Round-Optimal Privacy Preserving Authenticated Key Exchange Even for Incomplete Sessions

Xavier Bultel[0000−0002−8309−8984] and Khouredia Cisse[0009−0001−7203−7756]

INSA Centre Val de Loire, Université d'Orléans, Inria, France
{xavier.bultel,khouredia.cisse}@insa-cvl.fr

Abstract. Several modern applications, such as Signal or WireGuard, use efficient Noise-like *implicit* authentication key exchanges that require only a small number of exponentiations and two interactions. These protocols have been proven to be secure under the 'strong Diffie–Hellman' (SDH) assumption in the random oracle model (ROM). At ESORICS 2021, Ramacher, Slamanig and Weninger presented an extension to the implicit authenticated key exchange security model, which enables strong privacy properties to be captured in addition to key-secrecy, including man-in-the-middle privacy (ensuring privacy even if a session is interrupted) and forward privacy. They also proposed a protocol to instantiate their model. In this paper, we present an efficient Noise-like protocol that achieve privacy in this model. Our protocol is as efficient as Noise-like protocols that do not guarantee privacy in terms of exponentiations. Moreover, our protocol requires three interactions, which is optimal for this privacy model. It is also more efficient than the ESORICS 2021 protocol in terms of both exponentiations and interactions. Finally, we propose another round-optimal protocol, slightly less efficient in terms of exponentiations, but secure under the CDH assumption in the ROM.

Keywords: Authenticated Key Exchange · Privacy · Provable security.

1 Introduction

End-to-end encryption and key exchange protocols are among the most fundamental tools in cryptography and have been widely used in practice for several decades. These protocols must be resistant to man-in-the-middle attacks, whereby an active attacker can alter communications. A standard method involves running a Diffie–Hellman key exchange, whereby Alice sends an ephemeral key $X = g^x$ and Bob sends $Y = g^y$, and the shared secret key is $K = g^{xy}$. Alice and Bob are also equipped with long-term signature keys generated a priori, which allows them to sign the transcript of the key exchange (i.e. the pair (X, Y)). As Bob already knows X when he sends Y, he can send his signature at the same time. However, Alice must send her signature after receiving Y. This method therefore requires three interactions. If any of these interactions are altered, Alice and Bob can detect this and abort the protocol, making active

attacks ineffective. Under the impetus of the Noise framework[1], more efficient *implicit* authenticated key exchange protocols have been deployed in several real-world applications (e.g. Signal [11] or WireGuard [7]). In these protocols, authentication is implicit in the sense that users no longer sign the transcript, saving the last interaction, but instead use their long-term secret keys, in addition to g^{xy}, to generate the shared key. Therefore, even if an adversary can alter the interactions, they will not be able to share keys with Alice and Bob, meaning the attack will be ineffective for decrypting subsequent messages.

In [6], Cohn-Gordon *et al.* propose an optimised version of this type of protocol (and provide proof of optimal security in terms of reduction loss): Alice and Bob have long-term public keys $U = g^u$ and $V = g^v$, they exchange $X = g^x$ and $Y = g^y$, then compute the shared secret key $K = H(U\|V\|X\|Y\|g^{uy}\|g^{vx}\|g^{xy})$, where H is a hash function modeled as a random oracle. Knowledge of the long-term keys is necessary to compute Y^u for Alice and X^v for Bob, and knowledge of the ephemeral keys is necessary to compute, conversely, U^x for Alice and Y^v for Bob, which prevents the adversary from computing a key shared with Alice by impersonating Bob, and vice versa. Forward secrecy is ensured by g^{xy}; if the adversary finds the long-term keys a posteriori, the element g^{xy} does not depend on them, which prevents them from guessing K. Note that in this work, the authors implicitly assume that the responder, Bob, knows a priori that his interlocutor is Alice before receiving the first message containing X (otherwise he would not be able to computes U^y). In practice, Alice would have to send her identity to Bob in clear text before the protocol (since Alice and Bob do not have a shared key a priori). As a result, by its very nature, this protocol does not protect user identity.

In [13], Ramacher et al. propose to extend the model given by Cohn-Gordon et al. in order to capture two additional strong privacy properties: man-in-the-middle (MITM) privacy and forward privacy. However, this time the initiator must explicitly send their identity to the responder during the protocol, to prevent an adversary from deducing it. In their setup, Alice (the initiator) knows the long-term public key of the intended recipient (Bob). However, the responder does not initially know the identity of the party contacting them and only learns it during protocol execution. In MITM privacy, they consider an active adversary whose goal is to guess the identity of one of the two parties whose long-term keys are not corrupted during the execution of the protocol. Note that this model differs from other similar models in the literature in that it captures privacy even for incomplete sessions. In forward privacy, they consider an adversary attempting to guess the identity of the parties after the protocol has been successfully executed. The attacker is therefore not active during the interaction, but has access to the transcript and can corrupt the long-term keys of any user after the fact. The authors demonstrate that neither of these two privacy properties implies the other, and provide a protocol that verifies key-secrecy, MITM privacy, and forward privacy.

[1] See http://www.noiseprotocol.org/.

