Practical and Tightly-Secure Digital Signatures and Authenticated Key Exchange

Kristian Gjøsteen[1] and Tibor Jager[2]

[1] NTNU - Norwegian University of Science and Technology, Trondheim, Norway, kristian.gjosteen@ntnu.no
[2] Paderborn University, Paderborn, Germany, tibor.jager@upb.de

Abstract. Tight security is increasingly gaining importance in real-world cryptography, as it allows to choose cryptographic parameters in a way that is supported by a security proof, without the need to sacrifice efficiency by compensating the security loss of a reduction with larger parameters. However, for many important cryptographic primitives, including digital signatures and authenticated key exchange (AKE), we are still lacking constructions that are suitable for real-world deployment.
We construct the first truly practical signature scheme with tight security in a real-world multi-user setting with adaptive corruptions. The scheme is based on a new way of applying the Fiat-Shamir approach to construct tightly-secure signatures from certain identification schemes.
Then we use this scheme as a building block to construct the first practical AKE protocol with tight security. It allows the establishment of a key within 1 RTT in a practical client-server setting, provides forward security, is simple and easy to implement, and thus very suitable for practical deployment. It is essentially the "signed Diffie-Hellman" protocol, but with an additional message, which is crucial to achieve tight security. This additional message is used to overcome a technical difficulty in constructing tightly-secure AKE protocols.
For a theoretically-sound choice of parameters and a moderate number of users and sessions, our protocol has comparable computational efficiency to the simple signed Diffie-Hellman protocol with EC-DSA, while for large-scale settings our protocol has even better computational performance, at moderately increased communication complexity.

1 Introduction

Tight security. In modern cryptography it is standard to propose new cryptographic constructions along with a *proof of security*. The provable security paradigm, which goes back to a seminal work of Goldwasser and Micali [27], becomes increasingly relevant for "real-world" cryptosystems today. For instance, the upcoming TLS version 1.3[3] is the first version of this important protocol where formal security proofs were used as a basis for several fundamental design decisions [44].

[3] See https://tools.ietf.org/html/draft-ietf-tls-tls13-23 for the latest draft.

A security proof usually describes a reduction (in a complexity-theoretic sense), which turns an efficient adversary $\mathcal{A}$ breaking the considered cryptosystem into an efficient adversary $\mathcal{B}$ breaking some underlying complexity assumption, such as the discrete logarithm problem, for example. If $\mathcal{B}$ can be shown to have about the same running time and success probability as $\mathcal{A}$ (up to a constant factor), then the reduction is said to be *tight*. However, many security proofs are not tight. For example, we are often only able to show that if $\mathcal{A}$ runs in time $t_{\mathcal{A}}$ and has success probability $\epsilon_{\mathcal{A}}$, then $\mathcal{B}$ runs in time $t_{\mathcal{B}} \approx t_{\mathcal{A}}$, but we can bound its success probability only as $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}}/Q$, where Q is the *security loss*. Q can often be "large", e.g., linear or even quadratic in the number of users.

If Q is polynomially bounded, then this still yields a polynomial-time reduction in the sense of classical asymptotic complexity theory. However, if we want to deploy the cryptosystem in practice, then the size of cryptographic parameters (like for instance the size of an underlying algebraic group, where the discrete logarithm problem is assumed to be hard) must be determined. If we want to choose these parameters in a theoretically-sound way, then a larger loss Q must be compensated by larger parameters, which in turn has a direct impact on efficiency. For example, in the discrete logarithm setting, this would typically require an increase in the group order by a factor Q^2. As a concrete example, 2^{32} users with 2^{32} sessions each and quadratic security loss would force us to use 521 bit elliptic curves instead of 256 bit elliptic curves, which more than quintuples the cost of an exponentiation on one typical platform (as measured by `openssl speed`). Thus, in order to be able to instantiate the cryptosystem with "optimal" parameters, we need a tight security proof.

The possibility and impossibility of tight security proofs has been considered for many primitives, including symmetric encryption [29, 31, 36], public-key encryption [5, 32, 24, 3], (hierarchical) identity-based encryption [16, 11], digital signatures [37, 45, 32, 33, 43, 23, 47, 22], authenticated key exchange [2], and more. It also becomes increasingly relevant in "real-world" cryptography. For instance, most recently, Gueron and Lindell [29] improved the tightness of the AES-GCM-SIV nonce misuse-resistant encryption scheme, with a new nonce-based key derivation method. This construction is now part of the current draft of the corresponding RFC proposal,[4] won the best paper award at ACM CCS 2017, and is already used in practice in Amazon's AWS key management scheme.[5]

In many important areas with high real-world relevance, including digital signature schemes and authenticated key exchange protocols, we still do not have any tightly-secure constructions that are suitable for practical deployment. Known schemes either have a security loss which is at least linear in the number of users (typical for digital signatures) or even *quadratic* in the number of protocol sessions (typical for authenticated key exchange), if a real-world multi-user security model is considered. This huge security loss often makes it impossible to choose deployment parameters in a theoretically-sound way, because such parameters would have to be unreasonably large and thus impractical.

[4] See `https://tools.ietf.org/html/draft-irtf-cfrg-gcmsiv-07`.
[5] See `https://rwc.iacr.org/2018/Slides/Gueron.pdf`.

1.1 Tightly-Secure Digital Signatures

In the domain of digital signatures, there are two relevant dimensions for tightness: (i) the number of signatures issued per public key, and (ii) the number of users (=public keys).

For some important "real-world" schemes, such as Schnorr signatures, impossibility results suggest that current proof techniques are not sufficient to achieve tightness [43, 23, 47, 22], not even if only the first dimension is considered in a *single-user* setting. Some other schemes have a tight security proof in the first dimension [37, 45, 32, 10]. However, in a more realistic multi-user setting with adaptive corruptions, which appears to be the "right" real-world security notion for applications of signatures such as key exchange, cryptocurrencies, secure instant messaging, or e-mail signatures, there are only very few constructions with tight security in both dimensions.

One construction is due to Bader [1]. It is in the random oracle model [6], but this seems reasonable, given the objective of constructing simple and efficient real-world cryptosystems. However, the construction requires bilinear maps (aka. pairings). Even though bilinear maps have become significantly more efficient in the past years, their practical usability is still not comparable to schemes over classical algebraic groups, such as elliptic curves without pairings. Furthermore, bilinear maps involve rather complex mathematics, and are therefore rather difficult to implement, and not yet available on many platforms and software libraries, in particular not for resource-constrained lightweight devices or smartcards. Finally, recent advances in solving the discrete logarithm problem [4] restrain the applicability of bilinear maps in settings with high performance and security requirements.

The other two known constructions are due to Bader *et al.* [2]. Both have a security proof in the standard model (i.e., without random oracles), but are also based on bilinear maps. The first one uses a simulation-sound Groth-Sahai [28] proof system, which internally uses a tree-based signature scheme to achieve tightness. Thus, a signature of the resulting construction consists of hundreds of group elements, and is therefore not suitable for practical deployment. The second scheme is more efficient, but here public keys consist of hundreds of group elements, which is much larger than the public key size of any other schemes currently used in practice, and seems too large for many applications.

In summary, the construction of a practical signature scheme without bilinear maps, which provides *tight* security in a realistic multi-user setting with corruptions and in the standard sense of *existential unforgeability under chosen-message attacks*, is an important open problem. A solution would provide a very useful building block for applications where the multi-user setting with adaptive corruptions appears to be the "right" real-world security notion, such as those mentioned above.

The difficulty of constructing tightly-secure signatures. Constructing a tightly-secure signature scheme in a real-world multi-user security model with adaptive corruptions faces the following technical challenge. In the μ-user setting, the ad-

versary $\mathcal{A}$ receives as input a list $pk_1, \ldots, pk_\mu$ of public keys. We denote the corresponding secret keys with $sk_1, \ldots, sk_\mu$. $\mathcal{A}$ is allowed to ask two types of queries. It may either output a tuple (m, i), to request a signature for a chosen message m under secret key sk_i. The security experiment responds with $\sigma \xleftarrow{\$} \mathsf{Sign}(sk_i, m)$. Or it may "corrupt" keys. To this end, it outputs an index i, and the experiment responds with sk_i. Adversary $\mathcal{A}$ breaks the security, if it outputs (i^*, m^*, σ^*) such that σ^* verifies correctly for a new message m^* and with respect to an uncorrupted key pk_{i^*}. Note that this is the natural extension of *existential unforgeability under chosen-message attacks* (EUF-CMA) to the multi-user setting with corruptions. Following [2], we will call it $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$*-security*. Security in this sense is implied by the standard EUF-CMA security definition, by a straightforward reduction that simply guesses the index i^* of the uncorrupted key, which incurs a security loss of $Q = 1/\mu$ that is linear in the number of users.

Now let us consider the difficulty of constructing a security reduction $\mathcal{B}$ which does not lose a factor $Q = 1/\mu$. On the one hand, in order to avoid the need to guess an uncorrupted key, $\mathcal{B}$ must "know" *all* secret keys $sk_1, \ldots, sk_\mu$, in order to be able to answer key corruption queries.

On the other hand, however, the reduction $\mathcal{B}$ must be able to extract the solution to a computationally hard problem from the forgery (i^*, m^*, σ^*). If $\mathcal{B}$ "knows" sk_{i^*}, then it seems that it could compute (m^*, σ^*) even without the help of the adversary. Now, if $\mathcal{B}$ is then able to extract the solution of a "hard" computational problem from (m^*, σ^*), then this means that the underlying hardness assumption must be false, and thus the reduction $\mathcal{B}$ is not meaningful.

The above argument seems to suggests that achieving tight $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security is impossible. One can even turn this intuition into a formal impossibility result, as done in [3]. However, it turns out that the impossibility holds only for schemes where the distribution of signatures that are computable with a secret key sk_{i^*} known to reduction $\mathcal{B}$ is identical to the distribution of signatures (m^*, σ^*) output by adversary $\mathcal{A}$. This provides a leverage to overcome the seeming impossibility. Indeed, the known constructions of tightly $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-secure schemes [1, 2] circumvent the impossibility result. As we describe below in more detail, these constructions essentially ensure that with sufficiently high probability the adversary $\mathcal{A}$ will output a message-signature pair (m^*, σ^*) such that σ^* is *not* efficiently computable, even given sk_{i^*}.

The main technical challenge in constructing signature schemes with tight security in a real-world multi-user security model with corruptions is therefore to build the scheme in a way that makes it possible to argue that the reduction $\mathcal{B}$ is able to extract the solution to a computationally hard problem from the forged signature computed by $\mathcal{A}$, *even though $\mathcal{B}$ knows secret keys for all users.*

On constructing tightly-secure signatures without bilinear maps. All previously known tightly $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-secure signature schemes [1, 2] essentially work as follows. A public key pk consists of two public keys $pk = (pk_0, pk_1)$ of a "base" signature scheme, which is tightly-secure in a multi-user setting *without* corruptions. The secret key sk consists of a random secret key $sk = sk_b$, $b \xleftarrow{\$} \{0, 1\}$,

for either pk_0 or pk_1. A signature consists of a Groth-Sahai-based [28] witness-indistinguishable OR-proof of knowledge, proving knowledge of a signature that verifies either under pk_0 OR under pk_1. In the security proof, the reduction $\mathcal{B}$ basically knows sk_b (and thus is able to respond to all corruption-queries of $\mathcal{A}$), but it hopes that $\mathcal{A}$ produces a proof of knowledge of a signature under pk_{1-b}, which can then be extracted from the proof of knowledge and be used to break the instance corresponding to pk_{1-b}.

A natural approach to adopt this technique to signatures without pairings would be to use a Fiat-Shamir-like proof of knowledge [21], in combination with the very efficient OR-proofs of Cramer-Damgård-Schoenmakers (CDS) [18]. However, here we face the following difficulties. First, existing signature schemes that use a Fiat-Shamir-like proof *of knowledge*, such as for the Schnorr scheme [46], are already difficult to prove tightly secure in the single-user setting, due to known impossibility results [43, 23, 47, 22]. Second, its tightly-secure variants, such as the DDH-based scheme of Katz-Wang [37] and the CDH-based schemes of Goh-Jarecki [25] or Chevallier-Mames [17], do not use a proof *of knowledge*, but actually a proof of language membership, where we cannot extract a witness along the lines of [1, 2]. Thus, adopting the approach of [1, 2] to efficient signature schemes without pairings requires additional ideas and new techniques.

Our contributions. We construct the first tightly $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-secure signature scheme which does not require bilinear maps. We achieve this by describing a new way of combining the efficient EDL signature scheme considered in [25] with Cramer-Damgård-Schoenmakers proofs [18], in order to obtain tightly-secure signatures in the multi-user setting with adaptive corruptions.

The scheme is very efficient, in particular in comparison to previous schemes with tight multi-user security. A public key consists of only two group elements, while the secret key consists of a bit and one integer smaller than the group order. A signature consists of a random nonce, two group elements and four integers smaller than the group order. The computational cost of the algorithms is dominated by exponentiations. Key generation costs a single exponentiation. Signing costs a single exponentiation plus the generation of a proof, for a total of seven exponentiations. Verification costs eight exponentiations.

