B}_2$ takes as input a DH challenge $(X, Y) = (g^x, g^y)$ and outputs a group element Z. It has access to a strong DH-oracle $\mathsf{stDH}_x(\cdot, \cdot)$.

Our reduction $\mathcal{B}_2$ runs Game 3 with the following changes: for type I and II initiator oracles (we cannot distinguish these at this point in time), it computes $U = X \cdot g^{\rho_0}$, with ρ_0 random. For type III responder oracles, it computes $V = Y \cdot g^{\rho_1}$, with ρ_1 random.[2] Note that in this game, the reduction knows all static private keys, so user corruption is handled exactly as in Game 3.

We need to recognise hash queries of the form (5) for type II initiator oracles, as well as queries of the form (9) for type III oracles. Although we do not know the oracle's random exponents, we do know their private keys. This means that we only need to recognise if W_3 is V raised to $\log_g U = x + \rho_0$. Of course, if $W_3 = V^{x+\rho_0}$, then $W_3 \cdot V^{-\rho_0} = V^x$, which we can detect by checking if $\mathsf{stDH}_x(V, W_3 V^{-\rho_0}) = 1$. If this is the case for a query of the form (9), then we output

$$Z = W_3 \cdot V^{-\rho_0} \cdot X^{-\rho_1} = V^x \cdot X^{-\rho_1} = g^{yx+\rho_1 x} g^{-x\rho_1} = Y^x$$

as the solution to the DH challenge. In other words, $\mathcal{B}_2$ succeeds whenever F would happen in Game 3, hence

$$|\Pr[S_4] - \Pr[S_3]| \leq \Pr[F] \leq \mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_2). \tag{11}$$

Note that we do not stop the simulation in the case we detect a hash query of the form (5) for a type II initiator oracle, because in this case the responder message V does not contain the embedded DH challenge.

4.4 Type II Initiator Oracles

Game 5. In this game type II initiator oracles choose their session key at random, but do not modify the hash oracle unless the intended peer is corrupted. If the adversary corrupts the intended peer j of a type II oracle running as user i with private key $sk_i = a$, random exponent r, and chosen key k, then from that point in time, any query of the form

$$\hat{i} \| \hat{j} \| pk_i \| pk_j \| U \| V \| V^a \| pk_j^r \| V^r$$

to the key derivation hash oracle H will result in the hash value k.

Unless one of these queries happen before the user j is corrupted, the only change is at which point in time the key derivation hash oracle value is first defined, which is unobservable. Let F be the event that a query as above happens before the corresponding long-term key is corrupted. Then

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr[F].$$

Let F_j be the same event as F, but with the intended peer being user j. We then have that $\Pr[F] = \sum_j \Pr[F_j]$.

Next, consider the event E_j which is that for some type II oracle as above, any H-query of the form

$$\hat{i} \| \hat{j} \| pk_i \| pk_j \| U \| V \| W_1 \| W_2 \| W_3 \qquad\qquad W_2 = pk_j^r = U^b \tag{12}$$

[2] Again, independent random values ρ_0 and ρ_1 are chosen for each oracle.

happens before user j is corrupted (here, $sk_j = b$). Then $\Pr[F_j] \leq \Pr[E_j]$.

We shall now bound the probability of the event E_j by constructing an adversary against strong Diffie-Hellman. This adversary will embed its DH challenge in some user j's public key and type II oracle messages for oracles whose intended peer is user j, and recover the solution to its DH challenge from the hash query in event E_j.

Strong Diffie-Hellman adversary $\mathcal{B}_3$. The algorithm $\mathcal{B}_3$ takes as input a DH challenge $(X, Y) = (g^x, g^y)$ and outputs a group element Z. It has access to a strong DH-oracle $\mathsf{stDH}_x(\cdot, \cdot)$.

Our reduction $\mathcal{B}_3$ runs Game 4 with the following changes: It chooses j uniformly at random and sets user j's public key to $pk_j = X$ (and thus implicitly sets j's private key to the unknown value $b = x$). For type I and II initiator oracles whose intended peer is user j, $\mathcal{B}_3$ sets $U = Y \cdot g^\rho$, with ρ random. If the adversary corrupts user j, the reduction $\mathcal{B}_3$ aborts. For other users, the reduction simply returns the private key, as in Game 4.

We need to recognise hash queries of the form (4) and (5) that involve user j, as well as queries of the form (12). For (4), where user j is the responder, we know the oracle's random exponent s, so we only need to recognise if W_2 is U raised to user j's private key, which can be done by checking if $\mathsf{stDH}_x(U, W_2) = 1$. For (5), where user j is the initiator, we know the oracle's random exponent r, so we only need to recognise if W_1 is V raised to user j's private key, which can be done by checking if $\mathsf{stDH}_x(V, W_1) = 1$. Finally, for (12), we need to recognise if a group element W_2 is U raised to user j's private key, which can be done by checking if $\mathsf{stDH}_x(U, W_2) = 1$.

When we recognise a query of the form (12), meaning that $W_2 = U^x$ where know that $U = Y \cdot g^\rho$, then we output

$$Z = W_2 X^{-\rho} = U^x X^{-\rho} = Y^x g^{\rho x} g^{-x\rho} = Y^x.$$

In other words, our adversary $\mathcal{B}_3$ succeeds whenever E_j would happen in Game 4. Furthermore, E_j in Game 4 can only happen before user j is corrupted, so whenever E_j would happen in Game 4, $\mathcal{B}_3$ would not have aborted. We get that

$$\mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_3) \geq \frac{1}{\mu} \sum_j \Pr[E_j] \geq \frac{1}{\mu} \sum_j \Pr[F_j] = \frac{1}{\mu} \Pr[F],$$

from which it follows that

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr[F] \leq \mu \cdot \mathrm{Adv}^{\mathsf{stDH}}_{\mathbb{G},g}(\mathcal{B}_3). \tag{13}$$

4.5 Summary

Note that in Game 5, every session key is chosen at random independent of every key and sent message.

For type V oracles, the key derivation oracle is immediately programmed so that the session key is available to the adversary. But type V oracles are never fresh and therefore never subject to a Test query.

For type II and IV oracles, the key derivation hash oracle is programmed to make the session key available to the adversary only after the intended peer is corrupted. But if the intended peer is corrupted, a type II or IV oracle will become non-fresh, hence no Test query can be made to it.

For type I and III oracles, the key derivation hash oracle will never make the session key available to the adversary.

This means that for any oracle subject to a Test query, the session key is and will remain independent of every key and sent message. Which means that the adversary cannot distinguish the session key from a random key. It follows that

$$\Pr[S_5] = \frac{1}{2}. \tag{14}$$

Theorem 2 now follows from (1), (2), (6), (8), (11), (13) and (14). (Recall that Game 1 gives us $\Pr[\mathsf{break}_{\mathrm{Unique}}] \leq \mu\ell^2/p$, and that protocol Π has perfect correctness so $\Pr[\mathsf{break}_{\mathrm{Sound}}] = 0$.)

5 Avoiding the Strong Diffie-Hellman Assumption

The proof of protocol Π relies on the strong Diffie-Hellman assumption, which is an interactive assumption. A natural goal is to look for a protocol whose proof relies on standard non-interactive assumptions. In this section we present two protocols that solve this problem. Both can be seen as different modifications of protocol Π.

5.1 Protocol Π_{Twin}

The first protocol, which we call Π_{Twin}, applies the twinning technique of [15] to the different DH values in protocol Π. This requires some additional exponentiations over protocol Π, as well as the need to transmit one extra group element. The details are given in Fig. 2: instead of sending a single Diffie-Hellman share, the protocol initiator samples and sends two ephemeral shares, and both shares are used in the key derivation. This duplication allows us to reduce to twin Diffie-Hellman.

Theorem 3. *Consider the protocol Π_{Twin} defined in Fig. 2 where H is modeled as a random oracle. Let $\mathcal{A}$ be an adversary against the AKE security of Π_{Twin}. Then there exists adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ against strong twin Diffie-Hellman such that*

$$\mathrm{Adv}^{\mathsf{AKE}}_{\Pi_{\mathrm{Twin}}}(\mathcal{A}) \leq \mu \cdot \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_1) + \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_2) + \mu \cdot \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_3) + \frac{\mu\ell^2}{p}.$$

The adversaries all run in essentially the same time as $\mathcal{A}$ and make at most as many queries to their twin DH oracle as $\mathcal{A}$ makes to its hash oracle H.

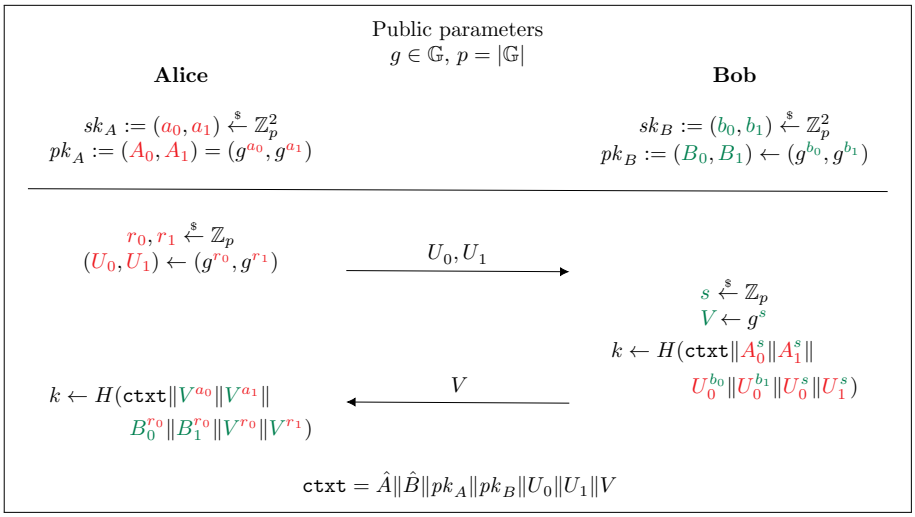

Figure 2: Protocol Π_{Twin}. It is obtained from protocol Π by applying the twinning trick of [15] to the DH values.

Since this protocol is a twinned version of protocol Π, the proof and its ideas follow the proof of Theorem 2 very closely. The full proof is given in Appendix A. Note that by Theorem 1, we can tightly replace the twin Diffie-Hellman terms in the theorem statement by ordinary computational Diffie-Hellman terms.

5.2 Protocol Π_{Com}

The second protocol, which we call Π_{Com}, again uses the twinning technique of [15], but this time only applied to the static DH values in Π. This provides tight implicit authentication. However, instead of also twinning the ephemeral DH values we use a variant of the commitment trick of [21]. This reduces the number of exponentiations compared to Π_{Twin}, but adds another round of communication. Also, we need to rely on the decisional Diffie-Hellman assumption instead of computational Diffie-Hellman. The details are given in Fig. 3.

Theorem 4. *Consider the protocol Π_{Com} defined in Fig. 3 where H and G are modeled as random oracles. Let $\mathcal{A}$ be an adversary against the AKE security of Π_{Com}. Then there exists adversaries $\mathcal{B}_1$ and $\mathcal{B}_3$ against computational Diffie-Hellman and an adversary $\mathcal{B}_2$ against Decision Diffie-Hellman such that*

$$\mathrm{Adv}^{\mathsf{AKE}}_{\Pi_{\text{Twin}}}(\mathcal{A}) \leq \mu \cdot \mathrm{Adv}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{B}_1) + \mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{B}_2) + \mu \cdot \mathrm{Adv}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{B}_3) + \frac{\mu \ell^2 (1 + 2t)}{p}.$$

The adversaries all run in essentially the same time t as $\mathcal{A}$ and make at most as many queries to their twin DH oracle as $\mathcal{A}$ makes to its hash oracle H.