In this article, we propose efficient protocols (in terms of the number of exponentiations and interactions) that verify key-secrecy, MITM privacy, and forward privacy in the models of [6, 13]. We propose a first protocol based on the strong Diffie-Hellman assumption, which is similar to the computational Diffie-Hellman assumption (guessing g^{xy} from $X = g^x$ and $Y = g^y$ in a prime order group is difficult), except that the adversary also has an oracle $\mathsf{stDH}_x(A, Z)$ allowing them to test whether $Z = A^x$. Note that this assumption is required for the protocol proposed in [6]. Our protocol is more efficient than that of [13] and is also optimal in terms of the number of interactions. In terms of exponentiations, our protocol is similar to [6] (which does not have privacy). We also propose a variant of our protocol using Schnorr's signatures of knowledge that can be proven secure and private under the CDH assumption, which is weaker than SDH and is considered more standard. However, this protocol is slightly less computationally efficient, as it requires a few additional exponentiations.

1.1 Technical Overview of our Contributions

Our SDH-based protocol Π_{SDH} is shown in Figure 1, where g is the generator of a group of prime order p, H is a hash function modeled as a random oracle, $\mathsf{Enc}/\mathsf{Dec}$ are the encryption/decryption algorithms of an authenticated symmetric encryption scheme, and ctxt denotes the transcript (X, Y, C) of the protocol. This protocol is similar to Noise, except that Alice and Bob compute the key $K \leftarrow H(g^{vx} \| g^{yx} \| X \| Y)$, then Alice uses it to encrypt her public key (associated with her identity) and sends the ciphertext to Bob. Bob discovers Alice's identity by decrypting this public key (remember that before this, Bob does not know who he is communicating with). After verifying that the public key corresponds to a registered user (in practice, Bob verifies that the public key is certificated), Bob generates the shared key in an implicit authentication manner.

Alice (initiator)		**Bob (responder)**
$\mathsf{sk_i} = u; \mathsf{pk_i} = U = g^u$		$\mathsf{sk_r} = v; \mathsf{pk_r} = V = g^v$
$x \leftarrow \mathbb{Z}_p; \quad X \leftarrow g^x$	$\xrightarrow{X}$	$y \leftarrow \mathbb{Z}_p; \quad Y \leftarrow g^y$
$h_1 \leftarrow (V^x \| Y^x)$	$\xleftarrow{Y}$	$h_1 \leftarrow (X^v \| X^y)$
$\mathsf{K} \leftarrow \mathsf{H}(h_1 \| X \| Y)$		$\mathsf{K} \leftarrow \mathsf{H}(h_1 \| X \| Y)$
$\mathsf{C} \leftarrow \mathsf{Enc_K}(\mathsf{U})$	$\xrightarrow{C}$	$U \leftarrow \mathsf{Dec_K}(\mathsf{C})$
		If U is not a registered public key, Bob aborts the protocol.
$h_2 \leftarrow Y^u$		$h_2 \leftarrow U^y$
$\mathsf{K_i} \leftarrow \mathsf{H}(\mathsf{ctxt} \| h_1 \| h_2)$		$\mathsf{K_r} \leftarrow \mathsf{H}(\mathsf{ctxt} \| h_1 \| h_2)$

Fig. 1. Private authenticated key exchange protocol based on SDH (Π_{SDH}).

Key-secrecy can be proved in a similar way to [6]. Remember that an active adversary cannot re-compute the final keys K_i and K_r, so they cannot use them to determine Alice's or Bob's identity. To attack privacy, the ciphertext containing Alice's public key must therefore be exploited. As long as Bob remains uncorrupted, an adversary who intercepts messages sent by Alice cannot deduce $g^{xv} = V^x = X^v$, and therefore cannot deduce any information about the key K, which appears to be random. Thus, disguessing Alice's identity amounts to attacking the secure encryption, which ensures MITM privacy. On the other hand, for forward privacy, if the adversary accesses the transcript after the key exchange, they are unable to compute $X^y = Y^x = g^{xy}$ even if they know the long-term keys u and v. Once again, the key K seems random to them, and guessing Alice's identity is equivalent to attacking the encryption. In this protocol, each user performs four exponentiations, and the number of interactions is optimal for forward privacy: Alice cannot send her identity during the first interaction because she does not share any secrets with Bob. This would mean that Bob could discover Alice's identity solely using his long-term key, which is impossible if the protocol is forward private. There are therefore at least 3 interactions.

Furthermore, by adding Schnorr's Signature Of Knowledge (SOK) [16] of discrete logarithm on ephemeral keys (signing the user role) and public keys (signing the user identity), we construct a slightly less efficient protocol (three additional exponentiations per user during a session, and two additional exponentiations when verifying the public key certificate to also verify the SOK), but that can be proven under the CDH assumption instead of SDH. SDH allows, during a reduction, to artificially verify via an oracle whether, for a given CDH share $X = g^x$, the triplet (X, A, Z) verifies $Z = A^x$ or not for a chosen (A, Z). Our technique is to attach an extractable SOK to each element $A = g^a$, allowing a to be extracted to verify whether $Z = X^a$ instead.

In Figure 2 we compare the efficiency, assumptions, and security of our protocols with those of [6, 13]. For [13], we only consider the protocol achieving both (strong) MITM and forward privacy, which we instantiate with Diffie-Hellman key exchange, ElGamal encryption [9] and Shnorr signature [16]. We observe that our Π_{SDH} is more efficient in terms of both interactions and exponentiations than the only proven (strongly) private protocol in the literature [13].