1.2 Tightly-Secure Authenticated Key Exchange

Modern security models for authenticated key exchange consider very strong adversaries, which control the entire communication network, may adaptively corrupt parties to learn their long-term secret keys, or reveal session keys of certain sessions. This includes all security models that follow the classical Bellare-Rogaway [7] or Canetti-Krawczyk [14] approach, for instance. The adversary essentially breaks the security, if it is able to distinguish a non-revealed session key from random. Furthermore, in order to achieve desirable properties like *forward security* (aka. *perfect forward secrecy*) [30], the attacker is even allowed to attack session keys belonging to sessions where one or both parties are corrupted,

as long as these corruptions do not allow the adversary to trivially win the security experiment (e.g., because it is able to corrupt a communication partner *before* the key is computed, such that the attacker can impersonate this party).

The huge complexity of modern security models for authenticated key exchange makes it difficult to construct tightly-secure protocols. Most examples of modern key exchange protocols even have a *quadratic* security loss in the total number of protocol sessions, which stems from the fact that a reduction has to guess two oracles in the security experiment that belong to the protocol session "tested" by the adversary (cf. the discussion of the "commitment problem" below).

Despite the huge practical importance of authenticated key exchange protocols, we currently know only a single example of a protocol that achieves tight security [2], but it requires complex building blocks, such as one of the tightly-secure signature schemes sketched above, as well as a special key encapsulation mechanisms that is composed of two public-key encryption schemes. Given the huge demand for efficient key exchange protocol in practice, the construction of a simple and efficient, yet tightly-secure, authenticated key exchange protocol *without* these drawbacks is an important open problem.

Our contributions. We describe the first truly practical key exchange protocol with tight security in a standard security model for authenticated key exchange. The construction (but not the security proof) is very simple, which makes the protocol easy to implement and ready to use, even on simple devices.

Our protocol is able to establish a key with very low latency, in three messages and within a single *round-trip time* (1-RTT) in a standard client-server setting. This holds even in a typical real-world situation where both communication partners are initially not in possession of their communication partner's public keys, and therefore have to exchange their certified public keys within the protocol. Furthermore, the protocol provides full *forward security*.

In Appendix 5 we analyse the computational efficiency of our protocol, instantiated with our signature scheme, by comparing it to the simple "signed Diffie-Hellman" protocol, instantiated with EC-DSA. The analysis is based on the benchmark for ECC arithmetic of OpenSSL, and considers a theoretically-sound choice of cryptographic parameters based on the tightness of security proofs. Even though our protocol requires a larger absolute number of exponentiations, already in small-to-medium-scale settings this is quickly compensated by the fact that arithmetic in large groups is significantly more costly than in small groups. In a large-scale setting, our protocol even outperforms signed Diffie-Hellman with EC-DSA with respect to computational performance. This comes at the cost of moderately increased communication complexity, when compared to the (extremely communication-efficient) EC-DSA-signed Diffie-Hellman protocol.

Sketch of our construction and technical difficulties. Our starting point is the standard "signed Diffie-Hellman" protocol, instantiated with our tightly-secure multi-user signature scheme. However, we stress that this is not yet sufficient to

achieve tight security, due to the *"commitment problem"* described below. We resolve this problem with an additional message, which is important to achieve tight security, but does not add any additional latency to the protocol.

More precisely, consider the standard "signed Diffie-Hellman" protocol, executed between Alice and Bob, where Bob first sends $v = (g^b, \sigma_B)$, where σ_B is a signature under Bob's secret key over g^b, Alice responds with $w = (g^a, \sigma_A)$, where σ_A is Alice's signature over g^a, and the resulting key is $k = g^{ab}$. Let us sketch why this protocol seems not to allow for a tight security proof.

In a Bellare-Rogaway-style security model, such as the one that we describe in Section 4.1, each session of each party is modelled by an oracle π_i^s, where $(i, s) \in [\mu] \times [\ell]$, μ is the number of parties and ℓ is the number of sessions per party. Now, consider a reduction $\mathcal{B}$ which receives as input a DDH challenge (g, g^x, g^y, g^z), and now wants to embed these values into the view of the key exchange adversary $\mathcal{A}$. One way to do this, which is used in most existing security proofs of "signed Diffie-Hellman-like" protocols (such as [34, 39, 35], for instance) is to guess two oracles π_i^s and π_j^t, embed g^x into the message sent by π_i^s, g^y into the message sent by π_j^t, and then hope that the adversary will forward g^y to π_i^s and "test" the key of oracle π_i^s, where the g^z-value from the DDH challenge is then embedded. Note that the need to guess two out of $\mu\ell$ oracles correctly incurs a quadratic security loss of $O(\mu^2\ell^2)$, which is extremely non-tight.

A natural approach to improve tightness is to use the well-known random self-reducibility of DDH [5], and embed randomised versions of g^x and g^y into the messages of more than one oracle. However, here we face the following difficulty. As soon as an oracle π_i^s has output a Diffie-Hellman share g^a, the reduction $\mathcal{B}$ has essentially committed itself to whether it "knows" the discrete logarithm a.

- If oracle π_i^s outputs a randomised version of g^x, $g^a = g^{x+e_i^s}$ where e_i^s is the randomization, then $\mathcal{B}$ does not "know" the discrete logarithm $a = x + e_i^s$. Now it may happen that the adversary $\mathcal{A}$, which controls the network and possibly also some parties, sends a value h to oracle π_i^s (on behalf of some third party), such that h is *not* controlled by the reduction $\mathcal{B}$. If then $\mathcal{A}$ asks the reduction to reveal the key of oracle π_i^s, then the reduction fails, because it is not able to efficiently compute $k = h^a$.
- This problem does not occur, if $g^a = g^{e_i^s}$ such that $\mathcal{B}$ "knows" the discrete logarithm a. However, if it now happens that the adversary $\mathcal{A}$ decides to distinguish the key k of oracle π_i^s from random, then again the reduction fails, because it is not able to embed g^z into k.

This *"commitment problem"* is the reason why many classical security proofs, in particular for signed Diffie-Hellman protocols, have a quadratic security loss. They embed a DDH challenge into the view of adversary $\mathcal{A}$ by guessing two out of $\mu\ell$ oracles, and the reduction will fail if the guess is incorrect.

We resolve the commitment problem in a novel way by adding an additional message. We change the protocol such that Alice initiates the protocol with a message $u = G(g^a)$, where G is a cryptographic hash function (cf. Figure 3). This message serves as a commitment by Alice to g^a. Then the protocol proceeds as before: Bob sends $v = (g^b, \sigma_B)$, Alice responds with $w = (g^a, \sigma_A)$, and the

resulting key is $k = g^{ab}$.[6] However, Bob will additionally check whether the first message u received from Alice matches the third protocol message, that is, $u = G(g^a)$, and abort if not.

As we will prove formally in Section 4.3, the additional message u resolves the commitment problem as follows. We will model G as a random oracle. This guarantees that from the point of view of the adversary $\mathcal{A}$, a value $G(h)$ forms a binding and hiding commitment to h. However, for the reduction $\mathcal{B}$, u is not binding, because $\mathcal{B}$ controls the random oracle. We will construct $\mathcal{B}$ such that whenever an oracle π_i^s outputs a first protocol message u, then receives back a message $v = (g^b, \sigma_B)$, and now has to send message $w = (g^a, \sigma_A)$, then $\mathcal{B}$ it is able to *retroactively* decide to embed the element g^x from the DDH challenge into u such that $u = G(g^{x+e_i^s})$, or not and it holds that $u = G(g^{e_i^s})$. This is possible by re-programming the random oracle in a suitable way.[7]

We will explain in Section 4.2 that the additional message u does not increase latency to the protocol, when used in a standard client-server setting. This is essentially because Alice can send cryptographically protected payload immediately after receiving message $v = (g^b, \sigma_B)$ from Bob, *along with* message $w = (g^a, \sigma_A)$. Thus, in a typical client-server setting, where the client initiates the protocol and then sends data to the server, the overhead required to establish a key is only 1 RTT, exactly like for ordinary signed Diffie-Hellman.

Outline. Section 2 recalls the necessary background and standard definitions. The signature scheme is described and proven secure in Section 3, the AKE protocol is considered in Section 4.

2 Background

In this section, we recap some background and standard definitions of Diffie-Hellman problems, the Fiat-Shamir heuristic, and digital signatures.

Diffie-Hellman Problems. Let $\mathbb{G}$ denote a cyclic group of prime order p and let g be a generator. Let $\mathcal{DDH}$ be the set of *DDH tuples* $\{(g^a, g^b, g^{ab}) \mid a, b \in \{0, 1, \ldots, p-1\}$.

Definition 1. *Let $\mathcal{A}$ be an algorithm that takes two group elements as input and outputs a group element. The* success probability *of $\mathcal{A}$ against the* Computational Diffie-Hellman (CDH) problem *is*

$$\mathrm{Succ}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{A}) = \Pr[\mathcal{A}(x, y) = z \mid (x, y, z) \leftarrow \mathcal{DDH}].$$

[6] Our actual protocol will compute the key as $k = H(g^{ab})$ for a hash function H, but this is not relevant here.

[7] We note that a programmable random oracle is not inherently necessary here. Instead, we could use an equivocal commitment scheme[19] in place of random oracle G. However, this would make the protocol more complex. Since we want to maximise efficiency and simplicity of the protocol, we consider the random oracle as an adequate choice for our purpose.

We say that $\mathcal{A}$ (t, ϵ)-breaks CDH if $\mathcal{A}$ runs in time t and its success probability $\mathrm{Succ}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{A})$ *is at least ϵ.*

Definition 2. *Let $\mathcal{A}$ be an algorithm that takes three group elements as input and outputs 0 or 1. The* advantage *of $\mathcal{A}$ against the* Decision Diffie-Hellman (DDH) problem *[12] is*

$$\mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{A}) = |\Pr[\mathcal{A}(x,y,z) = 0 \mid (x,y,z) \leftarrow \mathcal{DDH}] - \Pr[\mathcal{A}(x,y,z) = 0 \mid (x,y,z) \leftarrow \mathbb{G}^3]|.$$

We say that $\mathcal{A}$ (t, ϵ)-breaks DDH if $\mathcal{A}$ runs in time t and its advantage $\mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{A})$ *is at least ϵ.*

In proofs, it is often convenient to consider an adversary that sees multiple CDH/DDH problems. The n-CDH adversary must solve a CDH problem, but it gets to choose the group elements from two lists of randomly chosen group elements. The n-DDH adversary gets n tuples, all of which are either DDH tuples or random tuples. Again, it is often convenient if some of these DDH tuples share coordinates.

Definition 3. *Let $\mathcal{A}$ be an algorithm that takes as input $2n$ group elements and outputs two integers and a group element. The* success probability *of $\mathcal{A}$ against the n-*CDH problem *is*

$$\mathrm{Succ}^{n\text{-}\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{A}) = \Pr\left[(x_i, y_j, z) \in \mathcal{DDH} \,\middle|\, \begin{array}{c} x_1, \ldots, x_n, y_1, \ldots, y_n \leftarrow \mathbb{G}; \\ (i,j,z) \leftarrow \mathcal{A}(x, \ldots, x_n, y_1, \ldots, y_n) \end{array}\right].$$

Definition 4. *Let $\mathcal{A}$ be an algorithm that outputs 0 or 1. $\mathcal{A}$ has access to an oracle that on input of an integer i returns three group elements. If $i > 0$, then the first group element returned will be the same as the first group element in the oracle's ith response. Let $\mathcal{O}_0$ be such an oracle that returns randomly chosen DDH tuples. Let $\mathcal{O}_1$ be such an oracle that returns randomly chosen triples of group elements.*

The advantage *of the algorithm $\mathcal{A}$ against the n-*DDH problem *is*

$$\mathrm{Adv}^{n\text{-}\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{A}') = |\Pr[\mathcal{A}^{\mathcal{O}_0} = 0] - \Pr[\mathcal{A}^{\mathcal{O}_1} = 0]|.$$

It is clear that 1-CDH and 1-DDH correspond to the ordinary problems. Likewise, it is clear that we can embed a CDH or DDH problem in a n-CDH or n-DDH problem, so a hybrid argument would relate their advantage. However, the DH problems are random self-reducible, which means that we can create better bounds.

Theorem 1. *Let $\mathcal{A}$ be an adversary against n-CDH. Then there exists an adversary $\mathcal{B}$ against CDH such that*

$$\mathrm{Succ}^{n\text{-}\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{A}) = \mathrm{Succ}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{B}).$$

The difference in running time is linear in n.

Theorem 2. *Let $\mathcal{A}$ be an adversary against n-DDH. Then there exists an adversary $\mathcal{B}$ against DDH such that*

$$\mathrm{Adv}^{n\text{-}\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{A}') \leq \mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{B}) + \frac{1}{p}.$$

The difference in running time is linear in n.

The proof of the first theorem is straight-forward. A proof of the second theorem can be found in e.g. Bellare, Boldyreva and Micali [5].

Proofs of equality of discrete logarithms. Sigma protocols are special three-move protocols originating in the Schnorr identification protocol [46]. We shall need a proof of equality for discrete logarithms [15] together with the techniques for creating a witness-indistinguishable OR-proof [18].