Since this protocol is partially a twinned version of protocol Π, the proof and its ideas follow the proof of Theorem 2. However, since the ephemeral-ephemeral

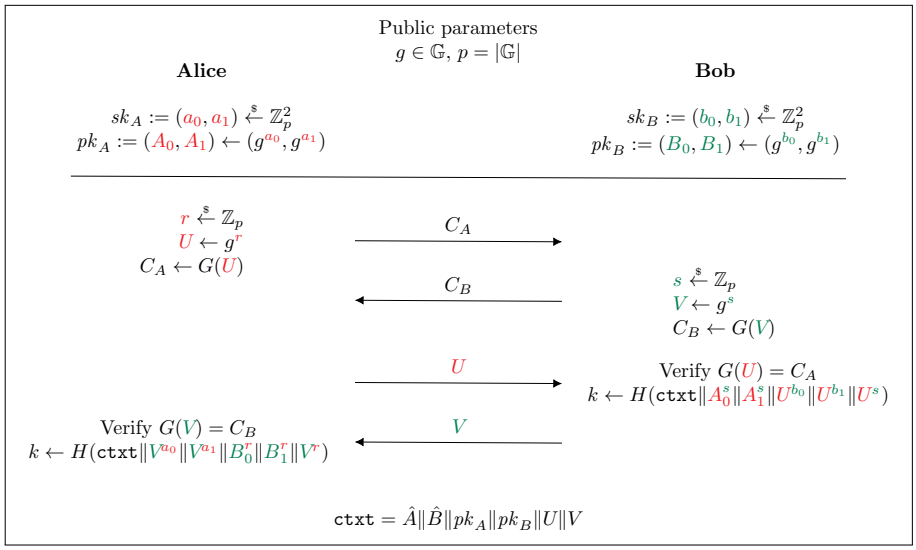

Figure 3: Protocol Π_{Com}. It is obtained from protocol Π by applying the twinning trick of [15] to the static DH values and the commitment trick of [21] to the ephemeral DH values.

part is different, its analysis is also quite different. The full proof is given in Appendix B.

6 Efficiency Analysis

In this section we argue that our protocols are more efficient than other comparable[3] protocols in the literature when instantiated with theoretically sound parameter choices. There are two reasons for this. First, the most efficient key protocols do not have tight proofs. Hence, for theoretically sound deployment they must use larger parameters to compensate for the proof's security loss, which directly translates into more expensive operations. The result is that although some protocols require fewer operations than ours (typically group exponentiations), the increase in computational cost *per operation* dominates whatever advantage they might have over our protocols in terms of *number of operations*.

Second, the few known key exchange protocols which *do* have tight proofs, require a large number of operations or heavy cryptographic machinery. Thus, even though they can use small parameters, such as the P-256 elliptic curve, here the sheer number of operations dominates their advantage over our protocols.

To illustrate the first point in more detail, here are some examples of very efficient key exchange protocols having non-tight security proofs: UM [35],

[3] Comparing protocols is complex, and we return to this at the end of this section.

Table 1: The number of group exponentiations in our protocols compared to other protocols in the literature. All protocols are one-round except Π_{Com}, which has two rounds of communication. All security proofs are in the random oracle model. The security loss is in terms of the number of users (μ), the number of protocol instances per user (ℓ), and reduction's running time (t).

Protocol	**#Exponentiations**	**Assumption**	**Security loss** $O(\cdot)$
HMQV [28]	2.5	CDH	$\mu^2\ell^2$
NAXOS [31]	3	Gap-DH	$\mu^2\ell^2$
UM [35]	3	Gap-DH	$\mu^2\ell^2$
Kudla-Paterson [30]	3	Gap-DH	$\mu^2\ell$
KEA+ [32]	3	Gap-DH	$\mu\ell^\dagger$
Π (Fig. 1)	4	Strong-DH	μ
Π_{Twin} (Fig. 2)	8/7	CDH	μ
Π_{Com} (Fig. 3)	6	DDH	μ
GJ [21]	17	DDH	1

† Only when using pairing-friendly curves; otherwise $L = O(\mu\ell t)$.

KEA+ [32], HMQV [28], CMQV [41], $\mathcal{TS}1/2/3$ [26], Kudla-Paterson [30], and NAXOS [31]. Typically, these proofs have a tightness loss between $L = O(\mu\ell)$ and $L = O(\mu^2\ell^2)$ as illustrated for a few of the protocols in Table 1.

Suppose we now want to compare the efficiency of the protocols Π, Π_{Twin}, Π_{Com} and HMQV, aiming for around 110-bits of security. Following Gjøsteen and Jager [21], let us imagine two different scenarios: a small-to-medium-scale setting with $\mu = 2^{16}$ users and $\ell = 2^{16}$ sessions per user, and a large-scale setting with $\mu = 2^{32}$ users and $\ell = 2^{32}$ sessions per user. To instantiate the protocols in a theoretically sound manner we need to select a group large enough so that the underlying DH-assumptions are still hard even when accounting for the security loss. For simplicity, we only consider selecting among elliptic curve groups based on the NIST curves P-256, P-384, and P-521, and assume that the CDH, DDH, and Gap-DH problems are equally hard in each group.

HMQV. Supposing HMQV has a tightness loss of $L \approx \mu^2\ell^2$, this translates into a loss of 2^{64} in the small-to-medium-scale setting, and a loss of 2^{128} in the large-scale setting. To compensate we have to increase the group size by a factor of $L^2 \approx 2^{128}$ and $L^2 \approx 2^{256}$, respectively. With a target of 110-bit security, this means that we have to instantiate HMQV with curve P-384 and P-521, respectively.

Π, Π_{Twin}, Π_{Com}. Our protocols' security proofs have a tightness loss of $L \approx \mu$, which translates into 2^{16} in the small-to-medium-scale setting and 2^{32} in the large-scale setting. In the first setting P-256 is still sufficient for 110-bit security, but in the later setting P-384 must be used instead.

We can now compare these instantiations by multiplying the number of exponentiations required with the cost of an exponentiation in the relevant group.

Table 2: OpenSSL Benchmark Results for NIST Curves [21, Table 1].

Curve	Exp. / Sec.	Time / Exp.
NIST P-256	476.9	2.1 ms
NIST P-384	179.7	5.6 ms
NIST P-521	62.0	16.1 ms

For the latter values we use the OpenSSL benchmark numbers from Gjøsteen and Jager [21] (reproduced in Table 2). Calculating the numbers we get:

	HMQV	Π	Π_{Twin}	Π_{Com}
S-M	$2.5 \times 5.6 = 14$	$4 \times 2.1 = 8.4$	$8 \times 2.1 = 16.8$	$6 \times 2.1 = 12.6$
L	$2.5 \times 16.1 = 40.3$	$4 \times 5.6 = 22.4$	$8 \times 5.6 = 44.8$	$6 \times 5.6 = 33.6$

Observe that Π is more efficient than HMQV in both the small-to-medium-scale setting as well as in the large-scale setting despite needing more exponentiations. This is because it can soundly use smaller curves than HMQV due to the relative tightness of its reduction. Protocol Π_{Twin} is about as efficient as HMQV in both settings, while Π_{Com} lies somewhere in between Π and Π_{Twin}, but since it requires one extra round of communication a direct comparison is more difficult. Of course, the main reason to prefer Π_{Twin} and Π_{Com} over Π is the reliance on the weaker CDH and DDH assumptions rather than strong DH. A complicating factor in comparing with HMQV is the difference in security properties and security models (see the end of this section).

To illustrate the second point mentioned above—that our protocols are also more efficient than protocols with fully tight proofs—we also compute the numbers for the recent protocol of Gjøsteen and Jager (GJ) which is currently the most efficient key exchange protocol with a fully tight proof. Since GJ can use P-256 independent of the number of users and sessions its cost is $17 \times 2.1 = 35.7$ in both the small-to-medium scale setting as well as the large-scale setting. Nevertheless, we observe that the large number of exponentiations in GJ dominates its tightness advantage in realistic settings.

Thus, absent a fully tight proof, our protocols hit a proverbial "sweet spot" between security loss and computational complexity: they can be instantiated soundly on relatively small curves using only a few exponentiations.

Communication complexity. For completeness we also briefly mention communication complexity. Since in most implicitly-authenticated DH-based protocols each user only sends one or two group elements, there is in practice little difference between Π, Π_{Twin}, and Π_{Com}, and protocols like HMQV when it comes to communication cost. Especially if elliptic curve groups are used.

This is in contrast to the fully tight signature-based GJ protocol, which in total needs to exchange two group elements for the Diffie-Hellman key exchange,

two signatures (each consisting of a random 256-bit exponent, two group elements, and four 256-bit exponents), and one hash value. Altogether, this gives a total of ≈ 545 bytes communicated when instantiated for a security level of, say, 128 bits [21, Section 5]. In comparison, Π, Π_{Twin}, and Π_{Com} would only need to exchange around 160 to 224 bytes for the same security level. This assumes curve P-384 and includes the addition of two 256-bit key-confirmation messages to provide explicit entity authentication in order to make the comparison with the GJ protocol fair.

On the (im)possibility of fairly comparing protocols. Our protocols are the first implicitly authenticated key exchange protocols that were designed to provide efficient deployment in a theoretically sound manner. This implies that we must compare their efficiency with other protocols with slightly different goals. In Table 1 we included protocols with closely related goals and similar structure, but not aiming for exactly the same target.

One example of such a different goal is that NAXOS was designed to be proven in the eCK model, which also allows the reveal of the randomness of the tested session, similar to HMQV. Our protocols, like TLS 1.3, currently do not offer this property. We conjecture that the NAXOS transformation could be directly applied to our protocols to obtain eCK-secure protocols without adding exponentiations, but it is currently unclear if this could be done with a *tight* proof, and hence we leave this to future work.

7 Optimality of our Security Proofs

In this section we will show that the tightness loss of $L = O(\mu)$ in Theorem 2, Theorem 3 and Theorem 4 is essentially optimal—at least for "simple" reductions. Basically, a "simple" reduction runs a *single* copy of the adversary only *once*. To the best of our knowledge, all known security reductions for AKE protocols are either of this type or use the forking lemma. For example, the original reduction for HMQV uses the forking lemma and thus is very non-tight, but does not fall under our lower bound. In contrast, the HMQV reduction by Barthe et al. [5] is simple and thus our lower bound applies. Hence, in order to give a tighter security proof, one would have to develop a completely new approach to prove security for such protocols.

Tightness bounds for different cryptographic primitives were given in [4, 17, 19, 20, 23, 25, 27, 33, 37, 39, 42], for instance. Bader et al. [4] describe a generic framework that makes it possible to derive tightness lower bounds for many different primitives. However, these techniques are only able to consider tight reductions from *non-interactive* assumptions, while our first protocol is based on the *interactive* strong Diffie-Hellman assumption. Morgan and Pass [36] showed how to additionally capture *bounded-round* interactive assumptions, but the strong Diffie-Hellman assumption does not bound the number of possible oracle queries, so we cannot use their approach directly.

Therefore we develop a new variant of the approach of Bader et al. [4], which makes it possible to capture interactive assumptions with an unbounded number of oracle queries, such as strong Diffie-Hellman assumption. For clarity and simplicity, we formulate this specifically for the class of assumptions and protocols that we consider, but we discuss possible extensions below.

Considered class of protocols. In the following we consider protocols where public keys are group elements of the form $pk = g^x$ and the corresponding private key is $sk = x$. We denote the class of all protocols with this property with Π_{DH}. Note that this class contains, in particular, NAXOS [31], KEA+ [32], and HMQV [28].

Remark 1. One can generalize our results to *unique and verifiable* private keys, which essentially requires that for each value pk there exists only one unique matching private key sk, and that there exists an efficiently computable relation R such that $R(pk, sk) = 1$ if and only if (pk, sk) is a valid key pair. Following Bader et al. [4], one can generalize this further to so-called *efficiently re-randomizable* keys. We are not aware of concrete examples of protocols that would require this generality, and thus omit it here. All protocols considered in the present paper and the vast majority of high-efficiency protocols in the literature have keys of the form $(pk, sk) = (g^x, x)$, so we leave such extensions for future work.

Why does GJ18 not contradict our lower bound? As mentioned in Remark 1, our bound applies to protocols with *unique and verifiable* private keys. In contrast, the protocol of Gjøsteen and Jager [21] constructs a tightly-secure digital signature scheme based on OR-proofs, where private keys are *not* unique. As explained in [21, Section 1.1], these non-unique private keys seem inherently necessary to achieve fully-tight security.