1.2 Related work

As mentioned previously, our work is based on the privacy-preserving authenticated key exchange (PPAKE) model proposed by [13], but it should be noted that there are other competing PPAKE models. The earliest attempts to (informally) capture privacy in key agreement can be traced back to Aiello et al. [2], who introduced the idea of keeping a participant's identity hidden from unauthorised parties, including active attackers impersonating peers. Although they proposed two specific protocols, one for protecting the initiator's identity and the other for protecting the responder's, their notion of privacy was neither formally defined nor analysed. Prior to this, Canetti and Krawczyk [4] briefly mentioned

Protocol	**Assump.**	**Privacy**	**Inter.**	**Exp.** (Alice)	**Exp.** (Bob)
Π [6]	SDH (ROM)	×	2	4	4
Π_{Twin} [6]	CDH (ROM)	×	2	8	7
Π_{Com} [6]	DDH (ROM)	×	4	6	6
Π^4_{PKE} [13]	Generic (ROM)	✓	4	5	5
Π_{SDH}	SDH (ROM)	✓	3	4	4
Π_{CDH}	CDH (ROM)	✓	3	7^*	7^*

Fig. 2. Comparison of our work with [6] and [13]. $*$ Π_{CDH} also requires an additional exponentiation when generating long-term keys, and two additional exponentiations when verifying a certificate.

the concept of identity concealment through encryption, albeit without providing a formal framework. Zhao [19] introduced the CAKE model, which formalises forward identity privacy, and limited MITM privacy. However, in this model, the MITM-privacy is only guaranteed if the key exchange is successful, which seems insufficient in practice, because even if the adversary who disrupted the exchange is unable to read the communication, they still learn the identities of the users who wish to communicate. To the best of our knowledge, comparing to prior models [15, 19] only the one proposed by [13] considers MITM attacks on incomplete sessions.

Another line of work involves studying how to add privacy to real-world protocols. Schäge et. al. [15] proposed a tailored PPAKE model and applied it to real-world protocols such as IKEv2. However, their model also assumes prior knowledge of participants' identities and targets specific deployment contexts. On the other hand, the latest versions of TLS have been adapted to ensure certain (weak) privacy properties, which have been analysed in [3]. However, as the general structure of TLS has not been modified, it does not guarantee properties as strong as those of [15].

In a more recent article, Lyu et al. [10] propose another PPAKE model that differs from [13], and provide a protocol proven in the standard model requiring only 3 exchanges. The authors claim that their model is equivalent to the one in [13], and claim an attack on the forward-privacy of the protocol in [13]. However, as added in the 'follow-up work' section of the technical report [14] of [13], the two models differ because [10] assumes that messages are sent on broadcasting channels and that every potential receiver answers to every received message. Furthermore, the attack does not hold in the forward-privacy model of [13], since it considers an active attacker during the session, whereas [13] considers passive attackers only who can corrupt the long-term keys a posteriori. Furthermore, we remark that the protocol in [10] does not achieve the MITM-privacy definition given in [13]. In the MITM-privacy of [13], a user only reveals their identity to another user during a session if they are certain that they are communicating with the right person (formally, the attacker must guess the identity of a user in a session where they either do not yet know the identity of their interlocutor, or they know it and it is an uncorrupted user). In the protocol in [10], the first

flow consists of Alice sending a MAC key to Bob encrypted with Bob's long-term key, then Bob uses this key to generate a MAC on they response. Thus, an adversary can verify whether a user is Bob or not by simply sending the first flow of the protocol, because only Bob will be able to respond, and then interrupt the session. In essence, once again, this paper does not consider MITM-privacy for interrupted sessions.

2 Preliminaries

In this section, we recall some cryptographic tools.

Definition 1 (Computational Diffie-Hellman assumptions). *Let λ be a security parameter, and let $\mathbb{G}$ be a cyclic group of prime order $p > 2^\lambda$ with generator g. Let $x, y \xleftarrow{\$} \mathbb{Z}_p$ be two scalars sampled uniformly at random.*

- *The* **Computational Diffie-Hellman** *(CDH) assumption in $\mathbb{G}$ states that for any Probabilistic Polynomial-Time (PPT) in λ algorithm $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{CDH}}_{\mathcal{A}}(\lambda) = \Pr\left[\mathcal{A}(g^x, g^y) = g^{xy}\right]$ of $\mathcal{A}$ over CDH is negligible.*
- *The* **Strong Diffie-Hellman** *(SDH) assumption in $\mathbb{G}$ is defined as CDH except that $\mathcal{A}$ can access a test oracle $\mathsf{stDH}_x(A, Z)$ that returns 1 if $Z = A^x$, 0 otherwise: it states that for any PPT algorithm $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{SDH}}_{\mathcal{A}}(\lambda) = \Pr\left[\mathcal{A}^{\mathsf{stDH}_x(\cdot,\cdot)}(g^x, g^y) = g^{xy}\right]$ of $\mathcal{A}$ over SDH is negligible. Note that, equivalently, the test oracle can be stDH_y, but the adversary cannot have access to both stDH_x and stDH_y.*

Theorem 7 in [1] proves that SDH holds in Shoup's generic group model. SDH trivially implies CDH.