Let $y, x, z \in \mathbb{G}$ be such that $x = g^a$ and $z = y^a$. The standard sigma protocol [15] for proving that $\log_g x = \log_y z$ works as follows:

Commitment Sample $\rho \leftarrow \{0, 1, \ldots, p-1\}$. Compute $\alpha_0 = g^\rho$ and $\alpha_1 = y^\rho$. The commitment is (α_0, α_1).
Challenge Sample $\beta \leftarrow \{0, 1, \ldots, p-1\}$. The challenge is β.
Response Compute $\gamma \leftarrow \rho - \beta a \bmod p$. The response is γ.
Verification The verifier accepts the response if

$$\alpha_0 = g^\gamma x^\beta \qquad\qquad \alpha_1 = y^\gamma z^\beta.$$

The usual special honest verifier zero knowledge simulator producing a simulated conversation on public input (x, y, z) and challenge β is denoted by $\mathsf{ZSim}_{\mathrm{eq}}(x, y, z; \beta)$, and it is a perfect simulator. The cost of generating a proof is dominated by the two exponentiations, while the simulation cost is dominated by four exponentiations.

We turn the proofs non-interactive using the standard Fiat-Shamir [21] heuristic, in which case the proof is a pair of integers (β, γ). We denote the algorithm for generating a non-interactive proof π_{eq} that $\log_g x = \log_y z$ by $\mathsf{ZPrv}_{\mathrm{eq}}(a; x, y, z)$. The algorithm for verifying that π_{eq} is a valid proof of this claim is denoted by $\mathsf{ZVfy}_{\mathrm{eq}}(\pi_{\mathrm{eq}}; x, y, z)$, which outputs 1 if and only if the proof is valid.

Based on this proof of equality for a pair of discrete logarithms, an OR-proof for the equality of one out of two pairs of discrete logarithms can be constructed using standard techniques [18].

Briefly, the prover chooses a random challenge β_{1-b} and uses the perfect simulator $\mathsf{ZSim}_{\mathrm{eq}}(\ldots)$ to generate a simulated proof for the inequal pair. It then runs the equal d.log. prover which produces a commitment. When the verifier responds with a challenge β, the prover completes the proof for the equal pair using the challenge $\beta_b = \beta - \beta_{1-b}$. It then responds with both challenges and both responses. The verifier checks that the challenges sum to β.

We denote the special honest verifier simulator by

$$\mathsf{ZSim}_{\mathrm{eq,or}}(x_0, x_1, y_0, y_1, z_0, z_1; \beta_0, \beta_1)$$

We note that for any given challenge pair (β_0, β_1), the simulator generates a particular transcript with probability $1/p^2$.

Again, we can turn these proofs non-interactive using Fiat-Shamir and a hash function H_2. In this case, the proof is a tuple $(\beta_0, \beta_1, \gamma_0, \gamma_1)$ of integers, and the verifier additionally checks that the hash value equals the sum of β_0 and β_1. The non-interactive algorithms for generating and verifying proofs are denoted by $\mathsf{ZPrv}_{\mathrm{eq,or}}(b, a_b; x_0, x_1, y_0, y_1, z_0, z_1)$ and $\mathsf{ZVfy}_{\mathrm{eq,or}}(\pi_{\mathrm{eq,or}}; x_0, x_1, y_0, y_1, z_0, z_1)$. The cost of generating a proof is dominated by the two exponentiations for the real equality proof and the four exponentiations for the fake equality proof.

As usual, the simulator is perfect. In addition, these proofs have very strong properties in the random oracle model.

Theorem 3. *Let $\mathcal{A}$ be an algorithm in the random oracle model, making at most l hash queries, that outputs a tuple $(x_0, x_1, y_0, y_1, z_0, z_1)$ of group elements and a proof $\pi_{\mathrm{eq,or}}$. The probability that $\mathsf{ZVfy}_{\mathrm{eq,or}}(\pi_{\mathrm{eq,or}}; x_0, x_1, y_0, y_1, z_0, z_1) = 1$, but $(x_0, y_0, z_0) \notin \mathcal{DDH}$ and $(x_1, y_1, z_1) \notin \mathcal{DDH}$, is at most $\frac{l+1}{p}$.*

The proof of the theorem is straightforward and is implicit in *e.g.* Goh and Jarecki [25].

Digital Signatures. A digital signature scheme consists of a triple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ of algorithms. The *key generation* algorithm Gen (possibly taking a set of parameters Π as input) outputs a key pair (vk, sk). The *signing* algorithm Sign takes a signing key sk and a message m as input and outputs a signature σ. The *verification* algorithm Vfy takes a verification key vk, a message m and a signature σ as input and outputs 0 or 1. For correctness, we require that for all $(vk, sk) \leftarrow \mathsf{Gen}$ we have that $\Pr[\mathsf{Vfy}(vk, m, \mathsf{Sign}(sk, m))] = 1$.

3 Signatures with Tight Multi-User Security

Now we are ready to describe our signature scheme with tight multi-user security in a "real-world" security model with adaptive corruptions.

3.1 Security Definition

We define multi-user existential unforgeability under adaptive chosen-message attacks with adaptive corruptions, called $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$ security in [2]. Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, which is parametrized by the number of public keys μ.

1. For each $i \in [\mu]$, it computes $(pk^{(i)}, sk^{(i)}) \xleftarrow{\$} \mathsf{Gen}(\Pi)$. Furthermore, it initializes a set $\mathcal{S}^{\mathsf{corr}}$ to keep track of corrupted keys, and μ sets $\mathcal{S}_1, \ldots, \mathcal{S}_\mu$, to keep track of chosen-message queries. All sets are initially empty. Then it outputs $(pk^{(1)}, \ldots, pk^{(\mu)})$ to $\mathcal{A}$.

2. $\mathcal{A}$ may now issue two different types of queries. When $\mathcal{A}$ outputs an index $i \in [\mu]$, then $\mathcal{C}$ updates $\mathcal{S}^{\mathsf{corr}} := \mathcal{S}^{\mathsf{corr}} \cup \{i\}$ and returns $sk^{(i)}$. When $\mathcal{A}$ outputs a tuple (m, i), then $\mathcal{C}$ computes $\sigma := \mathsf{Sign}(sk_i, m)$, adds (m, σ) to $\mathcal{S}_i$, and responds with σ.
3. Eventually $\mathcal{A}$ outputs a triple (i^*, m^*, σ^*).

Definition 5. *Let $\mathcal{A}$ be a $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-adversary against a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$. The* advantage *of $\mathcal{A}$ is*

$$\mathrm{Adv}_{\Sigma}^{\mathsf{euf\text{-}cma}}(\mathcal{A}) = \Pr\left[(m^*, i^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{C}} : \begin{array}{c} i^* \notin \mathcal{S}^{corr} \wedge (m^*, \cdot) \notin \mathcal{S}_{i^*} \\ \wedge \mathsf{Vfy}(vk^{(i^*)}, m^*, \sigma^*) = 1 \end{array}\right].$$

We say that $\mathcal{A}$ (t, ϵ, μ)-breaks the $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security of Σ if $\mathcal{A}$ runs in time t and $\mathrm{Adv}_{\Sigma}^{\mathsf{euf\text{-}cma}}(\mathcal{A}) \geq \epsilon$. Here, we include the running time of the security experiment into the running time of $\mathcal{A}$.

Remark 1. We include the running time of the security experiment into the running time t of $\mathcal{A}$, because this makes it slightly simpler to analyse the running time of our reduction precisely. Let t_{Exp} denote the time required to run the security experiment alone, and let $t_{\mathcal{A}}$ be the running time of the adversary alone. Given that the experiment can be implemented very efficiently, we may assume $t_{\mathcal{A}} \geq t_{\mathsf{Exp}}$ for any conceivable adversary $\mathcal{A}$, so this increases the running time at most by a small constant factor. It allows us to make the analysis of our reduction more rigorous.

3.2 Construction

Let $H_1 : R \times \{0,1\}^* \to \mathbb{G}$ be a hash function from a randomness set R and a message space $\{0,1\}^*$ to the group $\mathbb{G}$. The digital signature scheme Σ_{mu} works as follows:

Key generation Sample $b \leftarrow \{0,1\}$, $a_b \leftarrow \{0, 1, \ldots, p-1\}$ and $x_{1-b} \leftarrow \mathbb{G}$. Compute $x_b \leftarrow g^{a_b}$. The signing key is $sk = (b, a_b)$ and the verification key is $vk = (x_0, x_1)$.

Signing To sign a message m using signing key $sk = (b, a_b)$, sample $t \leftarrow R$ and $z_{1-b} \leftarrow \mathbb{G}$, let $y = H_1(t, m)$ and compute $z_b \leftarrow y^{a_b}$. Then create a non-interactive zero knowledge proof

$$\pi_{\mathrm{eq,or}} \leftarrow \mathsf{ZPrv}_{\mathrm{eq,or}}(b, a_b; x_0, x_1, y, y, z_0, z_1)$$

proving that $\log_g x_0 = \log_y z_0$ or $\log_g x_1 = \log_y z_1$. The signature is $\sigma = (t, z_0, z_1, \pi_{\mathrm{eq,or}})$.

Verification To verify a signature $\sigma = (t, z_0, z_1, \pi_{\mathrm{eq,or}})$ on a message m under verification key $vk = (x_0, x_1)$, compute $y = H_1(t, m)$ and verify that $\pi_{\mathrm{eq,or}}$ is a proof of the claim that $\log_g x_0 = \log_y z_0$ or $\log_g x_1 = \log_y z_1$ by checking that $\mathsf{ZVfy}_{\mathrm{eq,or}}(\pi_{\mathrm{eq,or}}; x_0, x_1, y, y, z_0, z_1) = 1$.

The correctness of the scheme follows directly from the correctness of the non-interactive zero knowledge proof.

Theorem 4. *Let $\mathcal{S}$ be a forger for the signature scheme Σ_{mu} in the random oracle model, making at most l hash queries (with no repeating queries), interacting with at most μ users and asking for at most n signatures. Then there exists adversaries $\mathcal{B}$ and $\mathcal{C}$ against DDH and CDH, respectively, such that*

$$\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma_{mu}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{B}) + 2\,\mathrm{Succ}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{C}) + \frac{nl}{p^2} + \frac{nl}{|R|} + \frac{1}{p} + \frac{ln}{2p} + \frac{l+1}{p}.$$

The difference in running time is linear in $\mu + l + n$.

3.3 Proof of Theorem 4

The proof proceeds as a sequence of games between a simulator and a forger for the signature scheme. For each game G_i, there is an event E_i corresponding to the adversary "winning" the game. We prove bounds on the differences $\Pr[E_i] - \Pr[E_{i+1}]$ for consecutive games, and finally bound the probability $\Pr[E_5]$ for the last game. Our claim follows directly from these bounds in the usual fashion.

Game 0 The first game is the standard multi-user signature game where μ key pairs are generated. The adversary $\mathcal{S}$ may ask for signatures on any message under any un-revealed key. The adversary may also ask for any signing key.

Let E_0 be the event that the adversary produces a valid forgery (and let E_i be the corresponding event for the remaining games). We have that

$$\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma_{\mathrm{mu}}}(\mathcal{S}) = \Pr[E_0]. \tag{1}$$

Game 1 In this game, when the adversary asks for a signature on a message, instead of creating the zero knowledge proofs using $\mathsf{ZPrv}_{\mathrm{eq,or}}(\dots)$, we sample challenges β_0, β_1 and create a simulated proof using $\mathsf{ZSim}_{\mathrm{eq,or}}(\dots;\beta_0,\beta_1)$ and then reprogram the random oracle H_2 such that $H_2(\dots) = \beta_0 + \beta_1 \bmod p$.

Since the challenge in the simulated conversation has been chosen uniformly at random, this change is not observable unless the random oracle H_2 had been queried at this exact position before the reprogramming, and the reprogramming attempt fails.

As discussed in Section 2, the simulator will choose any particular proof with probability at most $1/p^2$, so the probability that any reprogramming attempt fails is at most l/p^2. The probability of the exceptional event, that at least one of the n attempts fail, is then upperbounded by nl/p^2, giving us

$$|\Pr[E_1] - \Pr[E_0]| \leq nl/p^2. \tag{2}$$

Game 2 Next, when the adversary asks for a signature on a message, instead of just computing the hash of the message directly, we sample $\xi \leftarrow \{0, 1, \ldots, p-1\}$, compute $y \leftarrow g^{\xi}$ and then reprogram the random oracle H_1 such that $H_1(t, m) = y$.

Since t is sampled from a set R with $|R|$ elements, if there are at most l hash queries in the game, the probability that any one reprogramming attempt fails is at most $l/|R|$. The probability of the exceptional event, that at least one of the n attempts fail, is then upperbounded by $nl/|R|$, giving us

$$|\Pr[E_2] - \Pr[E_1]| \leq \frac{nl}{|R|}. \tag{3}$$

Game 3 We now modify the key generation algorithm used by the simulator, so that instead of sampling x_{1-b} from $\mathbb{G}$, it samples $a_{1-b} \leftarrow \{0, 1, \ldots, p-1\}$ and computes $x_{1-b} \leftarrow g^{a_{1-b}}$. The experiment stores the a_{1-b} along with a_b as (b, a_0, a_1). However, when the adversary asks for a signing key, the simulator still returns (b, a_b).