Simple reductions from (strong) Diffie-Hellman. Intuitively, a *simple* reduction $\mathcal{R} = \mathcal{R}^{\mathcal{O}}$ from (strong) CDH takes as input a CDH instance (g^x, g^y) and may query an oracle $\mathcal{O}$ that, on input Y, Z, returns 1 if and only if $Y^x = Z$ (cf. Definition 3). More formally:

Definition 10. A *simple* reduction $\mathcal{R}$ interacts with an adversary $\mathcal{A}$ as follows.

1. $\mathcal{R}$ receives as input a CDH instance (g^x, g^y).
2. It generates μ public keys and starts $\mathcal{A}(pk_1, \ldots, pk_\mu)$. $\mathcal{R}$ provides $\mathcal{A}$ with access to all queries provided in the security model described in Section 3.
3. $\mathcal{R}$ outputs a value h.

We say that $\mathcal{R}$ is a $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, \epsilon_{\mathcal{A}})$-*reduction*, if it runs in time at most $t_{\mathcal{R}}$ and for any adversary $\mathcal{A}$ with $\epsilon_{\mathcal{A}} = \mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{A})$ holds that

$$\Pr\left[h = g^{xy}\right] \geq \epsilon_{\mathcal{R}}.$$

We say that $\mathcal{R} = \mathcal{R}^{\mathcal{O}}$ is a reduction from the *strong* CDH problem if it makes at least one query to its oracle $\mathcal{O}$, and a reduction from the CDH problem if not.

Remark 2. The formalization in this section very specifically considers the computational problems CDH and sCDH, as concrete examples of reasonable hardness assumptions that a typical security proof for the protocols considered in this work may be based on. We will later discuss how our results can be extended to other interactive and non-interactive problems.

Theorem 5. *Let* Π *be an AKE protocol such that* $\Pi \in \Pi_{\mathsf{DH}}$. *Let* $|\mathcal{K}|$ *denote the size of the key space of* Π. *For any simple* $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, 1-1/|\mathcal{K}|)$*-reduction* $\mathcal{R}^{\mathcal{O}}$ *from (strong) CDH to breaking* Π *in the sense of Definition 9 there exists an algorithm* $\mathcal{M}^{\mathcal{O}}$*, the meta-reduction, that solves the (strong) CDH problem in time* $t_{\mathcal{M}}$ *and with success probability* $\epsilon_{\mathcal{M}}$ *such that* $t_{\mathcal{M}} \approx \mu \cdot t_{\mathcal{R}}$ *and*

$$\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - \frac{1}{\mu}.$$

Remark 3. Note that the lower bound $\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - 1/\mu$ implies that the success probability $\epsilon_{\mathcal{R}}$ cannot significantly exceed $1/\mu$, as otherwise there exists an efficient algorithm $\mathcal{M}$ for a computationally hard problem. Note also that this implies that the reduction cannot be tight, as it "loses" a factor of at least $1/\mu$, even if the running time of $\mathcal{R}$ is not significantly larger than that of the adversary.

In the sequel we write $[\mu \setminus i]$ as a shorthand for $[1 \ldots i-1, i+1 \ldots \mu]$.

Proof. We describe a *meta-reduction* $\mathcal{M}$ that uses $\mathcal{R}$ as a subroutine to solve the (strong) CDH problem. Following Hofheinz et al. [23] and Bader et al. [4], we will first describe a *hypothetical* inefficient adversary $\mathcal{A}$. Then we explain how this adversary is efficiently simulated by $\mathcal{M}$. Finally, we bound the success probability of $\mathcal{M}$, which yields the claim.

Hypothetical adversary. The hypothetical adversary $\mathcal{A}$ proceeds as follows.

1. Given μ public keys $pk_1 = g^{x_1}, \ldots, pk_\mu = g^{x_\mu}$, $\mathcal{A}$ samples a uniformly random index $j^* \xleftarrow{\$} [\mu]$. Then it queries $\mathsf{RevLTK}(i)$ for all $i \in [\mu \setminus j^*]$ to obtain all private keys except for sk_{j^*}.
2. Next, $\mathcal{A}$ computes $sk_{j^*} = x_{j^*}$ from $pk_{j^*} = g^{x_{j^*}}$, e.g., by exhaustive search.[4]
3. Then $\mathcal{A}$ picks an arbitrary oracle, say π_s^1 for $s = (j^*+1) \bmod \mu$, and executes the protocol with π_s^1, impersonating user j^*. That is, $\mathcal{A}$ proceeds exactly as in the protocol specification, but on behalf of user j^*. Note that $\mathcal{A}$ it is able to compute all messages and the resulting session key on behalf of user j^*, because it "knows" sk_{j^*}.
4. Finally, $\mathcal{A}$ asks $\mathsf{Test}(s, 1)$. Note that this is a valid Test-query, as $\mathcal{A}$ has never asked any $\mathsf{RevSessKey}$-query or $\mathsf{RevLTK}(j^*)$ to the peer j^* of oracle π_s^1. If the experiment returns the "real" key, then $\mathcal{A}$ outputs "1". Otherwise it outputs "0".

[4] Note that we are considering an inefficient adversary here. As usual for meta-reductions, we will later describe how $\mathcal{A}$ can be simulated *efficiently*.

Note that $\mathcal{A}$ wins the security experiment with optimal success probability $1 - 1/|\mathcal{K}|$, where $|\mathcal{K}|$ is the size of the key space. The loss of $1/|\mathcal{K}|$ is due to the fact that the random key chosen by the Test-query may be equal to the actual session key.

Description of the meta-reduction. Meta-reduction $\mathcal{M}$ interacts with reduction $\mathcal{R}$ by simulating the hypothetical adversary $\mathcal{A}$ as follows.

1. $\mathcal{M}$ receives as input a CDH instance (g^x, g^y). It starts $\mathcal{R}$ on input (g^x, g^y).
2. Whenever $\mathcal{R}$ issues a query to oracle $\mathcal{O}$, $\mathcal{M}$ forwards it to its own oracle. Note that both oracles are equivalent, because $\mathcal{M}$ has simply forwarded the CDH instance.
3. When $\mathcal{R}$ outputs public keys $pk_1 = g^{x_1}, \ldots, pk_\mu = g^{x_\mu}$ to $\mathcal{A}$, $\mathcal{M}$ makes a snapshot of the current state $st_{\mathcal{R}}$ of $\mathcal{R}$.
4. For $j \in [1 \ldots \mu]$, $\mathcal{M}$ now proceeds as follows.
   (a) It lets $\mathcal{A}$ query $\mathsf{RevLTK}(i)$ for all $i \in [\mu \setminus j]$, in order to obtain all private keys except for sk_j. Note that the reduction may or may not respond to all $\mathsf{RevLTK}(i)$ queries. For instance, $\mathcal{R}$ may abort for certain queries.
   (b) Then it resets $\mathcal{R}$ to state $st_{\mathcal{R}}$.
5. Now $\mathcal{M}$ proceeds to simulate the hypothetical adversary. That is:
   (a) It picks a uniformly random index $j^* \xleftarrow{\$} [1 \ldots \mu]$ and queries $\mathsf{RevLTK}(i)$ for all $i \in [\mu \setminus j^*]$.
   (b) Then it executes the protocol with π_s^1, impersonating user j^*. Note that this works only if $\mathcal{M}$ was able to obtain sk_{j^*} in Step (4).
   (c) Finally, $\mathcal{M}$ lets $\mathcal{A}$ ask $\mathsf{Test}(s, 1)$. If the experiment returns the "real" key, then $\mathcal{A}$ outputs "1". Otherwise it outputs "0".
6. If $\mathcal{R}$ outputs some value h throughout the experiment, then $\mathcal{M}$ outputs the same value.

Note that $\mathcal{M}$ provides a perfect simulation of the hypothetical adversary, provided that it "learns" sk_{j^*} in the loop in Step (4).

Analysis of the meta-reduction. $\mathcal{M}$ essentially runs reduction $\mathcal{R}$ at most μ times. Apart from that, it performs only minor additional operations, such that we have $t_{\mathcal{M}} \approx \mu \cdot t_{\mathcal{R}}$.

In order to analyse the success probability of $\mathcal{M}$, let us say that bad occurs, if j^* is the *only* index for which $\mathcal{R}$ did not abort in Step (4) of the meta-reduction. Note that in this case $\mathcal{M}$ learns all private keys, *except* for sk_{j^*}, in which is the only case where the simulation of $\mathcal{A}$ in Step (5.b) fails. Since we may assume

without loss of generality that the reduction $\mathcal{R}$ works for at least one index $j \in [\mu]$ and we chose $j^* \xleftarrow{\$} [\mu]$ uniformly random, we have

$$\Pr[\mathsf{bad}] \leq \frac{1}{\mu}.$$

Let $\mathsf{win}(\mathcal{R}, \mathcal{A})$ denote the event that $\mathcal{R}$ outputs $h = g^{xy}$ when interacting with $\mathcal{A}$, and $\mathsf{win}(\mathcal{R}, \mathcal{M})$ the corresponding event with $\mathcal{M}$. Since $\mathcal{M}$ simulates $\mathcal{A}$ perfectly unless bad occurs, we have

$$|\Pr[\mathsf{win}(\mathcal{R}, \mathcal{A})] - \Pr[\mathsf{win}(\mathcal{R}, \mathcal{M})]| \leq \Pr[\mathsf{bad}].$$

Furthermore, note that by definition we have $\epsilon_{\mathcal{R}} = \Pr[\mathsf{win}(\mathcal{R}, \mathcal{A})]$ and $\epsilon_{\mathcal{M}} = \Pr[\mathsf{win}(\mathcal{R}, \mathcal{M})]$. Hence we get $|\epsilon_{\mathcal{R}} - \epsilon_{\mathcal{M}}| \leq 1/\mu$, which in turn yields the lower bound $\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - 1/\mu$. □

Generalizations. The tightness lower bound proven above makes several very specific assumptions about the considered protocols, hardness assumptions, and security models. The main purpose of this is to keep the formalization and proof focused on the type of protocols that we are considering in this paper. However, a natural question is to which extent the results also apply to more general protocols, models, and assumptions, and whether and how the tightness bound can be evaded by tweaking the considered setting.

First of all, we consider only protocols where long-term secrets are of the form $(pk, sk) = (g^x, x)$. As already briefly discussed above, one can generalize this to other protocols, as long as the simulation of the hypothetical adversary by the meta-reduction is able to recover properly distributed private keys. In particular, one can generalize to arbitrary *efficiently re-randomizable* long-term keys, as defined by Bader et al. [4]. Note that current AKE protocols with tight security proofs [3, 21] do *not* have efficiently rerandomizable keys, and therefore do not contradict our result.

In order to obtain a tighter security proof one may try to make different complexity assumptions. These can be either *non-interactive* (i.e., the reduction does not have access to an oracle $\mathcal{O}$, such as e.g. DDH), or stronger *interactive* assumptions. Let us first consider non-interactive assumptions. A very general class of such assumptions was defined abstractly in Bader et al. [4], and it is easy to verify that our proof works exactly the same way with such an abstract non-interactive assumption instead of CDH.