Definition 2. ***(Authenticated Encryption [12])*** *Let λ be a security parameter. A Symmetric Encryption scheme SE is a triplet $(\mathcal{K}, \mathsf{Enc}, \mathsf{Dec})$, where $\mathcal{K}$ is a secret key set verifying $|\mathcal{K}| > 2^\lambda$, $\mathsf{Enc}_k(m)$ is a an encryption algorithm that takes as input a secret key $k \in \mathcal{K}$ and a message m, and returns a ciphertext c, and $\mathsf{Dec}_k(c)$ is a decryption algorithm that takes as input a secret key $k \in \mathcal{K}$ and a ciphertext c, and returns a message m.*

We also define the authenticated encryption (AE) experiment $\mathsf{Exp}^{\mathsf{AE}}_{\mathsf{SE},b,\mathcal{A}}(\lambda)$ for the two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against SE as follows, where $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$ are encryption/decryption oracles such that $\mathsf{Dec}_k(\cdot)$ returns $\perp$ on c, and returns $\perp$ on any query when $b = 1$:

$\mathsf{Exp}^{\mathsf{AE}}_{\mathsf{SE},b,\mathcal{A}}(\lambda)$:
$k \xleftarrow{\$} \mathcal{K}$; $b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_k(.),\mathsf{Dec}_k(.)}(\lambda)$
$c \leftarrow \mathsf{Enc}_k(m_b)$
returns $b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_k(.),\mathsf{Dec}_k(.)}(c)$

SE *is said to be AE secure if for any any PPT two-stage adversary $\mathcal{A}$, the advantage* $\mathsf{Adv}^{\mathsf{AE}}_{\mathsf{SE},\mathcal{A}}(\lambda) = \left|\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{AE}}_{\mathsf{SE},0,\mathcal{A}}(\lambda) = 1\right] - \mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{AE}}_{\mathsf{SE},1,\mathcal{A}}(\lambda) = 1\right]\right|$ *is negligible.*

For any of these assumptions/properties denoted X, we define the advantage over X as $\mathsf{Adv}^{\mathsf{X}}(\lambda) = \max_{\mathcal{A}\in\mathsf{Poly}(\lambda)} \mathsf{Adv}^{\mathsf{X}}_{\mathcal{A}}(\lambda)$, where $\mathsf{Poly}(\lambda)$ refers to the set of all the probabilistic polynomial-time (possibly two-stage) algorithms in lambda.

For any positive integer x, we write $[x] = \{1, 2, \ldots, x\}$ for the set of the first x positive integers.

3 Security Model

We rely on the authenticated key exchange (AKE) model of [6] for key-secrecy, and the extension of [13] for MITM and forward privacy. Note that in [13], a weak MITM-privacy is also proposed, which assumes that no user is ever corrupted. Since we do not use this weak property, we do not mention it here.

Execution environment. We consider μ parties. Each party P_i is represented by a set of oracles $\{\pi_i^1, ..., \pi_i^\ell\}$, where each oracle corresponds to a session (a single execution of a protocol role), and where $\ell \in \mathbb{N}$ is the maximum number of protocol sessions per party. Each oracle π_i^s has access to the long-term key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ of the party P_i and the list L_{pk} containing the indices/public keys (i, pk_i) of all the parties. It also maintains a list of internal state variables that are described in the following. For each oracle π_i^s, the corresponding variables are initially set to the empty string $\perp$.

- Pid_i^s ("peer id") stores the identity of the intended communication partner. Note that in private key exchange protocols, a user may not know the identity of their partner at the start of a session. In this case, this variable will remain set to $\perp$ until the partner is identified.
- $\Psi_i^s \in \{\perp, \mathsf{Accept}, \mathsf{Reject}\}$ indicates if oracle π_i^s has successfully completed the protocol execution and "accepted" the resulting key, or aborted the protocol and "rejected" the key exchange. Until the protocol is complete or aborted, this value remains $\perp$. As a prerequisite, we stress that Pid_i^s must be instantiated (i.e. $\mathsf{Pid}_i^s \neq \perp$) for Ψ_i^s to be set to Accept.
- k_i^s stores the session key computed by π_i^s. Until the protocol is complete, this value remains $\perp$. It is assigned only if the oracle π_i^s reaches the Accept state. In other words: $k_i^s \neq \perp \Leftrightarrow \Psi_i^s = \text{accept}$.
- $\mathsf{esk}_i^s/\mathsf{epk}_i^s$: stores P_i's ephemeral secret/public keys specified by the protocol. These keys are generated specifically for each session by the parties involved and are renewed for each new session.
- $\overline{\mathsf{epk}}_i^s$: stores the ephemeral public key of the P_i's partner. This key is specific to the session and is received during an interaction specified in the protocol. Note that this key can be received before the partner's identity is learned by the oracle.
- $\mathsf{role}_i^s \in \{\perp, \mathsf{initiator}, \mathsf{responder}\}$ indicates π_i^s's role during the protocol.

Additionally, the environment keeps track of three initially empty lists: L_{corr}, containing all corrupted parties, L_{Send}, storing all sent messages, and L_{SessKey}, recording all established session keys.

Oracles and attacker model. Our attacker $\mathcal{A}$ has a full control over the communication network. Furthermore, the attacker knows the list of parties and their respective public keys. It interacts with the oracles through queries.

- $\mathsf{Send}(i, s, m)$: This allows $\mathcal{A}$ to send an arbitrary message m to oracle π_i^s if it exists. The oracle will respond according to the protocol specification and its current internal state. To start a new oracle, the message m takes a special form ($\mathbf{START} : \mathsf{role}, j$). If π_i^s was already initialized before, Send returns $\perp$. Otherwise, the oracle initializes π_i^s in the role role by setting $\mathsf{role}_i^s = \mathsf{role}$. If $\mathsf{role} = \mathsf{initiator}$, then it sets $\mathsf{Pid}_i^s = j$ and outputs the first message of the protocol. Else, if $\mathsf{role} = \mathsf{responder}$, the partner identity remains $\mathsf{Pid}_i^s = \perp$ (In this case, j is not used and can be set to the empty string $\perp$ in the query). Note that, in the case of the responder, the oracle does not receive the first message of the protocol during initialisation (it will receive it in the second query).
- $\mathsf{RevLTK}(i)$: For $i \leq \mu$, this returns the long-term private key sk_i of party P_i. After this query, P_i and all its oracles π_i^s (for any s) are said to be corrupted.
- $\mathsf{RegisterLTK}(i, \mathsf{pk}_i)$: For $i > \mu$, this allows the adversary to register a new party P_i with the public key pk_i. The adversary is not required to know the corresponding private key. After this query, the pair (i, pk_i) is added to L_{pk}. Parties registered by $\mathsf{RegisterLTK}(i, \mathsf{pk}_i)$ (and their oracles) are corrupted by definition.
- $\mathsf{RevSessKeySec}(i, s)$: This allows the adversary to learn the session key derived by an oracle. If $\Psi_i^s = \mathsf{accept}$, return k_i^s. Otherwise, return $\perp$. After this query, π_i^s is said to be revealed.