In the original key generation algorithm, x_{1-b} is sampled from the uniform distribution on the group. The key value a_{1-b} is sampled from the uniform distribution on $\{0, 1, \ldots, p-1\}$, so x_{1-b} will also be sampled from the same distribution in this game. Since a_{1-b} is never used and never revealed, this game is indistinguishable from the previous game and

$$\Pr[E_3] = \Pr[E_2]. \tag{4}$$

Game 4 We now modify the signing algorithm used by the simulator, so that instead of sampling z_{1-b} from $\mathbb{G}$, we compute $z_{1-b} \leftarrow y^{a_{1-b}}$.

The distinguisher has access to an oracle $\mathcal{O}$.
It proceeds to run Game 4 with $\mathcal{S}$ with the following modifications:

1. The key generation algorithm used by the simulator queries its oracle with 0 and gets the reply (x, y, z). It sets $x_{1-b} \leftarrow x$ and discards y, z.
2. The signing algorithm used by the simulator, when signing with the signing key (b, a_0, a_1) corresponding to the public key (x_0, x_1) with x_{1-b} equal to the first group element of the ith oracle response, the simulator sends i to its oracle and receives the response (x_{1-b}, y, z). It then uses y unchanged as the hash value and sets $z_{1-b} \leftarrow z$.

If $\mathcal{S}$ eventually produces a valid forgery, the distinguisher outputs 0. Otherwise it outputs 1.

Fig. 1. $\mu + n$-DDH distinguisher $\mathcal{B}'$ used in the proof of Theorem 4.

To bound the difference between this game and the previous one, we need the auxillary $\mu + n$-DDH distinguisher $\mathcal{B}'$ given in Figure 1.

Regardless of which oracle $\mathcal{B}'$ interacts with, the verification key element x_{1-b} and y are sampled from the uniform distribution on $\mathbb{G}$, just like it is in both this game and the previous game.

When the adversary $\mathcal{B}'$ interacts with the oracle $\mathcal{O}_1$ which returns random tuples, then the oracle samples its third coordinate from the uniform distribution on $\mathbb{G}$, and this value is independent of all other values. Thus z_{1-b} is sampled from the uniform distribution on $\mathbb{G}$, just like in Game 3.

When the adversary $\mathcal{B}'$ interacts with the oracle $\mathcal{O}_0$ which returns DDH tuples, then (x_{1-b}, y, z_{1-b}) is a DDH tuple, just like in Game 4.

We conclude that $\mathcal{B}'$ perfectly simulates the two games, depending on which oracle it has access to, and by Theorem 2 it follows that there exists a DDH adversary $\mathcal{B}$ such that

$$|\Pr[E_4] - \Pr[E_3]| = |\Pr[\mathcal{B}'^{\mathcal{O}_0}] - \Pr[\mathcal{B}'^{\mathcal{O}_1}]| \leq \mathrm{Adv}^{\mathsf{DDH}}_{\mathbb{G},g}(\mathcal{B}) + \frac{1}{p}. \tag{5}$$

At this point, we observe that in this game, the adversary has no information about b for any of the unrevealed keys.

Game 5 We now modify the signing algorithm, so that instead of computing $z_{1-b} \leftarrow y^{a_{1-b}}$, we compute $z_{1-b} \leftarrow x_{1-b}^{\xi}$, where ξ comes from the computation $y \leftarrow g^{\xi}$ introduced in Game 2.

Since $y^{a_{1-b}} = (g^{\xi})^{a_{1-b}} = (g^{a_{1-b}})^{\xi} = x_{1-b}^{\xi}$, the adversary cannot detect this change. Therefore

$$\Pr[E_5] = \Pr[E_4]. \tag{6}$$

Note that in this game, the fake signing key a_{1-b} introduced in Game 3 is no longer actually used for anything except computing x_{1-b}.

The solver takes $(x_1, \ldots, x_l, y_1, \ldots, y_l)$ as input.
It proceeds to run Game 5 with $\mathcal{S}$ with the following modifications:

1. When the key generation algorithm used by the simulator generates the ith key pair, it sets $x_{1-b} \leftarrow x_i$.
   The algorithm remembers (x_{1-b}, i).
2. When the forger $\mathcal{S}$ queries the hash oracle with the jth value (t, m) that has not been seen before, the hash oracle sets $y \leftarrow y_j$ and reprograms the hash oracle so that $H_1(t, m) = y$.
   The algorithm remembers (t, m, j).

When the signature forger outputs a valid signature $(t, z_0, z_1, \pi_{\mathrm{eq,or}})$ for a message m under an unrevealed key (b, a_0, a_1) with corresponding public key (x_0, x_1), the solver recalls (x_{1-b}, i) and (t, m, j) and outputs

$$(i, j, z_{1-b}).$$

Fig. 2. l-CDH adversary $\mathcal{C}'$ used in the proof of Theorem 4.

Suppose the adversary wins Game 5 by outputting a signature $(t, z_0, z_1, \pi_{\text{eq,or}})$ for a message m and hash $y = H_1(t, m)$ under the verification key (x_0, x_1) with signature key (b, a_0, a_1).

Since we can recover a tuple $(x_0, x_1, y, y, z_0, z_1)$ and a proof $\pi_{\text{eq,or}}$, we would like to apply Theorem 3. But this is tricky because we simulate proofs and reprogram the random oracle involved in the theorem. However, since the adversary's forgery must be on a message that has not been signed by our signature oracle, the forgery cannot involve any value for which we have reprogrammed the random oracle, unless the adversary has found a collision in H_1. This collision must involve a (t, m) pair from a signing query, which means that the probability of a collision is at most $ln/2p$.

When there is no such collision, Theorem 3 applies and we know that either $\log_y z_0 = \log_g x_0$ or $\log_y z_1 = \log_g x_1$ (or both), except with probability $(l+1)/p$.

Since the forger $\mathcal{S}$ has no information about b, it follows that if equality holds for one of the discrete logarithm pairs, then $\log_y z_{1-b} = \log_g x_{1-b}$ at least half the time.

Consider the l-CDH adversary $\mathcal{C}'$ given in Figure 2. It is clear that it perfectly simulates Game 5 with the adversary $\mathcal{S}$. Furthermore, when the output signature satisfies $\log_y z_{1-b} = \log_g x_{1-b}$, the l-CDH adversary outputs the correct answer. By Theorem 1 there exists a CDH adversary $\mathcal{C}$ such that

$$\Pr[E_5] \leq 2\,\mathrm{Succ}^{\mathsf{CDH}}_{\mathbb{G},g}(\mathcal{C}) + \frac{ln}{2p} + \frac{l+1}{p}. \tag{7}$$

Theorem 4 now follows from equations (1)–(7).

4 Key Exchange

Now we describe our construction of a tightly-secure key exchange protocol, which uses the signature scheme presented above as a subroutine and additionally resolves the "commitment-problem" sketched in the introduction. This yields the first authenticated key exchange protocol which does not require a trusted setup, has tight security, and truly practical efficiency. The security proof is in the Random Oracle Model [6].

4.1 Security Model

Up to minor notational changes and clarifications, our security model is identical to the model from [2], except that we use the recent approach of Li and Schäge [41] to define "partnering" of oracles. Furthermore, we include a "sender identifier" into the Send query (its relevance is discussed below). As in [2], we let the adversary issue more than one Test-query, in order to achieve tightness in this dimension, too.

Execution Environment. We consider μ parties $P_1, \ldots, P_\mu$. Each party P_i is represented by a set of ℓ oracles, $\{\pi_i^1, \ldots, \pi_i^\ell\}$, where each oracle corresponds to a single protocol execution, and $\ell \in \mathbb{N}$ 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 P_i and to the public keys of all other parties, and maintains a list of internal state variables that are described in the following:

- ρ_i^s is the randomness tape of π_i^s.
- Pid_i^s stores the identity of the intended communication partner.
- $\Psi_i^s \in \{\texttt{accept}, \texttt{reject}\}$ 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.

For each oracle π_i^s these variables are initialized as $(\mathsf{Pid}_i^s, \Psi_i^s, k_i^s) = (\emptyset, \emptyset, \emptyset)$, where $\emptyset$ denotes the empty string. 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 $k_i^s \neq \emptyset \iff \Psi_i^s = \texttt{accept}$.

Attacker Model. The attacker $\mathcal{A}$ interacts with these oracles through queries. Following the classical Bellare-Rogaway approach [7], we consider an active attacker that has full control over the communication network, and to model further real world capabilites of an attacker, we provide additionally queries. The $\mathsf{Corrupt}$-query allows the adversary to compromise the long-term key of a party. The Reveal-query may be used to obtain the session key that was computed in a previous protocol instance. The $\mathsf{RegisterCorrupt}$ enables the attacker to register maliciously-generated public keys, and we do not require the adversary to know the corresponding secret key. The Test-query does not correspond to any real world capability of an adversary, but it is used to evaluate the advantage of $\mathcal{A}$ in breaking the security of the key exchange protocol. However, we do not allow reveals of ephemeral randomness, as in [14, 8]. More precisely:

- $\mathsf{Send}(i, s, j, m)$: $\mathcal{A}$ can use this query to send any message m of its choice to oracle π_i^s on behalf of party P_j. The oracle will respond according to the protocol specification and depending on its internal state.
  If $(\mathsf{Pid}_i^s, \Psi_i^s) = (\emptyset, \emptyset)$ and $m = \emptyset$, then this means that $\mathcal{A}$ initiates a protocol execution by requesting π_i^s to send the first protocol message to party P_j. In this case, π_i^s will set $\mathsf{Pid}_i^s = j$ and respond with the first message according to the protocol specification.
  If $(\mathsf{Pid}_i^s, \Psi_i^s) = (\emptyset, \emptyset)$ and $m \neq \emptyset$, then this means that $\mathcal{A}$ sends a first protocol message from party P_j to π_i^s. In this case, π_i^s will set $\mathsf{Pid}_i^s = j$ and respond with the second message according to the protocol specification. This is the only reason why we include the "partner identifier" j in the Send query.
  If $\mathsf{Pid}_i^s = j' \neq \emptyset$ and $j \neq j'$, then this means that the partner id of π_i^s has already been set to j', but the adversary issues a Send-query with $j \neq j'$. In this case, π_i^s will abort by setting $\Psi_i^s = \texttt{reject}$ and responding with $\perp$.

Finally, if π_i^s has already rejected (that is, it holds that $\Psi_i^s = \texttt{reject}$), then π_i^s always responds with $\perp$.
If $\mathsf{Send}(i, s, j, m)$ is the τ-th query asked by $\mathcal{A}$, and oracle π_i^s sets variable $\Psi_i^s = \texttt{accept}$ after this query, then we say that π_i^s has τ-*accepted*.

- $\mathsf{Corrupt}(i)$: This query returns the long-term secret key sk_i of party P_i. If the τ-th query of $\mathcal{A}$ is $\mathsf{Corrupt}(i)$, then we call P_i τ-corrupted, or simply corrupted. If P_i is corrupted, then all oracles $\pi_i^1, \ldots, \pi_i^\ell$ respond with $\perp$ to all queries.
  We assume without loss of generality that $\mathsf{Corrupt}(i)$ is only asked at most once for each i. If $\mathsf{Corrupt}(i)$ has not yet been issued by $\mathcal{A}$, then we say that party i is currently ∞-corrupted.
- $\mathsf{RegisterCorrupt}(i, pk^{(i)})$: This query allows $\mathcal{A}$ to register a new party P_i, $i > \mu$, with public key $pk^{(i)}$. If the same party P_i is already registered (either via $\mathsf{RegisterCorrupt}$-query or $i \in [\mu]$), a failure symbol $\perp$ is returned to $\mathcal{A}$. Otherwise, P_i is registered, the pair $(P_i, pk^{(i)})$ is distributed to all other parties.
  Parties registered by this query are called *adversarially-controlled*. All parties controlled by the adversary are defined to be 0-corrupted. Furthermore, there are no oracles corresponding to these parties.
- $\mathsf{Reveal}(i, s)$: In response to this query π_i^s returns the contents of k_i^s. Recall that we have $k_i^s \neq \emptyset$ if and only if $\Psi_i^s = \texttt{accept}$. If $\mathsf{Reveal}(i, s)$ is the τ-th query issued by $\mathcal{A}$, we call π_i^s τ-revealed. If $\mathsf{Reveal}(i, s)$ has not (yet) been issued by $\mathcal{A}$, then we say that oracle π_i^s is currently ∞-revealed.
- $\mathsf{Test}(i, s)$: If $\Psi_i^s \neq \texttt{accept}$, then a failure symbol $\perp$ is returned. Otherwise π_i^s flips a fair coin b_i^s, samples $k_0 \xleftarrow{\$} \mathcal{K}$ at random, sets $k_1 = k_i^s$, and returns $k_{b_i^s}$.
  The attacker may ask many Test-queries to different oracles, but not more than one to each oracle. Jumping slightly ahead, we note that there exists a trivial adversary that wins with probability $1/4$, if we allow Test-queries of the above form to "partnered" oracles. In order to address this, we have to define partnering first. Then we will disallow Test-queries to partnered oracles in the AKE security definition (Definition 7).