Some stronger assumptions may yield tight security proofs, but not all of them do. Consider for instance the *gap Diffie-Hellman* assumption, which is identical to strong Diffie-Hellman, except that the first input to the provided DDH-oracle is not fixed, but can be arbitrary. It is easy to verify that our proof also works for this assumption, in exactly the same way. More generally, our proof works immediately for any assumption for which the "winning condition" of the reduction is independent of the sequence of oracle queries issued by the reduction. An example of an interactive assumptions where this does *not* hold

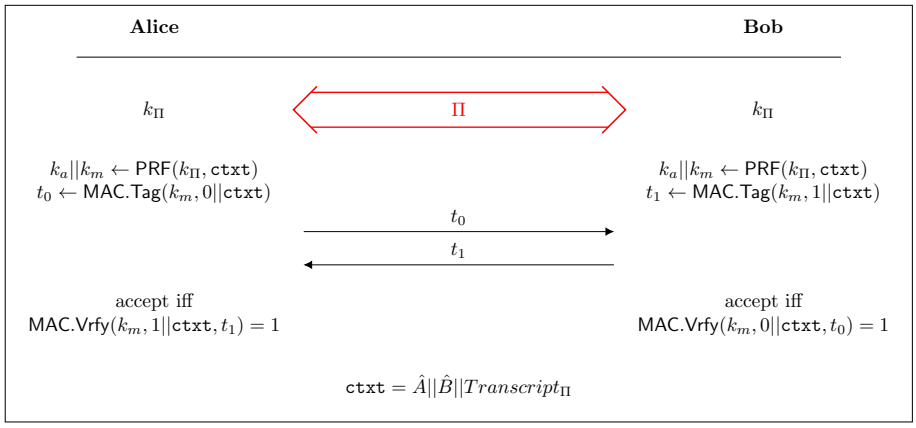

Figure 4: Generic compiler from an AKE protocol Π with implicit authentication to a protocol Π^+ with explicit entity authentication.

is the trivial interactive assumption that the protocol is secure (which, of course, immediately yields a tight security proof).

Finally, we note that our impossibility result holds also for many weaker or stronger AKE security models. We only require that the model allows for active attacks and provides a RevLTK query. Thus, the result immediately applies also to weaker models that, e.g., do not provide a RevSessKey-query or only a single Test-query, and trivially also for stronger models, such as eCK-style *ephemeral key reveals* [12, 14]. It remains an interesting open question whether stronger impossibility results (e.g., with quadratic lower bound) can be proven for such eCK-style definitions.

8 Adding Explicit Entity Authentication

In this section we describe how explicit entity authentication (EA) [11] can be added to our protocols by doing an additional key-confirmation step. Recall that the idea of EA is that there should only be two ways in which an attacker can get a session oracle to accept: (1) by passively forwarding the protocol messages between the session and its peer, or (2) by obtaining the peer's private long-term key and using that to impersonate the peer towards the session oracle. In other words, in the case of (1) there must exist some session oracle at the peer who created the messages the that accepting oracle received (and who also received the messages the accepting oracle sent).

Our construction for achieving EA is a generic compiler which transforms an arbitrary AKE protocol Π, secure according to Definition 9, into one that also provides EA (defined below). The details of the compiler are given in Fig. 4. Specifically, protocol Π^+ begins by running protocol Π to obtain a session key k_Π. This key, which we henceforth call the *intermediate key* for protocol Π^+, is then

used to derive two additional keys: k_a and k_m. The first key becomes the final session key of protocol Π^+, while k_m is used to compute a key-confirmation message, i.e., a MAC, for each party. The EA property of Π^+ reduces to the AKE security of the initial protocol Π, the *multi-user* PRF security of the function used to derive k_a and k_m, as well as the *multi-user strong UF-CMA* (mu-SUF-CMA) security of the MAC scheme. The formal definitions of mu-PRF security and mu-SUF-CMA security are given in Appendix C (see Fig. 5 and Fig. 6).

Our result is basically a restatement of the theorem proved by Yang [43], but with two minor differences: (1) our result is stated for arbitrary protocols and not only two-message protocols, and (2) since we use the AKE-RoR model the proof is tighter and slightly simpler.

Formal definition of explicit entity authentication. To formalize the idea of an adversary passively forwarding messages, we use the definition of *matching conversations* [11] (but without the temporal aspect). Note that there is an asymmetry in the definition depending on whether the session oracle sent or received the last message of the protocol.

Definition 11 (Matching conversations)**.** Let Π be an N-message two-party protocol in which all messages are sent sequentially.

- If a session oracle π_i^s sent the last message of the protocol, then π_j^t is said to have *matching conversations* to π_i if the first $N-1$ messages of π_i^s's transcript agrees with the first $N-1$ messages of π_j^t's transcript.
- If a session oracle π_i^s received the last message of the protocol, then π_j^t is said to have *matching conversations* to π_i if all N messages of π_i^s's transcript agrees with π_j^t's transcript.

EA can now be defined in terms of matching conversations as follows. Let $\mathrm{EA}_\Pi(\mu, \ell)$ be the same experiment as the AKE security game $G_\Pi(\mu, \ell)$ defined in Section 3, except that the adversary no longer has access to the Test query.

Definition 12 (Explicit entity authentication)**.** On game $\mathrm{EA}_\Pi(\mu, \ell)$ define $\mathsf{break}_{\mathrm{EA}}$ to be the event that there exists an oracle π_i^s for which all the following conditions are satisfied.

(i) π_i^s has accepted, that is, $\Psi_i^s = \texttt{accept}$.

(ii) $\mathsf{Pid}_i^s = j$ and party j is not corrupted.

(iii) There is no oracle π_j^t having:

  (a) matching conversations to π_i^s; *and*

  (b) $\mathsf{Pid}_j^t = i$; *and*

  (c) $\mathsf{role}_j^t \neq \mathsf{role}_i^s$.

Remark 4. Note that Items (iii.b) and (iii.c) in Definition 12 ensure that π_j^t not only has matching conversations to π_i^s (Item (iii.a)), but that π_i^s and π_j^t also agree on who they are talking to and which role they have in this protocol run.

Definition 13 (EA Security)**.** An attacker $\mathcal{A}$ breaks the explicit entity authentication (EA) of protocol Π if event $\mathsf{break}_{\mathrm{EA}}$ occurs in $\mathrm{EA}_\pi(\mu, \ell)$. The *EA-advantage* of adversary $\mathcal{A}$ against protocol Π is

$$\mathrm{Adv}_{\Pi}^{\mathsf{EA}}(\mathcal{A}) = \Pr\left[\mathsf{break}_{\mathrm{EA}}\right].$$

We say that $\mathcal{A}$ $(\epsilon_{\mathcal{A}}, t, \mu, \ell)$*-breaks* Π if its running time is t and $\mathrm{Adv}_{\Pi}^{\mathsf{EA}}(\mathcal{A}) \geq \epsilon_{\mathcal{A}}$. The running time of $\mathcal{A}$ includes the running time of the security experiment.

Theorem 6. *Let Π be an AKE protocol, let Π^+ be the protocol derived from Π as defined in Fig. 4, and let $\mathcal{A}$ be an adversary against the EA security of protocol Π^+. Then there exists adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{D}$, and $\mathcal{F}$, such that*

$$\begin{aligned}\mathrm{Adv}_{\Pi^+}^{\mathsf{EA}}(\mathcal{A}) \leq\ & \mathrm{Adv}_{\Pi}^{\mathsf{AKE}}(\mathcal{B}_1) + 2 \cdot \mathrm{Adv}_{\Pi}^{\mathsf{AKE}}(\mathcal{B}_2) \\ & + \mathrm{Adv}_{\mathsf{PRF},\mu\ell}^{\mathsf{mu\text{-}PRF}}(\mathcal{D}) + \mathrm{Adv}_{\mathsf{MAC},\mu\ell}^{\mathsf{mu\text{-}SUF\text{-}CMA}}(\mathcal{F}),\end{aligned} \tag{15}$$

where $\mu\ell$ is the number of sessions created by $\mathcal{A}$. The adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{D}$, and $\mathcal{F}$ all run in essentially the same time as $\mathcal{A}$.

The proof of Theorem 6 is given in Appendix C.

Remark 5. The PRF and MAC in Theorem 6 can be instantiated tightly in the multi-user setting, e.g. by using the simple and efficient AMAC construction [6].

9 Conclusion

We have showed that it is possible to achieve highly efficient AKE protocols that can be instantiated with theoretically sound parameters. Specifically, we gave protocol constructions that have only a linear tightness loss in the number of users, while using only a handful of exponentiations. Our constructions are at least as efficient as the best known AKE protocols in this setting. Perhaps surprisingly, our constructions only use standard building blocks as used by widely deployed protocols and are very similar to protocols like Noise-KK, and offer similar security guarantees.

While our proofs have a linear loss we have showed that this is actually unavoidable: any reduction from a protocol in our class to a wide class of hardness assumptions must lose a factor of at least μ. Thus, our reductions are optimal in this regard. Additionally, we proved that adding a key confirmation step tightly provides explicit authentication.

Taken together, these results demonstrate for the first time that AKE protocols can be instantiated in a theoretically sound way in real-world deployments without sacrificing performance.

References

[1] Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE password-authenticated key exchange protocol. In: 2015 IEEE Symposium on Security and Privacy. pp. 571–587. IEEE Computer Society Press, San Jose, CA, USA (May 17–21, 2015) 7

[2] Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, vol. 3386, pp. 65–84. Springer, Heidelberg, Germany, Les Diablerets, Switzerland (Jan 23–26, 2005) 7

[3] Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015: 12th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 9014, pp. 629–658. Springer, Heidelberg, Germany, Warsaw, Poland (Mar 23–25, 2015) 3, 5, 6, 8, 28

[4] Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 273–304. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016) 5, 24, 25, 26, 28

[5] Barthe, G., Crespo, J.M., Lakhnech, Y., Schmidt, B.: Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 689–718. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015) 24

[6] Bellare, M., Bernstein, D.J., Tessaro, S.: Hash-function based PRFs: AMAC and its multi-user security. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 566–595. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016) 31

[7] Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 259–274. Springer, Heidelberg, Germany, Bruges, Belgium (May 14–18, 2000) 7

[8] Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th Annual Symposium on Foundations of Computer Science. pp. 394–403. IEEE Computer Society Press, Miami Beach, Florida (Oct 19–22, 1997) 3

[9] Bellare, M., Goldreich, O., Mityagin, A.: The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309 (2004), http://eprint.iacr.org/2004/309 48

[10] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93: 1st Conference on Computer and Communications Security. pp. 62–73. ACM Press, Fairfax, Virginia, USA (Nov 3–5, 1993) 6

[11] Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 232–249. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994) 29, 30

[12] Bergsma, F., Jager, T., Schwenk, J.: One-round key exchange with strong security: An efficient and generic construction in the standard model. In: Katz, J. (ed.) PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 9020, pp. 477–494. Springer, Heidelberg, Germany, Gaithersburg, MD, USA (Mar 30 – Apr 1, 2015) 9, 29

[13] Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zanella Béguelin, S.: Proving the TLS handshake secure (as it is). In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part II. Lecture Notes in Computer Science, vol. 8617, pp. 235–255. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014) 3

[14] Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 453–474. Springer, Heidelberg, Germany, Innsbruck, Austria (May 6–10, 2001) 9, 29

[15] Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 127–145. Springer, Heidelberg, Germany, Istanbul, Turkey (Apr 13–17, 2008) 4, 7, 19, 20, 21

[16] Cohn-Gordon, K., Cremers, C.J.F., Garratt, L.: On post-compromise security. In: IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016. pp. 164–178. IEEE Computer Society (2016), https://doi.org/10.1109/CSF.2016.19 11

[17] Coron, J.S.: Optimal security proofs for PSS and other signature schemes. In: Knudsen, L.R. (ed.) Advances in Cryptology – EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 272–287. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28 – May 2, 2002) 24

[18] Donenfeld, J.A.: WireGuard: Next generation kernel network tunnel. In: ISOC Network and Distributed System Security Symposium – NDSS 2017. The Internet Society, San Diego, CA, USA (Feb 26 – Mar 1, 2017) 4

[19] Fleischhacker, N., Jager, T., Schröder, D.: On tight security proofs for Schnorr signatures. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 512–531. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014) 24

[20] Garg, S., Bhaskar, R., Lokam, S.V.: Improved bounds on security reductions for discrete log based signatures. In: Wagner, D. (ed.) Advances in Cryptology – CRYPTO 2008. Lecture Notes in Computer Science, vol. 5157, pp. 93–107. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2008) 24

[21] Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 95–125. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018) 3, 4, 6, 7, 8, 11, 20, 21, 22, 23, 24, 25, 28

[22] Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 415–432. Springer, Heidelberg, Germany, Istanbul, Turkey (Apr 13–17, 2008) 6