Furthermore, privacy properties require an alternative definition of the session key revelation oracle, which uses the definition of *matching conversations.*

Definition 3 (Matching Conversation). *Let Π be an N-message two-party protocol in which all messages are sent sequentially. If a session oracle π_i^s sent (resp. receive) the last message of the protocol, then π_j^t is said to have matching conversations to π_i^s if the first $N-1$ (resp. all N) messages of π_i^s transcript agrees with the first $N-1$ (resp. all) messages of π_j^t's transcript (excluding the initialisation messages received by both oracles).*

- $\mathsf{RevSessKeyPriv}(i, s)$: This is an alternative oracle to learn the session key. If there already is an entry (π_i^s, k^*) in L_{SessKey}, or if there is an entry (π_j^t, k^*) so that π_i^s and π_j^t have matching conversation, this oracle returns k^*. If $\Psi_i^s = \mathsf{accept}$, it sets $k^* \leftarrow k_i^s$, adds (π_i^s, k^*) to L_{SessKey}, and returns k^*, else if $\Psi_i^s = \mathsf{reject}$, it picks a random session key k^*, adds (π_i^s, k^*) to L_{SessKey}, and returns k^*. Otherwise (i.e. If $\Psi_i^s = \perp$), it returns $\perp$. After this query, π_i^s is said to be revealed.

Partnering. Here we recall the definitions used to formally determine whether a session key is corrupted or not, and whether two session oracles correspond to the execution of a protocol between the two corresponding parties.

Definition 4 (Origin-oracle). *An oracle π_j^t is an origin-oracle for an oracle π_i^s if $\Psi_j^t \neq \perp$, $\Psi_i^s = \mathsf{Accept}$ and the messages sent by π_j^t equal the messages received by π_i^s, except for the initialisation message received by π_i^s.*

Definition 5 (Partner oracles). *We say that two oracles π_i^s and π_j^t are partners if (i) each is an origin-oracle for the other; (ii) each one's identity is the other one's peer identity, i.e., $\mathsf{Pid}_i^s = j$ and $\mathsf{Pid}_j^t = i$; and (iii) they do not have the same role, i.e., $\mathsf{role}_i^s \neq \mathsf{role}_j^t$, $\mathsf{role}_i^s \neq \perp$, and $\mathsf{role}_j^t \neq \perp$.*

Security experiment. The security model is formally defined as a game between an adversary, denoted by $\mathcal{A}$, and a challenger, denoted by $\mathcal{C}$. In this game, the adversary may issue the standard set of queries described earlier. Additionally, they are granted access to a special oracle $\mathsf{Test}(m)$ depending on a secret random bit b that the adversary has to guess in order to win. In key-secrecy, the oracle returns either the key for a given session or a random key. For privacy, this oracle initializes a session for one of two parties chosen by the adversary. Since our main focus is privacy, we defer the key-secrecy experiment in Appendix A.

Privacy. The oracle $\mathsf{Test}(m)$ is defined as follows: for $m = (X, i, j)$ such that $X \in \{\mathsf{MITM\text{-}privacy}, \mathsf{Forward\text{-}privacy}\}$ and $i, j \leq \mu$, it creates a new party $P_{i|j}$ with identifier $i|j$. This party has all the properties of P_i (if $b = 0$) or P_j (if $b = 1$), but no active sessions. The public key of $P_{i|j}$ is not announced to the adversary and the query $\mathsf{RevLTK}(i|j)$ always returns $\perp$. Furthermore it creates exactly one session $\pi_{i|j}^1$, and returns the new identifier $i|j$.

The experiment $\mathsf{Exp}^X_{\mathsf{PPAKE},\mathcal{A}}(\lambda)$, with X the label sent to the oracle Test, is defined as follows:

1. Let μ be the number of parties in the game and ℓ the number of sessions per user. $\mathcal{C}$ begins by drawing a random bit $b \xleftarrow{\$} \{0,1\}$ and generating key pairs $\{(\mathsf{sk}_i, \mathsf{pk}_i) \mid 1 \leq i \leq \mu\}$ as well as oracles $\{\pi_i^s \mid 1 \leq i \leq \mu, 1 \leq s \leq \ell\}$.
2. $\mathcal{C}$ now runs $\mathcal{A}$, providing all the public keys as input. During its execution, $\mathcal{A}$ may adaptively issue $\mathsf{Send}(i, s, m)$, $\mathsf{RevLTK}(i)$, $\mathsf{RevSessKeyPriv}(i, s)$ and $\mathsf{RegisterLTK}(i, \mathsf{pk}_i)$ queries any number of times and the $\mathsf{Test}(m)$ query once.
3. Depending on what argument X the $\mathsf{Test}(m)$ oracle was called with, we require the corresponding property below to hold through the entire game.
   (a) $\mathsf{MITM\text{-}privacy}$: P_i and P_j are never corrupted. Furthermore, we require that $\mathsf{Pid}_{i|j}^1 = \perp$ or $\mathsf{Pid}_{i|j}^1 = k$ for some $k \leq \mu$, while P_k is never corrupted.
   (b) $\mathsf{Forward\text{-}privacy}$: The returned oracle $\pi_{i|j}^1$ has a partner oracle π_k^r at the end of the game. Furthermore, no oracle besides π_k^r may be instructed to start a protocol run with intended partner $P_{i|j}$.
4. The game ends when $\mathcal{A}$ terminates with the output b' representing the guess of the secret bit b. If $b' = b$, it outputs 1. Otherwise, it outputs 0.