Partnering and original keys. In order to exclude trivial attacks, we need a notion of "partnering" of two oracles. Bader *et al.* [2] base their security definition on the classical notion of *matching conversations* of Bellare and Rogaway [7]. However, Li and Schäge [41] showed recently that this notion is error-prone and argued convincingly that it captures the cryptographic intuition behind "secure authenticated key exchange" in a very conservative way. This is because the strong requirement of matching conversation even rules out theoretical attacks based on "benign malleability" (e.g., efficient re-randomizability of signatures), which does not match any practical attacks, but breaks matching conversations, and thus seems stronger than necessary. This may hinder the design of simple and efficient protocols.

The new idea of [41] is to based "partnering" on an *original key* of a pair of oracles (π_i^s, π_j^t). Recall that we consider an oracle π_i^s as a deterministic algorithm,

but with access to a fixed randomness tape ρ_i^s. The *original key* $K_0(\pi_i^s, \pi_j^t)$ of a pair of oracles (π_i^s, π_j^t) consists of the session key that both oracles would have computed by executing the protocol with each other, and where π_i^s sends the first message. Note that $K_0(\pi_i^s, \pi_j^t)$ depends deterministically on the partner identities i and j and the randomness ρ_i^s and ρ_j^t of both oracles. Note also that for certain protocols it may not necessarily hold that $K_0(\pi_i^s, \pi_j^t) = K_0(\pi_j^t, \pi_i^s)$, thus the order of oracles in the K_0 function matters.

Definition 6 (Partnering). *We say that oracle π_i^s is partnered to oracle π_j^t, if at least one of the following two condition holds.*

1. *π_i^s has sent the first protocol message and it holds that $k_i^s = K_0(\pi_i^s, \pi_j^t)$*
2. *π_i^s has received the first protocol message and it holds that $k_i^s = K_0(\pi_j^t, \pi_i^s)$*

Security experiment. Consider the following game, played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The game is parameterized by two numbers μ (the number of honest identities) and ℓ (the maximum number of protocol executions per party).

1. $\mathcal{C}$ generates μ long-term key pairs $(sk^{(i)}, pk^{(i)}), i \in [\mu]$. It provides a $\mathcal{A}$ with all public keys $pk^{(1)}, \ldots, pk^{(\mu)}$.
2. The challenger $\mathcal{C}$ provides $\mathcal{A}$ with the security experiment, by implementing a collection of oracles $\{\pi_i^s : i \in [\mu], s \in [\ell]\}$. $\mathcal{A}$ may adaptively issue Send, $\mathsf{Corrupt}$, Reveal, $\mathsf{RegisterCorrupt}$ and Test queries to these oracles in arbitrary order.
3. At the end of the game, $\mathcal{A}$ terminates and outputs (i, s, b'), where (i, s) specifies an oracle π_i^s and b' is a guess for b_i^s.

We write $G_{\Pi}(\mu, \ell)$ to denote this security game, carried out with parameters μ, ℓ and protocol Π.

Definition 7 (AKE Security). *An attacker $\mathcal{A}$ breaks the security of protocol Π, if at least one of the following two events occurs in $G_{\Pi}(\mu, \ell)$:*

Attack on authentication. Event $\mathsf{break}_{\mathrm{A}}$ denotes that at any point throughout the security experiment there exists an oracle π_i^s such that all the following conditions are satisfied.

1. *π_i^s has accepted, that is, it holds that $\Psi_i^s = \mathtt{accept}$.*
2. *It holds that $\mathsf{Pid}_i^s = j$ for some $j \in [\mu]$ and party P_j is ∞-corrupted.*
3. *There exists no unique oracle π_j^t that π_i^s is partnered to.*

Attack on key indistinguishability. We assume without loss of generality that $\mathcal{A}$ issues a $\mathsf{Test}(i, s)$-query only to oracles with $\Psi_i^s = \mathtt{accept}$, as otherwise the query returns always returns $\perp$. We say that event $\mathsf{break}_{\mathrm{KE}}$ occurs if $\mathcal{A}$ outputs (i, s, b') and all the following conditions are satisfied.

1. *$\mathsf{break}_{\mathrm{A}}$ does not occur throughout the security experiment.*

2. *The intended communication partner of* π_i^s *is not corrupted before the* $\mathsf{Test}(i,s)$*-query. Formally, if* $\mathsf{Pid}_i^s = j$ *and* π_i^s *is* τ*-tested, then it holds that* $j \leq \mu$ *and party* P_j *is* τ'*-corrupted with* $\tau' \geq \tau$*.*
3. *The adversary never asks a* Reveal*-query to* π_i^s*. Formally, we require that* π_i^s *is* ∞*-revealed throughout the security experiment.*
4. *The adversary never asks a* Reveal*-query to the partner oracle of* π_i^s*.*[8] *Formally, we demand that* π_j^t *is* ∞*-revealed throughout the security experiment.*
5. $\mathcal{A}$ *answers the* Test*-query correctly. That is, it holds that* $b_i^s = b'$*, and if there exists an oracle* π_j^t *that* π_i^s *is partnered to, then* $\mathcal{A}$ *must not have asked* $\mathsf{Test}(j,t)$*.*

The advantage *of the adversary* $\mathcal{A}$ *against AKE security of* Π *is*

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

We say that $\mathcal{A}$ $(\epsilon_{\mathcal{A}}, t, \mu, \ell)$-breaks Π *if its running time is* t *and* $\mathrm{Adv}_{\Pi}^{\mathsf{AKE}}(\mathcal{A}) \geq \epsilon_{\mathcal{A}}$*. Again, we include the running time of the security experiment into the running time of* $\mathcal{A}$ *(cf. Remark 1).*

Remark 2. Note that Definition 7 defines event $\mathsf{break}_{\mathrm{KE}}$ such that it occurs only if $\mathsf{break}_{\mathrm{A}}$ does *not* occur. We stress that this is without loss of generality. It makes the two possible ways to break the security of the protocol mutually exclusive, which in turn makes the reasoning in a security proof slightly simpler.

Remark 3. Note that an oracle π_i^s may be corrupted before the $\mathsf{Test}(i,s)$-query. This provides security against *key-compromise impersonation* attacks. Furthermore, the communication partner π_j^t may be corrupted as well, but only after π_i^s has accepted (to prevent the trivial impersonation attack), which provides *forward security* (aka. *perfect forward secrecy*).

4.2 Construction

In this section, we construct our protocol, based on a digital signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$, a prime-order group $(\mathbb{G}, g, p)$, and cryptographic hash functions $G : \{0,1\}^* \to \{0,1\}^{\kappa}$ and $H : \mathbb{G} \to \{0,1\}^d$ for some $d \in \mathbb{N}$.

Protocol description. Let us consider a protocol execution between two parties Alice and Bob. The protocol is essentially the classical "signed Diffie-Hellman" with hashed session key, except that there is an additional first message which contains a cryptographic commitment to the Diffie-Hellman share g^a of the initiator of the protocol. This adds another message to the protocol, but is an important ingredient to achieve tightness, along the lines sketeched in the introduction. We stress that this additional message does not increase the latency of the protocol. That is, the protocol initiator is able to send cryptographically-protected payload data after one round-trip times (RTTs), exactly as with ordinary signed Diffie-Hellman.

[8] Note that conditions 1. and 2. together imply that there exists a unique oracle π_j^t that π_i^s is partnered to, as otherwise $\mathsf{break}_{\mathrm{A}}$ occurs.

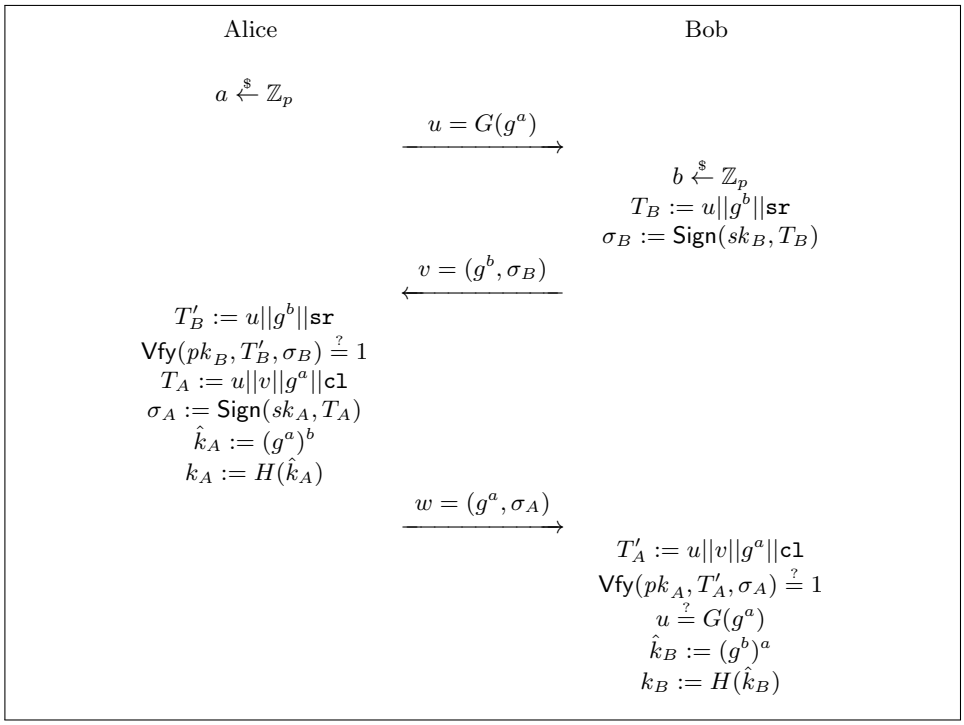

Fig. 3. Basic protocol outline.

Each party is in possession of a long-term key pair $(pk, sk) \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$ for signature scheme Σ. We write (pk_A, sk_A) and (pk_B, sk_B) to denote the key pair of Alice and Bob, respectively. If Alice initiates a key exchange, then both parties proceed as follows.

1. Alice chooses a random exponent $a \xleftarrow{\$} \mathbb{Z}_p$, computes $u := G(g^a)$, and sends u to Bob.
2. When Bob receives u, he picks $b \xleftarrow{\$} \mathbb{Z}_p$ and defines its local transcript of messages as $T_B = u||g^b||\mathtt{sr}$, where $\mathtt{sr}$ is a constant that indicates that Bob acts as a server in this session. Then it computes $\sigma_B := \mathsf{Sign}(sk_B, T_B)$, and responds with $v := (g^b, \sigma_B)$ to Alice.
3. When Alice receives $v := (g^b, \sigma_B)$, she first defines her local view of Bob's transcript as $T'_B = u||g^b||\mathtt{sr}$ and checks $\mathsf{Vfy}(pk_B, T'_B, \sigma_B) \stackrel{?}{=} 1$. If not, then she terminates the protocol execution and sets $\Psi_A := \mathtt{reject}$. Otherwise, she defines her local transcript as $T_A = u||v||g^a||\mathtt{cl}$, where $\mathtt{cl} \neq \mathtt{sr}$ is a constant indicating that Alice acts as a client. Then she computes $\sigma_A := \mathsf{Sign}(sk_A, T_A)$ and sends $w := (g^a, \sigma_A)$ to Bob. Furthermore, she first computes an "internal Diffie-Hellman key" $\hat{k}_A = g^{ab}$, and then the actual session key as $k_A = H(\hat{k}_A)$, and sets $\Psi_A := \mathtt{accept}$.

4. When Bob receives $w := (g^a, \sigma_A)$, he first defines his local view of Alice's transcript as $T'_A = u||v||g^a||\texttt{cl}$ and checks whether $\mathsf{Vfy}(pk_A, T'_A, \sigma_A) = 1$ and whether g^a matches the commitment from the first message, that is, it holds that $u = G(g^a)$. If one of these checks fails, then he sets $\Psi_B := \texttt{reject}$ and terminates. Otherwise he first computes its "internal Diffie-Hellman key" $\hat{k}_B = g^{ab}$, and then the actual session key $k_B = H(\hat{k}_B)$, and sets $\Psi_A := \texttt{accept}$.

Remark 4. We make the "internal Diffie-Hellman key" explicit in the above description, because it will be useful to refer to it in order to define a certain event in the security proof.

Remark 5. We point out that the signatures σ_A and σ_B over $T_A = u||v||g^a||\texttt{cl}$ and $T_B = u||g^b||\texttt{sr}$ protect the whole message transcripts, which is more than actually necessary for our security proof (for which signing g^a and g^b, respectively, would actually be sufficient). However, this is not only a more conservative design, but also facilitates a future security proof of the protocol in a security model based on matching conversations, such as the one from [2].

This seems easily possible, by instantiating the protocol with a *strongly* $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-secure signature scheme in the sense of [13]. Indeed, our signature scheme can easily be made tight strongly-unforgeable, by applying the generic transformation of [49], but this would increase the size of signatures by one group element and one exponent. We leave it as an interesting open problem to prove tight strong $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security *directly* for our signature scheme, without increasing the size of signatures.

Correctness. It is straightforward to verify that this protocol is correct.

Efficiency and latency. At a first glance, our protocol seems less efficient than ordinary signed Diffie-Hellman, because the additional message u adds another protocol round and thus latency. We stress that this is actually not the case, for typical applications. Consider a setting where Alice (a client) wants to send cryptographically protected payload data to a server (Bob). To this end, she initiates the protocol by sending message u. Then she waits for message v, which takes about 1 RTT (round trip time). Finally, she computes message w. At this time Alice has already accepted the key exchange, in particular she has computed the key k_A. This means that she can immediately send cryptographically protected payload data along with message w. Thus, the latency overhead of our protocol, defined as the time that Alice has to wait before she can send cryptographically protected payload, is only 1 RTT.