[23] Hofheinz, D., Jager, T., Knapp, E.: Waters signatures with optimal security reduction. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 7293, pp. 66–83. Springer, Heidelberg, Germany, Darmstadt, Germany (May 21–23, 2012) 24, 26

[24] Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 273–293. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012) 3

[25] Jager, T., Stam, M., Stanley-Oakes, R., Warinschi, B.: Multi-key authenticated encryption with corruptions: Reductions are lossy. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017: 15th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 10677, pp. 409–441. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017) 24

[26] Jeong, I.R., Katz, J., Lee, D.H.: One-round protocols for two-party authenticated key exchange. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS

04: 2nd International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 3089, pp. 220–232. Springer, Heidelberg, Germany, Yellow Mountain, China (Jun 8–11, 2004) 4, 22

[27] Kakvi, S.A., Kiltz, E.: Optimal security proofs for full domain hash, revisited. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 537–553. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012) 24

[28] Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 546–566. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005) 11, 22, 25

[29] Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 429–448. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013) 3

[30] Kudla, C., Paterson, K.G.: Modular security proofs for key agreement protocols. In: Roy, B.K. (ed.) Advances in Cryptology – ASIACRYPT 2005. Lecture Notes in Computer Science, vol. 3788, pp. 549–565. Springer, Heidelberg, Germany, Chennai, India (Dec 4–8, 2005) 22

[31] LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007: 1st International Conference on Provable Security. Lecture Notes in Computer Science, vol. 4784, pp. 1–16. Springer, Heidelberg, Germany, Wollongong, Australia (Nov 1–2, 2007) 22, 25

[32] Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 3958, pp. 378–394. Springer, Heidelberg, Germany, New York, NY, USA (Apr 24–26, 2006) 4, 5, 22, 25

[33] Lewko, A.B., Waters, B.: Why proving HIBE systems secure is difficult. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441, pp. 58–76. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014) 24

[34] Li, Y., Schäge, S.: No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th

Conference on Computer and Communications Security. pp. 1343–1360. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017) 8

[35] Menezes, A., Ustaoglu, B.: Security arguments for the UM key agreement protocol in the NIST SP 800-56A standard. In: Abe, M., Gligor, V. (eds.) ASIACCS 08: 3rd ACM Symposium on Information, Computer and Communications Security. pp. 261–270. ACM Press, Tokyo, Japan (Mar 18–20, 2008) 21, 22

[36] Morgan, A., Pass, R.: On the security loss of unique signatures. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018: 16th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 11239, pp. 507–536. Springer, Heidelberg, Germany, Panaji, India (Nov 11–14, 2018) 24

[37] Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: Roy, B.K. (ed.) Advances in Cryptology – ASIACRYPT 2005. Lecture Notes in Computer Science, vol. 3788, pp. 1–20. Springer, Heidelberg, Germany, Chennai, India (Dec 4–8, 2005) 24

[38] Perrin, T.: Noise protocol framework (2018), http://noiseprotocol.org 4, 12

[39] Seurin, Y.: On the exact security of Schnorr-type signatures in the random oracle model. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 554–571. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012) 24

[40] Signal Messenger: Technical information (2018), https://signal.org/docs 4

[41] Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Des. Codes Cryptography 46(3), 329–342 (2008) 22

[42] Wang, Y., Matsuda, T., Hanaoka, G., Tanaka, K.: Memory lower bounds of reductions revisited. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part I. Lecture Notes in Computer Science, vol. 10820, pp. 61–90. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018) 24

[43] Yang, Z.: Modelling simultaneous mutual authentication for authenticated key exchange. In: Danger, J., Debbabi, M., Marion, J., García-Alfaro, J., Zincir-Heywood, A.N. (eds.) Foundations and Practice of Security - 6th International Symposium, FPS 2013, La Rochelle, France, October 21-22, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8352, pp. 46–62. Springer (2013), https://doi.org/10.1007/978-3-319-05302-8_4 5, 30

A Proof of Protocol Π_{Twin}

Here we prove Theorem 3. The proof structure is basically identical to that of Theorem 2. The only difference is that where we in the proof of Theorem 2 used the *strong* Diffie-Hellman oracle to ensure consistency between partnered session oracles, we now use the *twin* Diffie-Hellman oracle. More specifically, the general technique is again to have our session oracles refrain from computing the input to the key derivation function (i.e., the RO), but instead check to see if the adversary somehow computes it. Namely, the reduction can check if the adversary ever submits the correct input to the RO by using the twin DH oracle provided. In the proof of Theorem 2 we used the strong DH oracle for this check, but the additional "twinned" DH values in Π_{Twin} now allows us to get away with a twin DH oracle instead.

The following the game hops are all the same as those in the proof of Theorem 2, only adjusted to account for the extra twinned DH values used in protocol Π_{Twin} compared to protocol Π. Recall that a session oracle is called *honest* (at some point) if the user it belongs to has not yet been corrupted (at that point). Again, there are five types of oracles that we will have to deal with in separate ways, repeated below.

- (I) initiator oracles whose response message comes from a responder oracle, which has the same term `ctxt` and which is honest when the response is received;
- (II) other initiator oracles whose intended peer is honest until the oracle accepts;
- (III) responder oracles whose initial message comes from an initiator, which has the same `ctxt` (up to the responder message) and which is honest when the response is received;
- (IV) other responder oracles whose intended peer is honest until the oracle accepts; and
- (V) oracles whose intended peer is corrupted.

Note that at the time an initiator oracle starts, we cannot know if it will be of type I or II. However, we will know what type it is when it is time to compute the oracle's session key.

In the following, let S_j denote the event that the experiment in Game j outputs 1.

Game 0. The is original AKE security experiment, hence

$$\Pr\left[\mathsf{break}_{\text{KE}}\right] = \Pr[S_0]. \tag{16}$$

Game 1. In this game, we abort if two initiator oracles or two responder oracles ever arrive at the same `ctxt`, hence

$$|\Pr[S_1] - \Pr[S_0]| \leq \frac{\mu\ell^2}{p}. \tag{17}$$

A.1 Preparing Oracles

Game 2. In this game, we modify how our oracles determine their session keys. For a type III, IV, or V responder oracle with `ctxt` prefixed by $\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V$, having private key (b_0, b_1) and random exponent s, and where the intended initiator peer's public key is (A_0, A_1), the game now does the following to determine the responder's session key k. First, it checks to see if any oracle queries $\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V\|W_1\|\ldots\|W_6$ have been made satisfying

$$\begin{array}{lll} W_1 = A_0^s & W_3 = U_0^{b_0} & W_5 = U_0^s \\ W_2 = A_1^s & W_4 = U_0^{b_1} & W_6 = U_1^s \end{array} \tag{18}$$

If any such query is found, k is set to the corresponding hash value. Otherwise, the session key is chosen at random. And if such a hash query happens later, the hash value is set to the chosen session key.

A type I initiator oracle will simply use the key from the corresponding responder oracle.

For a type II or V initiator oracle with `ctxt` prefixed by $\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V$, having private key (a_0, a_1) and random exponents (r_0, r_1), and where the responder public key is (B_0, B_1), the game does the following to determine its session key k. First, it checks to see if any oracle queries $\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V\|W_1\|\ldots\|W_6$ have been made satisfying

$$\begin{array}{lll} W_1 = V^{a_0} & W_3 = B_0^{r_0} & W_5 = V^{r_0} \\ W_2 = V^{a_1} & W_4 = B_1^{r_0} & W_6 = V^{r_1} \end{array} \tag{19}$$

If any such query is found, k is set to the corresponding hash value. Otherwise, the session key is chosen at random. And if such a hash query happens later, the hash value is set to the chosen session key.

The only potential change in this game is at which point in time the key derivation hash oracle value is first defined, which is unobservable. It follows that

$$\Pr[S_2] = \Pr[S_1]. \tag{20}$$

A.2 Type IV Responder Oracles

Game 3. In this game type IV oracles choose their session key at random, but do not modify the hash oracle unless the intended peer is corrupted. If the adversary corrupts the intended peer i (with public key $pk_i = (A_0, A_1)$) of a type IV oracle running as user j with private key $sk_j = (b_0, b_1)$, random

exponent s, and chosen key k, then from that point in time, any query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V\|A_0^s\|A_1^s\|U_0^{b_0}\|U_0^{b_1}\|U_0^s\|U_1^s$$

to the key derivation hash oracle H will result in the hash value k.

Unless one of these queries happen before user i is corrupted, the only change is at which point in time the key derivation hash oracle value is first defined, which is unobservable. Let F be the event that a query as above happens before the corresponding long-term key is corrupted. Then

$$|\Pr[S_3] - \Pr[S_2]| \leq \Pr[F].$$

Let F_i be the same event as F, but with the intended peer being user i. We then have that $\Pr[F] = \sum_i \Pr[F_i]$.

Next, consider the event E_i which is that for some type IV oracle as above, any query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|U\|V\|W_1\|\ldots\|W_6 \quad W_1 = A_0^s = V^{a_0} \quad W_2 = A_1^s = V^{a_1} \tag{21}$$

to the key derivation hash oracle H happens before user i is corrupted. Then $\Pr[F_i] \leq \Pr[E_i]$.

We shall now bound the probability of the event E_i by constructing an adversary against strong twin Diffie-Hellman. This adversary will embed its twin DH challenge in some user i's public key and type IV oracle responses for oracles whose intended peer is user i. This will allow it to recover the solution to its DH challenge from the hash query in event E_i.

Strong Twin Diffie-Hellman adversary $\mathcal{B}_1$. The algorithm $\mathcal{B}_1$ takes as input a twin DH challenge $(X_0, X_1, Y) = (g^{x_0}, g^{x_1}, g^y)$ and outputs a pair of group elements (Z_0, Z_1). It has access to a twin Diffie-Hellman oracle $\mathsf{twinDH}_{x_0,x_1}(\cdot,\cdot,\cdot)$.

Reduction $\mathcal{B}_1$ runs Game 2 with the following changes: it chooses i uniformly at random and sets user i's public key to $pk_i = (X_0, X_1)$ (and thus implicitly sets i's private key to the unknown value (x_0, x_1)). For type IV oracles whose intended peer is user i, $\mathcal{B}_1$ sets $V = Y \cdot g^\rho$, with ρ random. If the adversary corrupts user i, the reduction $\mathcal{B}_1$ aborts. For other users, the reduction simply returns the private key, as in Game 2.

We need to recognise hash queries of the form (18) and (19) that involve user i, as well as queries of the form (21). For (18), where user i acts in the responder role, we know the oracle's random exponent s, so we only need to recognise if (W_3, W_4) is U_0 raised to user i's private key, which can be done by checking if $\mathsf{twinDH}_{x_0,x_1}(U_0, W_3, W_4) = 1$.

For (19), where user i is the initiator, we know the oracle's random exponents (r_0, r_1), so we only need to recognise if (W_1, W_2) is V raised to user i's private key, which can be done by checking if $\mathsf{twinDH}_{x_0,x_1}(V, W_1, W_2) = 1$.

Finally, for (21), we need to recognise if group elements (W_1, W_2) is V raised to user i's private key, which can be done by checking if $\mathsf{twinDH}_{x_0,x_1}(V, W_1, W_2) = 1$.

When we recognise a query of the form (21), since we know that $V = Y \cdot g^{\rho}$, we output

$$Z_0 = W_1 X_0^{-\rho} = V^{x_0} X_0^{-\rho} = Y^{x_0} g^{\rho x_0} g^{-x_0 \rho} = Y^{x_0}$$
$$Z_1 = W_2 X_1^{-\rho} = V^{x_1} X_1^{-\rho} = Y^{x_1} g^{\rho x_1} g^{-x_1 \rho} = Y^{x_1}$$

In other words, our adversary $\mathcal{B}_1$ succeeds whenever E_i would happen in Game 2. Furthermore, E_i in Game 2 can only happen before user i is corrupted, so whenever E_i would happen in Game 2, $\mathcal{B}_1$ would not have aborted.