Definition 6. *Let* PPAKE *be a key exchange protocol. The advantage of an adversary $\mathcal{A}$ in breaking the $X \in \{\mathsf{MITM\text{-}privacy}, \mathsf{Forward\text{-}privacy}\}$ security of protocol* PPAKE *is defined as:*

$$\mathsf{Adv}^X_{\mathsf{PPAKE},\mathcal{A}}(\lambda) = \left| \Pr\left[\mathsf{Exp}^X_{\mathsf{PPAKE},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} \right|.$$

4 Protocol Π_{SDH}

In this section, we prove that the protocol Π_{SDH} in Figure 1, already presented informally in the introduction, has the three properties of our security model under the SDH assumption in the ROM.

In our proofs, during a session simulated by the oracle π_i^s where P_i is the initiator, $(\mathsf{pk}_i, \mathsf{sk}_i)$ corresponds to (U, u), $(\mathsf{pk}_{\mathsf{Pid}_i^s}, \mathsf{sk}_{\mathsf{Pid}_i^s})$ corresponds to (V, v) and $(\mathsf{epk}_i^s, \mathsf{esk}_i^s)$ corresponds to (X, x) in Figure 1. When P_i is the responder, $(\mathsf{pk}_i, \mathsf{sk}_i)$ corresponds to (V, v), $(\mathsf{epk}_i^s, \mathsf{esk}_i^s)$ corresponds to (Y, y), and $(\mathsf{pk}_{\mathsf{Pid}_i^s}, \mathsf{sk}_{\mathsf{Pid}_i^s})$ corresponds to (U, u) when defined. Recall that when the oracle π_i^s is initialised for the role $\mathsf{responder}$, the value of Pid_i^s is set to $\perp$. In accordance with the protocol in Figure 1, Pid_i^s is updated after the third query is received on the condition that the message received is well-formed, i.e., its decryption gives the public key of a registered user P_j. In this case, Pid_i^s will be set to j.

Theorem 1. *Our protocol Π_{SDH} instantiated with a secure AE scheme is MITM-private under the SDH assumption in the ROM.*

We prove MITM-privacy in two steps. First, we show that the adversary can never guess and send the Diffie-Hellman values of V^x and U^y to the RO instead of uncorrupted users (owners of V and U) under the SDH assumption in the random oracle. This result (Lemma 1) allows us to show that during the MITM-privacy experiment, the key K used by $P_{i\|j}$ is perfectly random, and therefore that the identity encrypted in C can only be recovered by attacking the secure authenticated encryption scheme.

Lemma 1. *Let E be the event: "at the end of the* $\mathsf{MITM\text{-}privacy}$ *experiment, for some tuple (s, i, j, R), P_i and P_j are never corrupted, $\mathsf{Pid}_i^s = j$, and the R^{th} query to the RO contains the value $g^{\mathsf{esk}_i^s \mathsf{sk}_j}$, where esk_i^s is the ephemeral key used in π_i^s". For any PPT adversary $\mathcal{A}$ playing the* $\mathsf{MITM\text{-}privacy}$ *experiment on Π_{SDH}, the probability that E occurs and $\mathcal{A}$ wins the experiment is negligible under the* SDH *assumption in the ROM.*

Proof. We show that $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{MITM\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda) = 1 \wedge E\right]$ is negligible using the following sequence of games.

Game G_0: This is the experiment $\mathsf{Exp}^{\mathsf{MITM\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda)$ except that if E does not occur, the experiment returns 0. We have that:

$$\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{MITM\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda) = 1 \wedge E\right] = \mathsf{Pr}\left[G_0 \text{ returns } 1\right].$$

Game G_1: G_1 is the same as G_0 except that the challenger picks $(s_*, i_*, j_*) \xleftarrow{\$} [\ell] \times [\mu]^2$, and returns 0 if E does not occur on the tuple (s_*, i_*, j_*, R_*) for some query index R_*. We have that:

$$\mathsf{Pr}\left[G_0 \text{ returns } 1\right] \leq \ell\mu^2 \mathsf{Pr}\left[G_1 \text{ returns } 1\right].$$

We will show that $\mathsf{Pr}\left[G_1 \text{ returns } 1\right] \leq \mathsf{Adv}^{\mathsf{SDH}}(\lambda)$.