Now let us compare this to standard signed Diffie-Hellman, which essentially corresponds to our protocol restricted to messages v and w, without the additional commitment message u. In the same setting as above, the client Alice would now send the first protocol message v and then wait for w, which again takes 1 RTT. Only then is Alice able to compute the session key, and use it to send cryptographically protected payload. Thus, even though one message less is sent, it still takes 1 RTT before the session key can be used by Alice.

Thus, while our tightly-secure protocol uses an additional message u, this message does not increase the latency of key establishment at all. Furthermore, message u can be as small as 20-32 bytes in practice, such that the total communication overhead incurred by the key exchange protocol is not significantly increased. At the same time, the best known security proof of signed Diffie-Hellman has even *quadratic* security loss. In contrast, our protocol achieves tightness with only constant security loss, without significantly increasing latency or communication complexity.

Efficiency in real-world PKI settings. As usual in cryptographic theory, our security model considers a setting where each party "magically" has access to all public keys of all other parties. In practice, this is not realistic. Instead, in typical real-world protocols like TLS [20] public keys are typically exchanged within the protocol, along with certificates attesting their authenticity. Often this requires additional protocol rounds, and thus adds further messages and latency to the protocol.

We point out that our protocol does not require any such additional protocol rounds when used in a real-world PKI setting. Concretely, we could simply extend message v to $v = (g^b, \sigma_B, pk_B, \mathtt{cert}_B)$, where $(pk_B, \mathtt{cert}_B)$ is the certified public key of Bob. Message w would be adopted accordingly to $w = (g^a, \sigma_A, pk_A, \mathtt{cert}_A)$, where $(pk_A, \mathtt{cert}_A B)$ is the certified public key of Alice.

Preventing unknown key-shake (UKS) attacks. Blake-Wilson and Menezes [9] introduced UKS attacks, where a party Alice can be tricked into believing that it shares a key with Eve, even though actually the key is shared with a different party Bob. A simple generic method to prevent such attacks in protocols that use digital signatures for authentication (such as ours) is to include user identities in signatures. In a real-world setting where certified public keys are exchanged during the protocol, one could sign the certificates along with all other messages.

Server-only authentication. Another important real-world application scenario is where only the server is authenticated cryptographically, while the client is not in possession of a long-term cryptographic key pair, and thus the protocol can only achieve unilateral authentication. This setting has been considered e.g. in [38] for TLS, and in [48, 42, 26] for more general key exchange protocols. While we do not model and prove it formally, we expect that our protocol achieves tight security also for server-only authentication, by an adopting the security model from Section 4 and the proof to the unilateral setting. More precisely, in this setting we would consider a security model where we distinguish between client oracles (which are not in possession of a cryptographic long-term key), and server oracles in possession of long-term signature keys. For authentication, the proof is identical, except that event $\mathsf{break}_{\mathrm{A}}$ is restriced to accepting client oracles. For key indistinguishability, we would allow Test-queries only for sessions that involve a Diffie-Hellman share that originates from a client oracle controlled by the experiment (as otherwise the adversary is trivially able to win). In this

case, we are able to embed a DDH challenge exactly as in the proof for mutual authentication.

4.3 Security Proof

Theorem 5. *Consider protocol Π as defined above, where hash functions G and H are modeled as random oracles. Let $\mathcal{A}$ be an adversary that $(t, \mu, \ell, \epsilon_{\mathcal{A}})$-breaks Π. Then we can construct and adversaries $\mathcal{B}_{\mathrm{A}}$ and $\mathcal{B}_{\mathrm{KE}}$ such that:*

1. *Either $\mathcal{B}_{\mathrm{A}}$ (t', ϵ', μ)-breaks the $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ with $t' = O(t)$ and $\epsilon' \geq \epsilon_{\mathcal{A}} - \mu^2\ell^2/p$.*
2. *Or $\mathcal{B}_{\mathrm{KE}}$ (t', ϵ')-breaks the decisional Diffie-Hellman assumption in $(\mathbb{G}, g, p)$ with $t' = O(t)$ and $\epsilon' \geq \epsilon_{\mathcal{A}} - t^2/2^d - \mu^2\ell^2/p - \mu^2\ell^2/2^d - \mu\ell t/p$.*

Note that the reduction is tight in both cases, because in both cases the loss in the success probability amounts only to a negligibly small *additive* term.

Theorem 5 is proven by Lemmas 1 and 2 stated below. The proof of Lemma 1 will be a straightforward reduction to the $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security of the signature scheme. The proof of Lemma 2 contains the actual technical novelty. In particular, it resolves the "commitment problem" sketched in the introduction.

Lemma 1. *From each adversary $\mathcal{A}$ that (t, μ, ℓ, ϵ)-breaks Π according to Definition 7 with $\epsilon_{\mathcal{A}} = \mathsf{break}_{\mathrm{A}}$ we can construct an adversary $\mathcal{B}$ that (t', ϵ', μ)-breaks the $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ with $t' = O(t)$ and $\epsilon' \geq \Pr[\mathsf{break}_{\mathrm{A}}] - (\mu\ell)^2/p$.*

Proof. The proof is a straightforward reduction to the $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security of the signature scheme.

Recall that $\mathsf{break}_{\mathrm{A}}$ implies that there exists an oracle π_i^s with $\Psi_i^s = \mathtt{accept}$ and $\mathsf{Pid}_i^s = j$, where P_j is ∞-corrupted. Thus, $\mathcal{A}$ has never queried $\mathsf{Corrupt}(j)$. Furthermore, there exists no unique oracle π_j^t that π_i^s is partnered to.

In the protocol Π, the group element sent by an oracle depends only on that oracle's random bits. Also, the original key is uniquely determined by the group elements sent by the two oracles. This means that π_i^s will partner with any other oracle π_j^t that sends the group element that π_i^s received, and only such oracles.

We first argue that the probability that there is *more than one* partner oracle is negligibly small. To this end, note that since we are working in a prime order group $\mathbb{G}$, there can only be two different oracles π_j^t, $\pi_{j'}^{t'}$ that are both partnered to π_i^s (and thus π_i^s would compute the same original key with both oracles), if both output the same Diffie-Hellman share $g^{e_j^t} = g^{e_{j'}^{t'}}$. Since the Diffie-Hellman shares are sampled uniformly random over a group of order p and there are at most $\mu\ell$ oracle queries, the probability of such a collision bounded by $(\mu\ell)^2/p$.

Next, we show that if there is no partner oracle at all, then we can break the security of Σ. To this end, we design an algorithm $\mathcal{B}$ that simulates the $G_{\Pi}(\mu, \ell)$ security experiment with the following modifications.

– Instead of generating the keys pairs (pk_i, sk_i) for parties $P_1, \ldots, P_\mu$ by running $(pk_i, sk_i) \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$, $\mathcal{B}$ uses the public keys received from the $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$ security experiment.
– Instead of running $\sigma \xleftarrow{\$} \mathsf{Sign}(sk_i, m)$ to sign a message m, $\mathcal{B}$ makes a query (m, i) to its security experiment, and receives back a signature σ for message m under sk_i.
– When $\mathcal{A}$ queries $\mathsf{Corrupt}(i)$, then $\mathcal{B}$ outputs i, receives back sk_i, and outputs sk_i to $\mathcal{A}$.

It is straightforward to verify that this provides a perfect simulation of the $G_{\Pi}(\mu, \ell)$ security experiment. We have to show that $\mathcal{B}$ breaks the security of $\mathsf{MU\text{-}EUF\text{-}CMA}^{\mathsf{corr}}$-security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ with probability at least $\mathsf{break}_{\mathrm{A}}$.

Per the discussion above, if there is no partner oracle, this means that π_i^s has received a tuple (h, σ) such that $\mathsf{Vfy}(pk_j, u||v||h||\mathrm{cl}, \sigma) = 1$ (or $\mathsf{Vfy}(pk_j, u||h||\mathrm{sr}, \sigma) = 1$, respectively), even though no oracle π_j^t has ever output this message. Hence, $\mathcal{B}$ never made a signing query of the form (m, j) with $m = u||v||h||\mathrm{cl}$ (or $m = u||h||\mathrm{sr}$, respectively). Thus, whenever $\mathsf{break}_{\mathrm{A}}$ occurs and there is no partner oracle, then $\mathcal{B}$ is able to output a valid forgery (j, m, σ).

Finally, note that the running time t' of $\mathcal{B}$ essentially amounts to running $\mathcal{A}$ once and simulating the security experiment $G_{\Pi}(\mu, \ell)$, plus a minor number of additional operations. Thus, we have $t' = O(t)$.

Lemma 2. *From each adversary $\mathcal{A}$ that $(t, \mu, \ell, \epsilon_{\mathcal{A}})$-breaks Π according to Definition 7 with $\epsilon_{\mathcal{A}} = \Pr[\mathsf{break}_{\mathrm{KE}}] - 1/2$ we can construct an adversary $\mathcal{B}$ that (t', ϵ')-breaks the decisional Diffie-Hellman assumption in $(\mathbb{G}, g, p)$ with*

$$t' = O(t) \qquad \text{and} \qquad \epsilon' \geq \epsilon_{\mathcal{A}} - \frac{t^2}{2^d} - \frac{\mu^2 \ell^2}{p} - \frac{\mu^2 \ell^2}{2^d} - \frac{\mu \ell t}{p}$$

Proof. Recall that according to Definition 7 $\mathsf{break}_{\mathrm{KE}}$ occurs only if $\mathsf{break}_{\mathrm{A}}$ does not occur, thus we may assume in the sequel that $\mathsf{break}_{\mathrm{A}}$ does not occur.

Notation. In the sequel we will denote with (pk_i, sk_i) the signature key pair of party P_i. It will furthermore be convenient to distinguish between "client oracles" and "server oracles", because this will improve clarity of our proof. Thus, if an oracle π_i^s outputs the first protocol message u in response to a $\mathsf{Send}(i, s, j, \emptyset)$-query, then we call π_i^s a *client oracle* and denote it with C_i^s. Likewise, if π_i^s receives the first protocol message u and responds with the second message v, we call it a *server oracle* and denote it with S_i^s. For a client oracle C_i^s, we write u_i^s to denote the first protocol message output by this oracle, and w_i^s to denote the third protocol message (if it exists). Likewise, if $\pi_i^s = S_i^s$ is a server oracle, then we denote the message output by this oracle with v_i^s.

Before constructing our actual adversary $\mathcal{B}$, we gradually make a few changes to the original security experiment in a sequence of games. In Game α, we define $\mathsf{query}_H^{(\alpha)}$ as the following event. Event $\mathsf{query}_H^{(\alpha)}$ occurs, if the adversary $\mathcal{A}$ outputs (i, s, b), and it has previously queried $\hat{k}_i^s$ to random oracle H, where $\hat{k}_i^s$ is the "internal Diffie-Hellman key" of oracle π_i^s, as defined above.

Game 0. This is the original $G_{\Pi}(\mu, \ell)$ security experiment, except that we describe a specific way how random oracles G and H are implemented.

The experiment in Game 0 maintains two lists L_G and L_H. Whenever a query to random oracle G on input h is made (either by $\mathcal{A}$ or by the experiment), then the experiment checks whether there exists an entry of the form $(h, z) \in L_G$. If this holds, then it returns $z = G(h)$. Otherwise, it samples $z \xleftarrow{\$} \{0,1\}^d$ uniformly random from the range of G, appends (h, z) to G, and returns z. Random oracle H is simulated accordingly.

Note that we only introduce a specific way to implement random oracles G and H, thus, this is a perfectly indistinguishable implementation of the original security experiment.

Furthermore, we claim that

$$\Pr\left[\mathsf{query}_H^{(0)}\right] \geq \epsilon_{\mathcal{A}} = \Pr\left[\mathsf{break}_{\mathrm{KE}}\right] - 1/2$$

To see this, observe that since H is a random oracle, an adversary $\mathcal{A}$ which outputs (i, s, b), but never queries H on input $\hat{k}_i^s$, receives no information about $k_i^s = H(\hat{k}_i^s)$.

Game 1. Game 1 is identical to Game 0, except that we add an abort condition. If throughout in the security experiment ever two queries $h \neq h'$are made to random oracle G such that $G(h) = G(h')$, then the experiment raises event $\mathsf{abort}_{\mathsf{coll}}$ and terminates. Since Game 1 and Game 0 proceed identical until $\mathsf{abort}_{\mathsf{coll}}$, we have

$$\Pr\left[\mathsf{query}_H^{(1)}\right] \geq \Pr\left[\mathsf{query}_H^{(0)}\right] - \Pr\left[\mathsf{abort}_{\mathsf{coll}}\right]$$

Furthermore, since the hash values for random oracle G are chosen uniformly random from the range $\{0,1\}^d$ of G, the collision probability is bounded by $Q^2 \cdot 2^{-d}$, where Q is an upper bound on the size of list L_G. Since at most one entry is added for each oracle query made by adversary $\mathcal{A}$, and an adversary running in time t can issue at most t oracle queries in the security experiment, we get

$$\Pr\left[\mathsf{query}_H^{(1)}\right] \geq \Pr\left[\mathsf{query}_H^{(0)}\right] - \frac{t^2}{2^d}$$

Game 2. This game contains the main idea behind the reduction. We have to make multiple changes to the security experiment, which must all be made at once in a single game, to preserve consistency.