We get that

$$\mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_1) \geq \frac{1}{\mu} \sum_i \Pr[E_i] \geq \frac{1}{\mu} \sum_i \Pr[F_i] = \frac{1}{\mu} \Pr[F],$$

from which it follows that

$$|\Pr[S_3] - \Pr[S_2]| \leq \Pr[F] \leq \mu \cdot \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_1). \tag{22}$$

A.3 Type III Responder Oracles

Game 4. In this game type III responder oracles choose their session key at random, and do not modify the key derivation hash oracle.

Consider a type III responder oracle for user j with private key (b_0, b_1), random exponent s and intended peer i, who has private key (a_0, a_1). Unless the adversary ever makes a hash query of the form

$$\hat{i} \| \hat{j} \| pk_i \| pk_j \| U_0 \| U_1 \| V \| W_1 \| \ldots \| W_6 \qquad W_5 = U_0^s, \qquad W_6 = U_1^s, \tag{23}$$

this change is unobservable. Call this event F. We thus have

$$|\Pr[S_4] - \Pr[S_3]| \leq \Pr[F]. \tag{24}$$

We will bound the probability of F by constructing an adversary against strong twin Diffie-Hellman. This adversary will embed its challenge in type I or II initiator oracles' message, as well as in type III responder oracles' message. It will recover the solution to its twin DH challenge from the hash query in event F.

Strong Twin Diffie-Hellman adversary $\mathcal{B}_2$. The algorithm $\mathcal{B}_2$ takes as input a twin DH challenge $(X_0, X_1, Y) = (g^{x_0}, g^{x_1}, g^y)$ and outputs a pair of group elements (Z_0, Z_1). It has access to a twin DH-oracle $\mathsf{twinDH}_{x_0,x_1}(\cdot,\cdot,\cdot)$.

Our reduction $\mathcal{B}_2$ runs Game 3 with the following changes: for type I and II initiator oracles (we cannot distinguish these at this point in time), it computes $U_0 = X_0 g^{\rho_0}$ and $U_1 = X_1 g^{\rho_1}$, with ρ_0, ρ_1 random. For type III responder oracles, it computes $V = Y \cdot g^{\rho_2}$, with ρ_2 random. Note that the reduction knows all static private keys, so user corruption is handled exactly as in Game 3.

We need to recognise hash queries of the form (19) for type II initiator oracles, as well as queries of the form (23) for type III oracles. Although we do not know

the oracles' random exponents, we do know their secret keys. This means that we only need to recognise if W_5 is V raised to $\log_g U_0 = x_0 + \rho_0$, as well as the corresponding relations for W_6. Of course, if $W_5 = V^{x_0+\rho_0}$ and $W_6 = V^{x_1+\rho_1}$, then $W_5 V^{-\rho_0} = V^{x_0}$ and $W_6 V^{-\rho_1} = V^{x_1}$, which we can detect by checking if $\mathsf{twinDH}_{x_0,x_1}(V, W_5 V^{-\rho_0}, W_5 V^{-\rho_1}) = 1$. If this is the case for a query of the form (23), then we output

$$Z_0 = W_5 V^{-\rho_0} X_0^{-\rho_2} = V^{x_0} X_0^{-\rho_2} = g^{(y+\rho_2)x_0 - x_0\rho_2} = Y^{x_0}$$
$$Z_1 = W_6 V^{-\rho_0} X_1^{-\rho_2} = V^{x_1} X_1^{-\rho_2} = g^{(y+\rho_2)x_1 - x_1\rho_2} = Y^{x_1}$$

as the solution to the twin DH challenge. In other words, $\mathcal{B}_2$ succeeds whenever F would happen in Game 3, hence

$$|\Pr[S_4] - \Pr[S_3]| \leq \Pr[F] \leq \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_2). \tag{25}$$

Note that we do not stop the simulation in the case we detect a hash query of the form (19) for a type II initiator oracle, because in this case the responder message V does not contain the embedded twin DH challenge.

A.4 Type II Initiator Oracles

Game 5. In this game type II initiator oracles choose their session key at random, but do not modify the hash oracle unless the intended peer is corrupted. If, for a type II oracle π_i^d having private key $sk_i = (a_0, a_1)$, random exponents (r_0, r_1), and chosen key k, the adversary corrupts π_i^d's intended peer j, having public key $pk_j = (B_0, B_1)$, then from that point in time any H-query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V\|V^{a_0}\|V^{a_1}\|B_0^{r_0}\|B_1^{r_0}\|V^{r_0}\|V^{r_1}$$

will be answered with the value k.

Unless such a query happens before user j is corrupted, the only change is at which point in time the key derivation hash oracle value is first defined, which is unobservable. Let F be the event that a query as above happens before the corresponding long-term key is corrupted. Then

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr[F].$$

Let F_j be the same event as F, but with the intended peer being user j. We then have that $\Pr[F] = \sum_j \Pr[F_j]$.

Next, consider the event E_j which is that for some type II oracle as above, any H-query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|U_0\|U_1\|V\|W_1\|\ldots\|W_6, \quad W_3 = B_0^{r_0} = U_0^{b_0}, \quad W_4 = B_1^{r_0} = U_0^{b_1} \tag{26}$$

happens before user j is corrupted. Then $\Pr[F_j] \leq \Pr[E_j]$.

We shall now bound the probability of the event E_j by constructing an adversary against strong twin Diffie-Hellman. This adversary will embed its twin DH challenge in some user j's public key and type II oracle messages for oracles whose intended peer is user j, and recover the solution to its twin DH challenge from the hash query in event E_j.

Strong Twin Diffie-Hellman adversary $\mathcal{B}_3$. The algorithm $\mathcal{B}_3$ takes as input a twin DH challenge $(X_0, X_1, Y) = (g^{x_0}, g^{x_1}, g^y)$ and outputs a pair of group elements (Z_0, Z_1). It has access to a twin DH-oracle $\mathsf{twinDH}_{x_0,x_1}(\cdot,\cdot,\cdot)$.

Our reduction $\mathcal{B}_3$ runs Game 4 with the following changes: It chooses j uniformly at random and sets user j's public key to $pk_j = (X_0, X_1)$ (and thus implicitly sets j's private key to the unknown value (x_0, x_1)). For type I and II initiator oracles whose intended peer is user j, $\mathcal{B}_3$ sets $U_0 = Y \cdot g^\rho$, with ρ random. If the adversary corrupts user j, the reduction $\mathcal{B}_3$ aborts. For other users, the reduction simply returns the private key, as in Game 4.

We need to recognise hash queries of the form (18) and (19) that involve user j, as well as queries of the form (26). For (18), where user j is the responder, we know the oracle's random exponent s, so we only need to recognise if (W_3, W_4) is U_0 raised to user j's private key, which can be done by checking if $\mathsf{twinDH}_{x_0,x_1}(U_0, W_3, W_4) = 1$. For (19), where user j is the initiator, we know the oracle's random exponents (r_0, r_1), so we only need to recognise if (W_1, W_2) is V raised to user j's private key, which can be done by checking if $\mathsf{twinDH}_{x_0,x_1}(V, W_1, W_2) = 1$. Finally, for (26), we need to recognise if a pair of group elements (W_3, W_4) is U_0 raised to user j's private key, which can be done by checking if $\mathsf{twinDH}_{x_0,x_1}(U_0, W_3, W_4) = 1$.

When we recognise a query of the form (26), meaning that $W_3 = U_0^{x_0}$ and $W_4 = U_0^{x_1}$ where we know that $U_0 = Y \cdot g^\rho$, then we output

$$Z_0 = W_3 X_0^{-\rho} = U_0^{x_0} X_0^{-\rho} = Y^{x_0}$$
$$Z_1 = W_4 X_1^{-\rho} = U_0^{x_1} X_1^{-\rho} = Y^{x_1}$$

In other words, our adversary $\mathcal{B}_3$ succeeds whenever E_j would happen in Game 4. Furthermore, E_j in Game 4 can only happen before user j is corrupted, so whenever E_j would happen in Game 4, $\mathcal{B}_3$ would not have aborted. We get that

$$\mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_3) \geq \frac{1}{\mu} \sum_j \Pr[E_j] \geq \frac{1}{\mu} \sum_j \Pr[F_j] = \frac{1}{\mu} \Pr[F],$$

from which it follows that

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr[F] \leq \mu \cdot \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_3). \tag{27}$$

A.5 Summary

Note that in Game 5, every session key is chosen at random independent of every key and sent message.

For type V oracles, the key derivation hash oracle is immediately programmed so that the session key is available to the adversary. But type V oracles are not fresh and therefore never subject to a Test query.

For type II and IV oracles, the key derivation hash oracle is programmed to make the session key available to the adversary only after the intended peer is corrupted. But if the intended peer is corrupted, a type II or IV oracle will

become non-fresh, which means that it should never have been subject to a Test query.

For type I and III oracles, the key derivation hash oracle will never make the session key available to the adversary.

This means that for any oracle subject to a Test query, the session key is and will remain independent of every key and sent message. Thus, the adversary cannot distinguish the session key from a random key, so

$$\Pr[S_5] = \frac{1}{2}. \tag{28}$$

Theorem 3 now follows from (16), (17), (20), (22), (25), (27) and (28). (Recall that Game 1 gives us $\Pr[\mathsf{break}_{\mathrm{Unique}}] \leq \mu\ell^2/p$, and that protocol Π_{Twin} has perfect correctness so $\Pr[\mathsf{break}_{\mathrm{Sound}}] = 0$.)

B Proof of Protocol Π_{Com}

Here we prove Theorem 4. The proof is very similar to the proofs of Theorems 2 and 3. The main difference is that twinning the ephemeral-ephemeral part of the key exchange is replaced by mutually committing to the Diffie-Hellman messages before sending them, which allows us to get a tight proof.

Recall that an oracle is called *honest* (at some point) if the user it belongs to has not yet been corrupted (at that point). Also, there are again five types of oracles that we will have to deal with in separate ways, repeated below.

- (I) initiator oracles whose response message comes from a responder oracle, which has the same term `ctxt` and which is honest when the response is received;
- (II) other initiator oracles whose intended peer is honest until the oracle accepts;
- (III) responder oracles whose initial message comes from an initiator, which has the same `ctxt` (up to the responder message) and which is honest when the response is received;
- (IV) other responder oracles whose intended peer is honest until the oracle accepts; and
- (V) oracles whose intended peer is corrupted.

Note that at the time an initiator oracle starts, we cannot know if it will be of type I or II. However, we will know what type it is when it is time to compute the oracle's session key.

It is also worth mentioning that for any two oracles that accept, if their transcripts agree on the first two messages (the commitments), they will also agree on the final two messages.

In the following, let S_j denote the event that the experiment in Game j outputs 1. We call the adversary's queries to the key derivation function H for H-queries, and its queries to the commitment hash function G for G-queries.

Game 0. The is original AKE security experiment, hence

$$\Pr\left[\mathsf{break}_{\mathrm{KE}}\right] = \Pr[S_0]. \tag{29}$$

We begin with an administrative step to avoid pathologies where honest players choose the same random random exponents.

Game 1. In this game, we abort if two initiator oracles or two responder oracles ever arrive at the same `ctxt`, hence

$$|\Pr[S_1] - \Pr[S_0]| \leq \frac{\mu\ell^2}{p}. \tag{30}$$

B.1 Preparing Oracles

Game 2. In this game, we modify how our oracles determine their session keys. For a type III, IV, or V responder oracle with `ctxt` prefixed by $\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V$, having private key (b_0, b_1) and random exponent s, and where the initiator public key is (A_0, A_1), the game does following to determine its session key k. First, it checks to see if any H-queries $\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V\|W_1\|\ldots\|W_5$ have been made satisfying

$$\begin{array}{lll} W_1 = A_0^s & W_3 = U^{b_0} & \\ W_2 = A_1^s & W_4 = U^{b_1} & W_5 = U^s \end{array} \tag{31}$$

If any such query is found k is set to the corresponding hash value. Otherwise, the session key is chosen at random. And if such a hash query happens later, the hash value is set to the chosen session key.

A type I initiator oracle will simply use the key from the corresponding responder oracle.