Let $\mathcal{A} \in \mathsf{Poly}(\lambda)$ be an adversary against G_1. We build the following algorithm $\mathcal{B} \in \mathsf{Poly}(\lambda)$ against SDH. $\mathcal{B}$(A,B) perfectly simulates G_1 to $\mathcal{A}$, except that:

- If at some stage $\mathsf{Pid}_{i_*}^{s_*} \neq \perp$ and $\mathsf{Pid}_{i_*}^{s_*} \neq j_*$, $\mathcal{B}$ aborts the game (G_1 will returns 0 anyway).
- $\mathcal{B}$ sets $\mathsf{pk}_{j_*} \leftarrow B$, and sets $\mathsf{epk}_{i_*}^{s_*} \leftarrow A$, where $\mathsf{epk}_{i_*}^{s_*}$ is the ephemeral public key used in $\pi_{i_*}^{s_*}$, noted X in the protocol (resp. Y) for the initiator (resp. responder). Note that $\mathcal{B}$ does not know the respective discrete logarithms sk_{j^*} and $\mathsf{esk}_{i_*}^{s_*}$ of pk_{j^*} and $\mathsf{epk}_{i_*}^{s_*}$. It is therefore necessary to specify how $\mathcal{B}$ simulates the oracles that would use these values, i.e. oracles $\pi_{j_*}^{s}$ for all s for sk_{j^*} and the oracle $\pi_{i_*}^{s_*}$ for $\mathsf{esk}_{i_*}^{s_*}$. These two cases are detailed in the two following items.
- **For simulating the oracles $\pi_{j_*}^{s}$ for all** s, each time that $\mathcal{B}$ should compute $(\overline{\mathsf{epk}}_{j_*}^{s})^{\mathsf{sk}_{j_*}}$, it sets a tuple $T_s \leftarrow (\overline{\mathsf{epk}}_{j_*}^{s}, \mathsf{pk}_{j_*})$. If, afterward, $\mathcal{B}$ needs to compute the hash of an element containing $(\overline{\mathsf{epk}}_{j_*}^{s})^{\mathsf{sk}_{j_*}}$ using the random oracle, it replaces $(\overline{\mathsf{epk}}_{j_*}^{s})^{\mathsf{sk}_{j_*}}$ with T_s and labels the query as *'incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query).
- **For simulating the oracles** $\pi_{i_*}^{s_*}$, we distinguish two cases:

  There exists $(i, s) \neq (i_*, s_*)$ **such that** $\mathsf{epk}_i^s = \overline{\mathsf{epk}}_{i_*}^{s_*}$**:** In this case, the adversary $\mathcal{B}$ knows the secret key esk_i^s. Each time that $\mathcal{B}$ should compute $(\overline{\mathsf{epk}}_{i_*}^{s_*})^{\mathsf{esk}_{i_*}^{s_*}}$, it computes $(\mathsf{epk}_{i_*}^{s_*})^{\mathsf{esk}_i^s}$ instead. Each time that $\mathcal{B}$ should compute $\mathsf{pk}_{j_*}^{\mathsf{esk}_{i_*}^{s_*}}$, it sets the tuple $T \leftarrow (\mathsf{epk}_{i_*}^{s_*}, \mathsf{pk}_{j_*})$. If, afterward, $\mathcal{B}$ needs to compute the hash of an element containing $\mathsf{pk}_{j_*}^{\mathsf{esk}_{i_*}^{s_*}}$ using the random oracle, it replaces $\mathsf{pk}_{j_*}^{\mathsf{esk}_{i_*}^{s_*}}$ by T, and labels the query as *'incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query).

  There is no $(i, s) \neq (i_*, s_*)$ **such that** $\mathsf{epk}_i^s = \overline{\mathsf{epk}}_{i_*}^{s_*}$**:** In this case, $\mathcal{B}$ does not know the discrete logarithm of $\overline{\mathsf{epk}}_{i_*}^{s_*}$. We distinguish two cases:

  - If $\mathsf{role}_{i_*}^{s_*} = \mathsf{initiator}$, each time that $\mathcal{B}$ should compute $(\overline{\mathsf{epk}}_{i_*}^{s_*})^{\mathsf{esk}_{i_*}^{s_*}}$, it sets the result as $\perp$ instead. Each time that $\mathcal{B}$ should compute $\mathsf{pk}_{j_*}^{\mathsf{esk}_{i_*}^{s_*}}$, it sets the tuple $T \leftarrow (\mathsf{epk}_{i_*}^{s_*}, \mathsf{pk}_{j_*})$. If, afterward, $\mathcal{B}$ needs to compute the hash of an element containing both $(\overline{\mathsf{epk}}_{i_*}^{s_*})^{\mathsf{esk}_{i_*}^{s_*}}$ and $\mathsf{pk}_{j_*}^{\mathsf{esk}_{i_*}^{s_*}}$ using the random oracle, it replaces $(\overline{\mathsf{epk}}_{i_*}^{s_*})^{\mathsf{esk}_{i_*}^{s_*}}$ by $\perp$ and $\mathsf{pk}_{j_*}^{\mathsf{esk}_{i_*}^{s_*}}$ by T, and labels the query as *'$\perp$-incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query). For the sake of consistency with the random oracle, if at any time $\mathcal{B}$ generates an ephemeral key epk_i^s such that $\mathsf{epk}_i^s = \overline{\mathsf{epk}}_{i_*}^{s_*}$, then $\perp$ is replaced by $(\mathsf{epk}_{i_*}^{s_*})^{\mathsf{esk}_i^s}$ in the $\perp$-incomplete query, and the label is replaced by *'incomplete'*.

  - Else if $\mathsf{role}_{i_*}^{s_*} = \mathsf{responder}$, $\mathcal{B}$ proceeds as in the previous case, except that it does not compute the key K and does not decrypt the ciphertext C it receives. Instead, it assumes that the decryption of C is pk_{j_*}. Recall that according to game G_1, the adversary must send to $\pi_{i_*}^{s_*}$ a ciphertext correctly formed of the key pk_{j_*} in order to keep a non-negligible advantage of winning. The game is therefore correctly simulated to $\mathcal{A}$ as long as the adversary keeps its advantage.