In Game 2, the experiment initially samples a vector of group elements $V = (g, g^x, g^x, g^z) \in \mathbb{G}^4$ uniformly random, but such that $g^z = g^{xy}$. Using these values, it implements the client and server oracles as follows.

Implementation of client oracles. When $\mathcal{A}$ issues the first $\mathsf{Send}(i, j, s, \emptyset)$ query to client oracle C_i^s, then the experiment proceeds exactly as in Game 1. In particular, it chooses a random exponent $e_i^s \xleftarrow{\$} \mathbb{Z}_q$, computes $u_i^s = G(g^{e_i^s})$, and outputs u_i^s.

Whenever $\mathcal{A}$ issues the second $\mathsf{Send}(i, j, s, v)$-query to a client oracle C_i^s with message $v = (h, \sigma)$, then the experiment first checks whether $\mathsf{Vfy}(pk_j, h||\mathtt{sr}, \sigma) = 1$. If not, then it can trivially implement the rejecting oracle C_i^s. Otherwise, it proceeds as follows.

Handling corrupted communication partners. If party P_j is corrupted (either because $\mathcal{A}$ has previously asked a $\mathsf{RegisterCorrupt}(j, pk^{(j)})$-query and thus $j > \mu$, or because $\mathcal{A}$ asked a $\mathsf{Corrupt}(j)$-query), then the experiment proceeds exactly as in Game 1. In particular, it computes $k_i^s = H(h^{e_i^s})$, $\sigma_i^s = \mathsf{Sign}(sk_i, g^{e_i^s}||\mathtt{c1})$, and returns $w_i^s = (g^{e_i^s}, \sigma_i^s)$.

Handling non-corrupted communication partners. If party P_j is not (yet) corrupted, then note that this means that there must exist a server-oracle S_j^t which has output $v = (h, \sigma)$, as otherwise event $\mathsf{break}_{\mathrm{A}}$ would occur.
The experiment now checks whether the hash under random oracle G of $g^{x+e_i^s}$ is already defined, where g^x is from the DDH challenge. More precisely, it checks whether there exists an entry of the form $(g^{x+e_i^s}, z) \in L_G$. If this holds, then it raises event $\mathsf{abort}_{\mathsf{reprog}}$ and aborts. Otherwise, it removes the entry $(g^{x_i^s}, u_i^s)$ from list L_G, and appends $(g^{x_i^s+e_i^s}, u_i^s)$ instead.
Note here that the experiment retroactively embeds a "blinded" version of the element g^x into message u_i^s, by re-programming random oracle G.
Finally, the experiment has to compute the key $k_i^s = H(h^{x+e_i^s})$. Jumping slightly ahead, we will later define the implementation of the server oracle S_j^t such that $h = g^{y+e_j^t}$ where g^y is from the DDH challenge and $e_j^t \xleftarrow{\$} \mathbb{Z}_q$ is a random blinding term known to the experiment. Thus, the experiment is able to compute the "internal Diffie-Hellman key" as

$$\hat{k}_i^s = g^z \cdot (g^y)^{e_i^s} \cdot (g^x)^{e_j^t} \cdot g^{e_i^s e_j^t}$$

and then the actual session key as $k_i^s = H(\hat{k}_i^s)$.
To argue that this is a correct key, we use that $g^z = g^{xy}$. Therefore it holds that

$$g^z \cdot (g^y)^{e_i^s} \cdot (g^x)^{e_j^t} \cdot g^{e_i^s e_j^t} = g^{xy+ye_i^s+xe_j^t+e_i^s e_j^t} = g^{(x+e_i^s)(y+e_j^t)}$$

Thus, if $\mathsf{abort}_{\mathsf{reprog}}$ does not occur and the DDH challenge is a "real" DDH tuple, then this is perfectly indistinguishable from Game 1.

Simulation of server oracles. Whenever a server oracle S_i^s receives a message u, then the experiment samples $e_i^s \xleftarrow{\$} \mathbb{Z}_p$ and checks whether there exists a client oracle C_j^t which previously has output the message $u = u_j^t$.

Handling uncontrolled u. If there exists no client oracle C_j^t which has output $u = u_j^t$, then we say that u is "uncontrolled". In this case, the experiment computes the response by setting $v_i^s = (g^{e_i^s}, \sigma)$, exactly as in Game 1.
If at any point throughout the security experiment any oracle C_j^t ever outputs a first protocol message u_j^t such that $u_j^t = u$, then the experiment raises event $\mathsf{abort}_{\mathsf{control}}$ and terminates.

Finally, if server oracle S_i^s later receives a value $w = (h, \sigma')$, then the proceeds as in Game 1. In particular, it computes the key correctly as $\hat{k}_i^s = h^{e_i^s}$ and $k_i^s = H(\hat{k}_i^s)$. Note that this provides a perfect simulation of Game 1, provided that no abort occurs.

Handling controlled u. In this case we use that there exists an oracle C_j^t which has output $u = u_j^t$. In order to respond to the first protocol message u_j^t, the experiment computes its response by setting $v_i^s = (g^{y+e_i^s}, \sigma)$, where σ is computed exactly as in the original protocol and g^y is from the DDH challenge.

Furthermore, if the same server oracle S_i^s later receives a value $w = (h, \sigma')$, then the experiment proceeds follows. If $G(h) \neq u_j^t$, then it can trivially implement oracle S_i^s, because the oracle will reject without computing a key.

Otherwise we have to distinguish between two cases. Let us first assume that h was *not* output by the same oracle C_j^t from which the first message u_j^t originates. Since it holds that $G(h) = u_j^t$, this means that event $\mathsf{abort}_{\mathsf{coll}}$ has occured (due to a colliding random oracle query, made either by $\mathcal{A}$ or by the experiment). In this case the experiment aborts, as defined above.

Thus, we may from now on assume that h was output by oracle C_j^t, and thus is controlled by the experiment. By the construction of client oracles described above, this means that the experiment "knows" an exponent e_j^t such that either $h = g^{e_j^t}$, or $h = g^{x+e_j^t}$. Thus, it can compute the "internal Diffie-Hellman key" as

$$\hat{k}_i^s = \begin{cases} H((g^{y+e_i^s})^{e_j^t}) & \text{if } h = g^{e_j^t} \\ H(g^z \cdot (g^y)^{e_i^s} \cdot (g^x)^{e_j^t} \cdot g^{e_i^s e_j^t}) & \text{if } h = g^{x+e_j^t} \end{cases}$$

and then set $k_i^s = H(\hat{k}_i^s)$ Note that again this provides a perfect simulation of Game 1, provided that no abort occurs.

Since Game 2 is a perfect simulation of Game 1, unless an abort occurs, we have

$$\Pr\left[\mathsf{query}_H^{(2)}\right] \geq \Pr\left[\mathsf{query}_H^{(1)}\right] - \Pr\left[\mathsf{abort}_{\mathsf{reprog}}\right] - \Pr\left[\mathsf{abort}_{\mathsf{control}}\right]$$

Let consider $\Pr\left[\mathsf{abort}_{\mathsf{reprog}}\right]$. Recall that this happens, if a uniformly random group element $g^{x+e_i^s} \xleftarrow{\$} \mathbb{G}$, chosen in the simulation of a client oracle, is already contained in the list L_G. Since the order of the group is $|\mathbb{G}| = p$, there are at most $\mu\ell$ client oracles and thus we can bound the collision probability as

$$\Pr\left[\mathsf{abort}_{\mathsf{reprog}}\right] \leq \frac{\mu^2\ell^2}{p}$$

To consider $\Pr\left[\mathsf{abort}_{\mathsf{control}}\right]$, recall that this happens, if a uniformly random string $u \in \{0,1\}^d$, is equal to a first protocol message that the adversary has previously

sent to any server oracle. Again using that there are at most $\mu\ell$ client oracles, the collision probability is bounded by

$$\Pr\left[\mathsf{abort}_{\mathsf{control}}\right] \leq \frac{\mu^2\ell^2}{2^d}$$

In summary, this yields

$$\begin{aligned}\Pr\left[\mathsf{query}_H^{(2)}\right] &\geq \Pr\left[\mathsf{query}_H^{(1)}\right] - \frac{\mu^2\ell^2}{p} - \frac{\mu^2\ell^2}{2^d} \\ &\geq \Pr\left[\mathsf{query}_H^{(0)}\right] - \frac{t^2}{2^d} - \frac{\mu^2\ell^2}{p} - \frac{\mu^2\ell^2}{2^d} \\ &\geq \epsilon_{\mathcal{A}} - \frac{t^2}{2^d} - \frac{\mu^2\ell^2}{p} - \frac{\mu^2\ell^2}{2^d} \end{aligned} \tag{8}$$

Game 3. This game is identical to Game 2, except that now we choose $V = (g, g^x, g^y g^z) \xleftarrow{\$} \mathbb{G}^4$ uniformly random from $\mathbb{G}^4$. We claim that now the probability of $\mathsf{query}_H^{(3)}$ is negligibly small. More precisely, it holds that

$$\Pr\left[\mathsf{query}_H^{(3)}\right] \leq \frac{\mu\ell t}{p} \tag{9}$$

To see this, note that in Game 2 the uniformly random term g^z is only used in the computation of "internal Diffie-Hellman keys", which shifts all internal keys by a random offset z', where $z' \cdot g^{xy} = g^z$. For each query h of $\mathcal{A}$ to random oracle H the probability that h matches any of the at most $\mu\ell$ internal keys is bounded by $\frac{\mu\ell}{p}$. Since the adversary is able issue at most t queries to H, we obtain (9).

Construction of adversary $\mathcal{B}$. Now we are ready to construct the DDH-adversary $\mathcal{B}$. $\mathcal{B}$ receives as input a vector of group elements $W = (g, g^x, g^y, g^z)$, where either $g^z = g^{xy}$ or g^z is uniformly random. Adversary $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine, by proceeding exactly like the experiment from Game 3, except that it uses vector W instead of the vector V. Furthermore, if $\mathsf{query}_H^{(3)}$ occurs, then $\mathcal{B}$ outputs "1". If $\mathcal{A}$ terminates and $\mathsf{query}_H^{(3)}$ did not occur, then it outputs "0".

Analysis of adversary $\mathcal{B}$. Note that if $W = (g, g^x, g^y g^z)$ is random, then this provides a perfect simulation of Game 3. Thus we have

$$\Pr\left[1 \xleftarrow{\$} \mathcal{B} \mid g^z \text{ is random}\right] = \Pr\left[\mathsf{query}_H^{(3)}\right] \leq \frac{\mu\ell t}{p}$$

However, if $W = (g, g^x, g^y g^{xy})$, then this provides a perfect simulation of Game 2, and we have

$$\Pr\left[1 \xleftarrow{\$} \mathcal{B} \mid g^z = g^{xy}\right] = \Pr\left[\mathsf{query}_H^{(2)}\right] \geq \epsilon_{\mathcal{A}} - \frac{t^2}{2^d} - \frac{\mu^2\ell^2}{p} - \frac{\mu^2\ell^2}{2^d}$$

Thus, $\mathcal{B}$ solves the DDH problem with advantage

$$\Pr\left[1 \xleftarrow{\$} \mathcal{B} \mid g^z = g^{xy}\right] - \Pr\left[1 \xleftarrow{\$} \mathcal{B} \mid g^z \text{ is random}\right] \geq \epsilon_{\mathcal{A}} - \frac{t^2}{2^d} - \frac{\mu^2\ell^2}{p} - \frac{\mu^2\ell^2}{2^d} - \frac{\mu\ell t}{p}$$

Furthermore, note that the running time of $\mathcal{B}$ essentially amounts to running $\mathcal{A}$ once, which takes at most t operations, plus a minor number of additional operations required to simulate Game 3, which in total is bounded by $O(t)$.

5 Efficiency Analysis

Let us compare an instantiation of our protocol from Section 4.2, instantiated with our signature scheme from Section 3.2, to plain "signed Diffie-Hellman", instantiated with EC-DSA. The latter is the currently most efficient practical instantiation of an authenticated key exchange protocol over simple groups with *explicit* authentication (in contrast, some protocols, such as NAXOS [40], do not provide explicit authentication via digital signatures, but only implicit authentication via indistinguishability of keys).

We consider a setting where both the signature scheme and the Diffie-Hellman key exchange are instantiated over the same group. This is desirable in practice for many different reasons. Most importantly, it reduces the size of the implementation. This makes the protocol not only faster to implement, but also easier to implement securely (e.g., constant-time and resilient to other side-channels) and easier to maintain, which are very desirable properties, from a real-world security point of view.

Furthermore, an implementation requiring a small codebase or circuit size is particularly desirable for resource-constrained devices, such as IoT devices, where tightness is particularly relevant due to the large number of devices in use.

Computational efficiency. In order to compare the efficiency of protocols, we count the number of exponentiations, as this is the most expensive computation to be performed. Below we will also briefly discuss the potential impact of optimisations.

Our protocol. Each party running our protocol has to perform two exponentiations to perform the Diffie-Hellman key exchange, seven exponentiations to sign a message, and eight exponentiations to verify a signature. In total, this amounts to 17 exponentiations.