For a type II or V initiator oracle with `ctxt` prefixed by $\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V$, having private key (a_0, a_1) and random exponent r, and where the responder public key is (B_0, B_1), the game does the following to determine its session key k. First, it checks to see if any H-queries of the form $\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V\|W_1\|\ldots\|W_5$ have been made satisfying

$$\begin{array}{lll} W_1 = V^{a_0} & W_3 = B_0^{r_0} & \\ W_2 = V^{a_1} & W_4 = B_1^{r_0} & W_5 = V^r \end{array} \tag{32}$$

If any such query is found, k is set to the corresponding hash value. Otherwise, the session key is chosen at random. And if such a H-query happens later, the hash value is set to the chosen session key.

The only potential change in this game is at which point in time the key derivation hash oracle value is first defined, which is unobservable. It follows that

$$\Pr[S_2] = \Pr[S_1]. \tag{33}$$

B.2 Type IV Responder Oracles

Game 3. In this game type IV oracles choose their session key at random, but do not modify the hash oracle unless the intended peer is corrupted. If the adversary corrupts the intended peer i (with public key $pk_i = (A_0, A_1)$) of a type IV oracle running as user j with private key $sk_j = (b_0, b_1)$, random exponent s, and chosen key k, then from that point in time, any H-query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V\|A_0^s\|A_1^s\|U^{b_0}\|U^{b_1}\|U^s$$

will result in the value k.

Unless one of these H-queries happen before user i is corrupted, the only change is at which point in time the RO output is first defined, which is unobservable. Let F be the event that an H-query as above happens before the corresponding long-term key is corrupted. Using exactly the same reduction $\mathcal{B}_1$ as in the proof of Game 3, Theorem 3, we have

$$|\Pr[S_3] - \Pr[S_2]| \leq \Pr[F] \leq \mu \cdot \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_1). \tag{34}$$

B.3 Type III Responder Oracles

Game 4. In this game, we prepare for dealing with type III responder oracles by handling a potential problem with reprogramming the commitment hash G.

In this game, the oracles do not hash their message U or V using G. Instead, they choose a random hash value C_A or C_B as its commitment value. When it receives a response, G is reprogrammed such that $G(U) = C_A$ or $G(V) = C_B$.

The reprogramming will fail only if the adversary makes a G-query on either U or V before reprogramming. We can upper-bound the number of random oracle queries by the adversary's run time t, and we get

$$|\Pr[S_4] - \Pr[S_3]| \leq \frac{\mu \ell t}{p}. \tag{35}$$

Game 5. In this game type III responder oracles choose their session key at random, and do not modify the key derivation hash oracle.

Consider a type III responder oracle at user j having ephemeral exponent s. Unless the adversary ever makes an H-query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V\|W_1\|\ldots\|W_5 \qquad W_5 = U^s, \tag{36}$$

this change is unobservable. Call this event F. We thus have

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr[F]. \tag{37}$$

We shall bound the probability of F by constructing an adversary $\mathcal{B}_2$ against decisional Diffie-Hellman. Adversary $\mathcal{B}_2$ will embed its challenge in type I oracles' messages, as well as in type III responder oracles' messages. Unlike the

corresponding reductions for protocols Π and Π_{Twin}, we can avoid embedding the challenge in type II initiator oracles because the commitment messages in protocol Π_{Com} allow us to postpone choosing the messages until we know whether the oracle can be of type II.

Decisional Diffie-Hellman adversary $\mathcal{B}_2$. The algorithm $\mathcal{B}_2$ takes as input a DDH challenge (X, Y, Z) and outputs a bit. It simulates Game 4 as follows. For a type III responder oracle that receives a group element U, $\mathcal{B}_2$ will create a rerandomized tuple (X', Y', Z') and set its responding message V to be Y'. When computing its agreed key, it will use Z' as W_5 and compute W_1, W_2, W_3, W_4 using the corresponding private keys. In this computation, it will use X'. (If multiple type III responder oracles receive the same C_A, they will use rerandomized tuples with the same X'.)

When an initiator oracle receives its response commitment and that commitment was sent by a type III responder oracle with the same `ctxt` value, we know that the initiator oracle will be a type I oracle if it accepts. In this case, $\mathcal{B}_2$ will discard the initiator oracle's chosen message and instead use X' from the corresponding type III responder oracle.

Note that the reduction knows all static private keys, so user corruption is handled exactly as in Game 4. Also, type II, IV, and V oracles can be handled as in the previous game.

If a query of the form (36) every happens, algorithm $\mathcal{B}_2$ stops and outputs 1. Otherwise it outputs 0.

It is now clear that if the DDH challenge is a DDH tuple, $\mathcal{B}_2$ perfectly simulates Game 4 until the H-query happens. If the input is a random tuple, however, the probability that the adversary makes the H-query can be upper-bounded by $\mu\ell t/p$. It follows that

$$|\text{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{B}_2) - \Pr[F]| \leq \mu\ell t/p.$$

which means that

$$|\Pr[S_5] - \Pr[S_4]| \leq \Pr[F] \leq \text{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{B}_2) + \frac{\mu\ell t}{p}. \tag{38}$$

Note that we do not stop the simulation in the case we detect an H-query of the form (32) for a type II initiator oracle, because in this case the responder message V does not contain the embedded DDH challenge.

B.4 Type II Initiator Oracles

Game 6. In this game type II initiator oracles choose their session key at random, but do not modify the RO H unless the intended peer is corrupted. If, for a type II oracle π_i^d having private key $sk_i = (a_0, a_1)$, random exponent r, and chosen key k, the adversary corrupts π_i^d's intended peer j, having public key $pk_j = (B_0, B_1)$, then from that point in time any H-query of the form

$$\hat{i}\|\hat{j}\|pk_i\|pk_j\|C_A\|C_B\|U\|V\|V^{a_0}\|V^{a_1}\|B_0^r\|B_1^r\|V^r$$

will result in the value k.

Unless one of these queries happen before the user j is corrupted, the only change is at which point in time the key derivation hash oracle value is first defined, which is unobservable. Let F be the event that a query as above happens before the corresponding long-term key is corrupted. Using exactly the same reduction $\mathcal{B}_3$ as in the proof of Game 5, Theorem 3, we have

$$|\Pr[S_6] - \Pr[S_5]| \leq \Pr[F] \leq \mu \cdot \mathrm{Adv}^{\mathsf{2\text{-}CDH}}_{\mathbb{G},g}(\mathcal{B}_3). \tag{39}$$

B.5 Summary

Note that in Game 6, every session key is chosen at random independent of every key and sent message.

For type V oracles, the key derivation hash oracle is immediately programmed so that the session key is available to the adversary. But type V oracles are never fresh and therefore never subject to a Test query.

For type II and IV oracles, the key derivation hash oracle is programmed to make the session key available to the adversary only after the intended peer is corrupted. However, if the intended peer is corrupted, a type II or IV oracle will become non-fresh, which means that it should never have been subject to a Test query.

For type I and III oracles, the key derivation hash oracle will never make the session key available to the adversary.

This means that for any oracle subject to a Test query, the session key is and will remain independent of every key and sent message. Which means that the adversary cannot distinguish the session key from a random key. It follows that

$$\Pr[S_6] = \frac{1}{2}. \tag{40}$$

Theorem 4 now follows from (29), (30), (33), (34), (35), (38), (39) and (40). (Recall that Game 1 gives us $\Pr[\mathsf{break}_{\mathrm{Unique}}] \leq \mu\ell^2/p$, and that protocol Π_{Com} has perfect correctness so $\Pr[\mathsf{break}_{\mathrm{Sound}}] = 0$.)

C Proof of Key-Confirmation Compiler

Here we prove Theorem 6. First we formally define the mu-PRF and mu-SUF-CMA security notions.

mu-PRF security. Let $\mathsf{FUNC}(D, R)$ denote the set of all functions from $\{0,1\}^D$ to $\{0,1\}^R$. The multi-user PRF security of a function $F\colon \{0,1\}^\kappa \times \{0,1\}^D \to \{0,1\}^R$ is defined by the games $\textsc{Dist}^0_F$ and $\textsc{Dist}^1_F$ described by the parameterized game $\textsc{Dist}^b_F$ shown in Fig. 5. The mu-PRF advantage of an adversary $\mathcal{A}$ is

$$\mathrm{Adv}^{\mathsf{mu\text{-}PRF}}_{F,n}(\mathcal{A}) = \left|\Pr[\textsc{Dist}^0_{F,n}(\mathcal{A}) \Rightarrow 0] - \Pr[\textsc{Dist}^1_{F,n}(\mathcal{A}) \Rightarrow 0]\right|. \tag{41}$$

$\underline{\text{GAME DIST}^{b}_{F,n}(\mathcal{A})}$

1: **for all** $i \in [n]$ **do**
2: $\quad K_i \xleftarrow{\$} \{0,1\}^\kappa$
3: $\quad F_i^0 \leftarrow F(K_i, \cdot)$
4: $\quad F_i^1 \leftarrow \mathsf{FUNC}(D, R)$
5: $b' \leftarrow \mathcal{A}^{\{F_i^b(\cdot)\}}$
6: **return** b'

Figure 5: Experiment defining mu-PRF security.

mu-SUF-CMA security. A MAC scheme is a pair of algorithms $\mathsf{MAC} = (\mathsf{Tag}, \mathsf{Vrfy})$, where:

- $\mathsf{Tag}\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\tau$ is a *tag-generation* algorithm;
- $\mathsf{Vrfy}\colon \{0,1\}^\kappa \times \{0,1\}^* \times \{0,1\}^\tau \to \{0,1\}$ is a *verification* algorithm;

such that for all $K \in \{0,1\}^\kappa$, and all $m \in \{0,1\}^*$, $\mathsf{Vrfy}(K, m, \mathsf{Tag}(K, m)) = 1$. The multi-user SUF-CMA security of a MAC is defined by the $\text{FORGE}_{\mathsf{MAC}}$ game shown in Fig. 6. The mu-SUF-CMA advantage of an adversary $\mathcal{A}$ is

$$\mathrm{Adv}^{\mathsf{mu\text{-}SUF\text{-}CMA}}_{\mathsf{MAC},n}(\mathcal{A}) = \Pr[\text{FORGE}_{\mathsf{MAC}(\mathcal{A}),n} \Rightarrow 1]. \tag{42}$$

Remark 6. Our (mu-)SUF-CMA game is somewhat non-standard since the adversary can make multiple verification queries instead of just one at the end of the game (which is the normal formulation of (mu-)SUF-CMA). Bellare et al. [9] showed that this multi-query variant of UF-CMA is in general *not* equivalent to the standard 1-query definition, but for *strong* UF-CMA it is. Unfortunately, the reduction from multi-query SUF-CMA to 1-query SUF-CMA is not tight in general. But for PRF-based MACs—which many MACs are—it is.

$\underline{\text{GAME FORGE}_{\mathsf{MAC},n}(\mathcal{A})}$

1: $\mathsf{forgery} \leftarrow 0$
2: **for all** $i \in [n]$ **do**
3: $\quad K_i \xleftarrow{\$} \{0,1\}^\kappa$
4: $\quad T_i \leftarrow \emptyset$
5: $\mathcal{A}^{\text{TAG},\text{VRFY}}$
6: **return** $\mathsf{forgery}$

$\underline{\text{TAG}(i, m)}$

1: $t \leftarrow \mathsf{MAC.Tag}(K_i, m)$
2: $T_i \leftarrow T_i \cup \{(m, t)\}$
3: **return** t

$\underline{\text{VRFY}(i, m, t)}$:

1: $d \leftarrow \mathsf{MAC.Vrfy}(K_i, m, t)$
2: **if** $d = 1 \wedge (m, t) \notin T_i$ **then**
3: $\quad \mathsf{forgery} \leftarrow 1$
4: **return** d

Figure 6: Experiment defining mu-SUF-CMA security.

C.1 Proof of Theorem 6

In the following let S_i denote the event that $\mathsf{break}_{\mathrm{EA}}$ occurred in Game i.