Signed Diffie-Hellman. Executing the signed Diffie-Hellman protocol with EC-DSA takes two exponentiations to perform the Diffie-Hellman key exchange, one exponentiation to compute an EC-DSA signature, and two exponentiations to verify a signature. In total, this amounts to 5 exponentiations.

Thus, our protocol requires 3.4 times more exponentiations than signed Diffie-Hellman.

Theoretically-sound instantiations. Let us consider a desired security level equivalent to an 128-bit symmetric key.

Our protocol. The tightness of our security proof allows to instantiate out protocol on a 256-bit elliptic curve group, such as the NIST P-256 curve, independent of the number of users or sessions.

Signed Diffie-Hellman. When instantiating plain "signed Diffie-Hellman", we have to compensate the quadratic security loss of $Q = \mu^2\ell^2$ of the security proof, depending on the number of users μ and the number of sessions ℓ, by choosing a larger group. For instance:

- In a small-to-medium-scale setting with $\mu = 2^{16}$ and $\ell = 2^{16}$, the security loss amounts already to a factor of $Q = 2^{64}$. In order to compensate this with larger parameters, we have to increase the group size by a factor of $Q^2 = 2^{128}$. We can do this by using the NIST P-384 curve.
- In a large-scale setting with $\mu = 2^{32}$ and $\ell = 2^{32}$, the security loss amounts even to a factor of $Q = 2^{128}$. In order to compensate this with larger parameters, we have to increase the group size by a factor of $Q^2 = 2^{256}$, e.g., by using the NIST P-521 curve.

Remark 6. To justify the numbers chosen above, let us consider Facebook as an example. Facebook lists 2.13 billion active users in December 2017, see `https://newsroom.fb.com/company-info/`. Even if we assume that each user performs only a single TLS handshake (that is, only a single login) per month, this amounts to about 2^{31} execution of the TLS protocol per month, and about 2^{34} per year (the lifetime of the certified public key). Since known security proofs for TLS have a quadratic security loss, we thus have a security loss of 2^{68} already in the *single-user* setting where only Facebook is considered.

Comparison of computational efficiency. In order to estimate the time required for one exponentiation for different curves, we consider OpenSSL as an example. OpenSSL is a very widely-used and stable cryptographic library with good performance properties. The benchmark tests of elliptic curve Diffie-Hellman, which analyse the performance of different elliptic curves implemented by OpenSSL, can be run on a system where OpenSSL is installed by executing the command `openssl speed ecdh`.

We ran this benchmark on a MacBook Pro computer with 3.3 GHz Intel Core i7 CPU and 16 GB RAM, running Mac OS Version 10.13.2. Figure 1 summarises the results for the considered NIST curves (P256, P384, P521), as well as suitable alternatives. Note that one ECDH operation for the P384 curve takes about 2.7 times longer than for P256, while for P521 it is even about 7.7 times longer. The results for other families of curves (K233/409/571 and B283/409/571) are comparable.

Comparison of communication complexity. Now let us compare the amount of data to be transmitted for a key exchange. Again, we consider "128-bit security". We assume that each element of an n-bit elliptic group takes $n + 1$ bits, which can be achieved via standard point compression.

Curve	Security level	Time/Operation in s	Operations per s
NIST P256	128	0.0021	476.9
NIST P384	128	0.0056	179.7
NIST P521	128	0.0161	62.0
NIST K233	128	0.0016	640.1
NIST K409	128	0.0068	147.6
NIST K571	128	0.0151	66.4
NIST B283	128	0.0035	284.6
NIST B409	128	0.0074	135.1
NIST B571	128	0.0167	59.8

Table 1. OpenSSL Benchmark Results for NIST Curves

Our protocol. This protocol requires the transmission of two group elements for the Diffie-Hellman key exchange, each consisting of 257 bits, plus two signatures (each consisting of a random 256-bit nonce, two group elements, and four 256-bit exponents, which yields 1794 bits), plus the first protocol message, which corresponds to one 256-bit value, if SHA-256 is used.
In total, this yields $2 \cdot 257 + 2 \cdot 1794 + 256 = 4358$ bytes, which corresponds to ≈ 545 bytes.

Signed Diffie-Hellman. When instantiating plain "signed Diffie-Hellman" with EC-DSA, each party sends one group element plus one signature consisting of two exponents. This yields:

- When using the NIST P-384 curve, this amounts to $2 \cdot 385 + 4 \cdot 384 = 2306$ bits, which corresponds to ≈ 289 bytes.
- In a large-scale setting with the NIST P-521 curve, this amounts to $2 \cdot 522 + 4 \cdot 521 = 3128$ bits, or ≈ 391 bytes.

Conclusion. Even though the absolute number of exponentiations required to run our protocol is larger than for simple signed Diffie-Hellman, it turns out that for small-to-medium-scale settings the overall computational efficiency is already comparable to signed Diffie-Hellman, if the group order is chosen in a theoretically-sound way. For large-scale settings, it is even significantly better. Concretely, the fact that our protocol requires 3.4 times more exponentiations is already almost compensated by the fact that an exponentiation is about 2.7-times more expensive in the small-to-medium-scale setting. Furthermore, given that in the large-scale setting an exponentiation is about 7.7 times more expensive, it turns out that our protocol is even significantly more efficient by a factor greater than 2.25. We note that this pencil-and-paper analysis considers naïve exponentation, and does not yet involve optimisations, such as pre-computations, which usually tend to be more effective if more exponentiations are performed.

The improved computational efficiency comes at only very moderate cost of increased communication complexity, amounting to 256 bytes *for the entire protocol* in the small-to-medium-scale setting, and 154 bytes in the large-scale setting. This holds in comparison to the very minimalistic EC-DSA-signed Diffie-

Hellman protocol, which is of course extremely communication-efficient in comparison to any other protocol with similar properties.

Given that our protocol is the first proposal for a truly *practical* and tightly-secure key exchange protocol, we expect that future work building upon our techniques will be able to improve this further.

References

1. Bader, C.: Efficient signatures with tight real world security in the random-oracle model. In: Gritzalis, D., Kiayias, A., Askoxylakis, I.G. (eds.) CANS 14. LNCS, vol. 8813, pp. 370–383. Springer, Heidelberg (Oct 2014)
2. Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 629–658. Springer, Heidelberg (Mar 2015)
3. Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 273–304. Springer, Heidelberg (May 2016)
4. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Journal of Cryptology (Jan 2018), https://doi.org/10.1007/s00145-018-9280-5
5. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg (May 2000)
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993)
7. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994)
8. 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. LNCS, vol. 9020, pp. 477–494. Springer, Heidelberg (Mar / Apr 2015)
9. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the station-to-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) PKC'99. LNCS, vol. 1560, pp. 154–170. Springer, Heidelberg (Mar 1999)
10. Blazy, O., Kakvi, S.A., Kiltz, E., Pan, J.: Tightly-secure signatures from chameleon hash functions. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 256–279. Springer, Heidelberg (Mar / Apr 2015)
11. Blazy, O., Kiltz, E., Pan, J.: (Hierarchical) identity-based encryption from affine message authentication. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 408–425. Springer, Heidelberg (Aug 2014)
12. Boneh, D.: The decision Diffie-Hellman problem. In: Third Algorithmic Number Theory Symposium (ANTS). LNCS, vol. 1423. Springer, Heidelberg (1998), invited paper
13. Boneh, D., Shen, E., Waters, B.: Strongly unforgeable signatures based on computational Diffie-Hellman. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 229–240. Springer, Heidelberg (Apr 2006)
14. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (May 2001)

15. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (Aug 1993)
16. Chen, J., Wee, H.: Fully, (almost) tightly secure IBE and dual system groups. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 435–460. Springer, Heidelberg (Aug 2013)
17. Chevallier-Mames, B.: An efficient CDH-based signature scheme with a tight security reduction. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 511–526. Springer, Heidelberg (Aug 2005)
18. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (Aug 1994)
19. Di Crescenzo, G., Katz, J., Ostrovsky, R., Smith, A.: Efficient and non-interactive non-malleable commitment. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 40–59. Springer, Heidelberg (May 2001)
20. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (Aug 2008), `https://www.rfc-editor.org/rfc/rfc5246.txt`, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919
21. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987)
22. Fleischhacker, N., Jager, T., Schröder, D.: On tight security proofs for Schnorr signatures. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 512–531. Springer, Heidelberg (Dec 2014)
23. Garg, S., Bhaskar, R., Lokam, S.V.: Improved bounds on security reductions for discrete log based signatures. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 93–107. Springer, Heidelberg (Aug 2008)
24. Gay, R., Hofheinz, D., Kiltz, E., Wee, H.: Tightly CCA-secure encryption without pairings. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 1–27. Springer, Heidelberg (May 2016)
25. Goh, E.J., Jarecki, S.: A signature scheme as secure as the Diffie-Hellman problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 401–415. Springer, Heidelberg (May 2003)
26. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Des. Codes Cryptography 67(2), 245–269 (2013), `\url{https://doi.org/10.1007/s10623-011-9604-z}`
27. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984)
28. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008)
29. Gueron, S., Lindell, Y.: Better bounds for block cipher modes of operation via nonce-based key derivation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 17. pp. 1019–1036. ACM Press (Oct / Nov 2017)
30. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (Apr 1990)
31. Hoang, V.T., Tessaro, S.: The multi-user security of double encryption. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 381–411. Springer, Heidelberg (May 2017)

32. Hofheinz, D., Jager, T.: Tightly secure signatures and public-key encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 590–607. Springer, Heidelberg (Aug 2012)
33. Hofheinz, D., Jager, T., Knapp, E.: Waters signatures with optimal security reduction. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 66–83. Springer, Heidelberg (May 2012)
34. 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.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012)
35. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: Authenticated confidential channel establishment and the security of TLS-DHE. Journal of Cryptology 30(4), 1276–1324 (Oct 2017)
36. 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, Part I. LNCS, vol. 10677, pp. 409–441. Springer, Heidelberg (Nov 2017)
37. Katz, J., Wang, N.: Efficiency improvements for signature schemes with tight security reductions. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) ACM CCS 03. pp. 155–164. ACM Press (Oct 2003)
38. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer, Heidelberg (Aug 2013)
39. Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. In: IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016. pp. 81–96. IEEE (2016), https://doi.org/10.1109/EuroSP.2016.18
40. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (Nov 2007)
41. 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 17. pp. 1343–1360. ACM Press (Oct / Nov 2017)
42. Maurer, U., Tackmann, B., Coretti, S.: Key exchange with unilateral authentication: Composable security definition and modular protocol design. Cryptology ePrint Archive, Report 2013/555 (2013), http://eprint.iacr.org/2013/555
43. Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: Roy, B.K. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (Dec 2005)
44. Paterson, K.G., van der Merwe, T.: Reactive and proactive standardisation of TLS. In: Chen, L., McGrew, D.A., Mitchell, C.J. (eds.) Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10074, pp. 160–186. Springer (2016), https://doi.org/10.1007/978-3-319-49100-4_7
45. Schäge, S.: Tight proofs for signature schemes without random oracles. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 189–206. Springer, Heidelberg (May 2011)
46. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990)

47. Seurin, Y.: On the exact security of Schnorr-type signatures in the random oracle model. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 554–571. Springer, Heidelberg (Apr 2012)
48. Shoup, V.: On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012 (1999), `http://eprint.iacr.org/1999/012`
49. Steinfeld, R., Pieprzyk, J., Wang, H.: How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 357–371. Springer, Heidelberg (Feb 2007)

A The Goh-Jarecki Signature Scheme

Once we have a zero knowledge proof of equality for discrete logarithms, we immediately get a signature scheme [15] where the signature is the hash of the message raised to a secret exponent. The verifier then relies on a proof of equal discrete logarithms to verify that the signature is correct. Goh and Jarecki [25] show that if the hash is randomised, we get a tight reduction to CDH.

Let H_1 be a hash function from a randomness set R and a message space to the group $\mathbb{G}$. The digital signature scheme $\Sigma_{\text{cdh-nizk}}$ works as follows:

Key generation Sample $a \leftarrow \{0, 1, \ldots, p-1\}$ and compute $x \leftarrow g^a$. The signing key is $sk = a$ and the verification key is $vk = x$.

Signing To sign a message m using the signing key $sk = a$, sample $t \leftarrow R$, let $y = H_1(t, m)$, and compute $z = y^a$. Then create a non-interactive zero knowledge proof $\pi_{\text{eq}} \leftarrow \mathsf{ZPrv}_{\text{eq}}(a; x, y, z)$ for the claim that $\log_g x = \log_y z$. The signature is $\sigma = (t, z, \pi_{\text{eq}})$.

Verification To verify a signature $\sigma = (t, z, \pi_{\text{eq}})$ on a message m under verification key $vk = x$, compute $y = H_1(t, m)$ and verify that π_{eq} is a proof of the claim that $\log_g x = \log_y z$ by checking that $\mathsf{ZVfy}_{\text{eq}}(\pi_{\text{eq}}; x, y, z) = 1$.

By simulating the signing oracle using the honest verifier simulator for the equality proof, the strong properties of the equality proof coupled with the tightness results on l-CDH gives us a tight reduction for a single user [25]. Via random self-reducibility of CDH, this result can tightly be extended to a multi-user setting, but only *without* corruptions, which is not very realistic for many practical applications.