Game 0. This is the original $\mathrm{EA}_{\Pi^+}(\mu, \ell)$ experiment, hence

$$\Pr\left[\mathsf{break}_{\mathrm{EA}}\right] = \Pr[S_0]. \tag{43}$$

Game 1. This game proceeds like in Game 0, but it aborts if either event $\mathsf{break}_{\mathrm{Sound}}$ or event $\mathsf{break}_{\mathrm{Unique}}$ occurs in protocol Π. There exists an adversary $\mathcal{B}_1$ such that:

$$|\Pr[S_1] - \Pr[S_0]| \leq \mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{B}_1). \tag{44}$$

Proof of (44). Let $E = \mathsf{break}_{\mathrm{Sound}} \vee \mathsf{break}_{\mathrm{Unique}}$. Clearly Game 0 and Game 1 are identical unless event E occurs, hence

$$|\Pr[S_1] - \Pr[S_0]| \leq \Pr[E]. \tag{45}$$

To upper bound $\Pr[E]$ let $\mathcal{B}_1$ be the following AKE adversary against protocol Π. First $\mathcal{B}_1$ obtains a list of public keys from its AKE experiment which it forwards to $\mathcal{A}$. It then simulates the $\mathrm{EA}_{\Pi^+}(\mu, \ell)$ game for $\mathcal{A}$ as follows.

- All of $\mathcal{A}$'s RevLTK and $\mathsf{RegisterLTK}$ queries $\mathcal{B}_1$ simply forwards to its own AKE game.
- When $\mathcal{A}$ makes a Send query to an oracle which still hasn't accepted in protocol Π, then $\mathcal{B}_1$ forwards it to its own AKE experiment and returns the response back to $\mathcal{A}$.
- When an oracle π_i^s reaches the accept state in protocol Π, then $\mathcal{B}_1$ obtains its intermediate key k_Π by issuing a $\mathsf{RevSessKey}$ query to its own AKE game.
- From this point on $\mathcal{B}_1$ simulates the rest of protocol Π^+ for oracle π_i^s itself based on the intermediate key k_Π. (This includes deriving the keys k_a and k_m from k_Π, as well as creating/verifying the final key-confirmation messages.)
- Since $\mathcal{B}_1$ derives the session key k_a of all oracles in protocol Π^+, it can answer all of $\mathcal{A}$'s $\mathsf{RevSessKey}$ queries itself.

Once $\mathcal{A}$ stops, $\mathcal{B}_1$ stops as well and outputs some bit to its AKE experiment (we don't care about $\mathcal{B}_1$'s key-indistinguishability advantage).

From its description it is clear that $\mathcal{B}_1$ perfectly simulates Game 0, hence

$$\Pr[E] \leq \mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{B}_1) \tag{46}$$

and (44) follows. □

In the following, say that an oracle π_i^s is Π-*fresh* if it is fresh according to Definition 7 with respect to protocol Π.

Game 2. In this game the intermediate keys of all Π-fresh oracles in protocol Π^+ are replaced with random keys (partners get the same random key). There exists an adversary $\mathcal{B}_2$ such that:

$$|\Pr[S_2] - \Pr[S_1]| \leq 2 \cdot \mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{B}_2). \tag{47}$$

Proof of (47). Adversary $\mathcal{B}_2$ is almost identical to adversary $\mathcal{B}_1$ described in the previous game, except for the following difference.

- When an oracle π_i^s reaches the accept state in protocol Π, then $\mathcal{B}_2$ obtains its intermediate key k_Π by issuing the following query to its own AKE experiment:
  - if π_i^s is Π-fresh, issue a Test query;
  - if π_i^s is not Π-fresh, issue a $\mathsf{RevSessKey}$ query.

The rest of $\mathcal{B}_2$'s simulation is identical to that of $\mathcal{B}_1$. Additionally, if at any point during the simulation event $\mathsf{break}_{\mathrm{EA}}$ occurs, then $\mathcal{B}_2$ stops and outputs "0" to its AKE experiment. If $\mathsf{break}_{\mathrm{EA}}$ has not occurred by the time $\mathcal{A}$ stops, then $\mathcal{B}_2$ outputs "1".

Let b be the secret bit used in $\mathcal{B}_2$'s AKE experiment. If the Test queries return real keys ($b = 0$), then $\mathcal{B}_2$ perfectly simulates Game 1. If the Test queries return random keys ($b = 1$), then $\mathcal{B}_2$ perfectly simulates Game 2. In other words:

$$\Pr[\mathcal{B}_2^{\mathcal{A}} \Rightarrow 0 \mid b = 0] = \Pr[S_1] \tag{48}$$

and

$$\Pr[\mathcal{B}_2^{\mathcal{A}} \Rightarrow 0 \mid b = 1] = \Pr[S_2], \tag{49}$$

hence

$$\begin{aligned} 2 \cdot \mathrm{Adv}^{\mathsf{AKE}}_{\Pi}(\mathcal{B}_2) &= |\Pr[\mathcal{B}_2^{\mathcal{A}} \Rightarrow 0 \mid b = 0] - \Pr[\mathcal{B}_2^{\mathcal{A}} \Rightarrow 0 \mid b = 1]| && (50)\\ &= |\Pr[S_1] - \Pr[S_2]|, && (51) \end{aligned}$$

and (47) follows. □

Game 3. This game proceeds exactly as the previous one, except that the PRF is replaced with a truly random function for all oracles whose intermediate keys got replaced with a random key in Game 2. That is, computations of the form $\mathsf{PRF}(k_\Pi, \cdot)$ are replaced with $F_{k_\Pi}(\cdot)$ where F_{k_Π} is a truly random function. It is straightforward to create an adversary $\mathcal{D}$ against the PRF such that

$$|\Pr[S_3] - \Pr[S_2]| \leq \mathrm{Adv}^{\mathsf{mu\text{-}PRF}}_{\mathsf{PRF},\mu\ell}(\mathcal{D}). \tag{52}$$

Analyzing Game 3. We now show that if event $\mathsf{break}_{\mathrm{EA}}$ occurs in Game 3, then we can create an adversary $\mathcal{F}$ such that

$$\Pr[S_3] \leq \mathrm{Adv}^{\mathsf{mu\text{-}SUF\text{-}CMA}}_{\mathsf{MAC},\mu\ell}(\mathcal{F}). \tag{53}$$

Let $\mathcal{A}$ be an EA adversary against protocol Π^+ and let $\mathcal{F}$ be the following mu-SUF-CMA attacker against the MAC scheme MAC. Adversary $\mathcal{F}$ has access to two oracles $\textsc{Tag}(\cdot,\cdot)$ and $\textsc{Vrfy}(\cdot,\cdot,\cdot)$, where $\textsc{Tag}(i,m)$ returns a tag t on message m under the ith MAC key in the mu-SUF-CMA game, and $\textsc{Vrfy}(i,m,t)$ verifies m and t under the ith MAC key. $\mathcal{F}$ will use $\mathcal{A}$ to forge in its mu-SUF-CMA game by simulating Game 3 for $\mathcal{A}$ as follows.

- It creates all the long-term keys itself (hence can answer all RevLTK and $\mathsf{RegisterLTK}$ queries).
- It simulates all of protocol Π itself for all oracles. This includes deriving their keys k_Π, k_a, and k_m as in Game 3 (the random function can be implemented by lazy sampling). As a result, $\mathcal{F}$ can answer all of $\mathcal{A}$'s $\mathsf{RevSessKey}$ queries.
- To create and verify the key-confirmation messages in protocol Π^+, $\mathcal{F}$ proceeds as follows.
  - If an oracle is non-fresh, meaning that its intermediate key k_Π was not replaced with random in Game 2, then $\mathcal{F}$ uses its MAC key k_m to create and verify its key-confirmation messages itself.
  - If an oracle *is* fresh, then $\mathcal{F}$ uses its two oracles $\textsc{Tag}$ and $\textsc{Vrfy}$ to create and verify the oracle's key-confirmation messages. Note that the same MAC key-identifier x is used in the mu-SUF-CMA game when creating/verifying the key-confirmation messages of two partner oracles π_i^s and π_j^t. That is, $\textsc{Tag}(x,\cdot)$ and $\textsc{Vrfy}(x,\cdot,\cdot)$ is called for both π_i^s and π_j^t for some key index x.

First notice that $\mathcal{F}$ perfectly simulates Game 3. This is because by Game 2 and Game 3 the MAC keys k_m of fresh oracles are independent and uniformly distributed, which is exactly what the MAC keys in $\mathcal{F}$'s mu-SUF-CMA game are as well.

We now argue that if event $\mathsf{break}_{\mathrm{EA}}$ occurs during $\mathcal{F}$'s simulation, then $\mathcal{F}$ creates a forgery for some key in its mu-SUF-CMA game. Suppose π_i^s was the oracle for which event $\mathsf{break}_{\mathrm{EA}}$ occurred. By definition, this means that π_i^s accepted, its intended peer's long-term key was not corrupted, and no oracle at the peer has matching conversations to π_i^s.

In order for π_i^s to accept in protocol Π^+ the key-confirmation message it received, call it t, must have been valid. Also, since Pid_i^s was not corrupted, π_i^s was Π^+-fresh, and hence Π-fresh, at the time its intermediate key was derived. This means that π_i^s's key-confirmation messages were created and verified using $\textsc{Tag}$ and $\textsc{Vrfy}$ in $\mathcal{F}$'s simulation. In particular, t was verified with the call

$\textsc{Vrfy}(x, recv, t)$, where x is the key index associated with π_i^s in the mu-SUF-CMA game and $recv$ is the string used to verify π_i^s's incoming key-confirmation message. Specifically, if π_i^s sends the last message then $recv = 0||\texttt{ctxt}$, while if it receives the last message then $recv = 1||\texttt{ctxt}$ (see Fig. 4).

We claim that $(recv, t)$ is a valid forgery in $\mathcal{F}$'s mu-SUF-CMA game for key index x. Recall that the oracles $\textsc{Tag}(x, \cdot)$ and $\textsc{Vrfy}(x, \cdot, \cdot)$ are only called in $\mathcal{F}$'s simulation when π_i^s—and possibly its Π-partner[5]—accepts in protocol Π. Correspondingly, we consider two cases:

1. π_i^s *does not have a partner in protocol* Π. In this case, $\mathcal{F}$ makes no call of the form $\textsc{Tag}(x, recv)$ in its mu-SUF-CMA game so $(recv, t)$ is a valid forgery for key index x. (Here we are using that the *sent* and *recv* verification strings are distinct.)

2. π_i^s has *a partner in protocol* Π. Suppose this partner is π_j^t. In response to π_j^t accepting in protocol Π, $\mathcal{F}$ called $\textsc{Tag}(x, recv)$ in order to create its key-confirmation message t'. If $t' = t$, then π_j^t would have matching conversations to π_i^s, contradicting event $\mathsf{break}_{\mathrm{EA}}$. So we must have $t' \neq t$. But then $(recv, t)$ must be a valid forgery since t was not produced by a call of the form $\textsc{Tag}(x, recv)$. (Here we are again using that *sent* and *recv* are distinct.)

We have thus shown that event $\mathsf{break}_{\mathrm{EA}}$ in Game 3 implies a MAC forgery; proving (53). Together with (44), (47), and (52), this proves Theorem 6.

D Version History

v1.0 (2019-06-20) Original ePrint version.

v1.1 (2019-08-16) This version.

- Updated security model. Removed the sentence "and all oracles $\pi_i^1, \ldots, \pi_i^\ell$ now respond with $\perp$ to all queries." in the definition of the RevLTK query. That is, the adversary can now continue to interact with oracles also after a RevLTK query. The restriction effectively made it impossible for the adversary to carry out KCI attacks.

  Note that the proofs never relied on this semantic; the update only ensures to make the model actually reflect the attacks captured by our protocols and proofs.

- Updated the EA definition (Definition 12) to be based on matching conversations (Definition 11) instead of partnering (Definition 6). The old version artificially allowed EA to be trivially broken (in the model).

  Thanks to Paul Rösler for identifying this issue.

[5] Note that there is no contradiction between π_i^s having a partner π_j^t in protocol Π, while at the same time π_j^t not having matching conversations to π_i^s in protocol Π^+. This is because the transcript of protocol Π^+ also includes the key-confirmation messages.

- Updated the abstract to give a more detailed description of the paper's contents.