  We note that in the two cases, $T = (A, B)$, and that there is at most one unique $\perp$-incomplete query.
- Each time $\mathcal{B}$ must process a query to the random oracle that matches the pattern of a previous query labelled *'incomplete'* (i.e. all elements that are not tuples match in the queries), it checks whether each group element Z corresponding to a tuple $T = (\mathsf{epk}, pk_{j_*})$ satisfies $\mathsf{stDH}_b(\mathsf{epk}, Z) = 1$. If so, it returns the value associated with the incomplete query; otherwise, it processes the query as usual. For instance, if $\mathcal{B}$ simulates the first response of P_{j_*} to the first message send by P_i in the session π_i^s, when generating K, it will replace the query $(X^v \| X^y \| X \| Y)$ with the incomplete query $((X, V) \| X^y \| X \| Y)$ to the random oracle H. If, subsequently, $\mathcal{A}$ sends a query $(Z \| X^y \| X \| Y)$, $\mathcal{B}$ will check whether $\mathsf{stDH}_v(X, Z) = 1$, and if so, it will return K (otherwise it will treat $(Z \| X^y \| X \| Y)$ as a standard query to a random oracle).
- Each time $\mathcal{B}$ must process a query to the random oracle that matches the pattern of the query labeled *'$\perp$-incomplete'* (i.e. all elements that are not tuples or $\perp$ match in the queries), it checks whether the group element Z of the query corresponding to the tuple $T = (A, B)$ in the incomplete query satisfies $\mathsf{stDH}_b(A, Z) = 1$. If so, it aborts the experiment and returns Z; otherwise, it processes the query as usual. As a result, it is never necessary to know the value replaced by $\perp$.
- To maintain consistency in responses, when $\mathcal{B}$ stores a new incomplete query, it must also check that no previous query matches this incomplete query in the same way; if so, it must return the corresponding hash of that previous query.
- In order to take into account all cases where the adversary sends a query containing $g^{\mathsf{esk}_{i_*}^{s_*}\mathsf{sk}_{j_*}}$ (possibly by chance) to the random oracle, $\mathcal{B}$ tests, for each element Z contained in each query, whether $\mathsf{stDH}_b(A, Z) = 1$.
- If, at any other point, $\mathcal{B}$ tests $\mathsf{stDH}_b(A, Z) = 1$, it aborts the experiment and returns Z.
- If $\mathcal{B}$ must process the query j_* to RevLTK, $\mathcal{B}$ aborts the game ($\mathcal{A}$ can no longer win G_1 because P_{j_*} must not be corrupted for event E to occur on j_*).

In this way, G_1 is perfectly simulated for $\mathcal{A}$ except for abortions when $\mathcal{A}$ can no longer win the game or when $\mathcal{B}$ is guaranteed to win their SDH experiment, and therefore:

$$\mathsf{Pr}\left[G_1 \text{ returns } 1\right] \leq \mathsf{Adv}^{\mathsf{SDH}}(\lambda).$$

Finally, the following concludes the proof:

$$\mathsf{Pr}\left[\mathsf{Exp}_{\Pi_{\mathsf{SDH}},\mathcal{A}}^{\mathsf{MITM\text{-}privacy}}(\lambda) = 1 \wedge E\right] \leq \ell\mu^2 \mathsf{Adv}^{\mathsf{SDH}}(\lambda).$$

Proof (Theorem 1). We use the following sequence of games.
Game G_0**:** This is the experiment $\mathsf{Exp}^{\mathsf{MITM\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda)$. We have that:

$$\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{MITM\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda) = 1\right] = \mathsf{Pr}\left[G_0 \text{ returns } 1\right].$$

Game G_1**:** Same as G_0 except that at the end of the game, if $\mathcal{A}$ wins according to the conditions of G_0, and if for some tuple (s, i_*, j_*, R), P_{i_*} and P_{j_*} are never corrupted, $\mathsf{Pid}^s_{i_*} = j_*$, and the R^{th} query to the RO contains the value $g^{\mathsf{esk}^s_{i_*}\mathsf{sk}_{j*}}$ (where $\mathsf{esk}^s_{i_*}$ is the ephemeral key used in $\pi^s_{i_*}$), then G_1 sets $\mathsf{abort}_1 = \mathsf{true}$, aborts the game, and returns a random bit. According to Lemma 1, we have:

$$|\mathsf{Pr}\left[G_0 \text{ returns } 1\right] - \mathsf{Pr}\left[G_1 \text{ returns } 1\right]| \leq \mathsf{Pr}\left[\mathsf{abort}_1\right] \leq \ell\mu^2\mathsf{Adv}^{\mathsf{SDH}}(\lambda).$$

Game G_2**:** G_2 is the same as G_1 except that if in two distinct sessions, the ephemeral public keys randomly chosen by the oracles are the same, then G_2 sets $\mathsf{abort}_2 = \mathsf{true}$, aborts and returns a random bit. At most, there is $\ell\mu$ sessions during the game, we deduce that:

$$|\mathsf{Pr}\left[G_1 \text{ returns } 1\right] - \mathsf{Pr}\left[G_2 \text{ returns } 1\right]| \leq \mathsf{Pr}[\mathsf{abort}_2] \leq (\mu\ell)^2/|\mathbb{G}|.$$

Game G_3**:** G_3 is the same as G_2 except that if the RO returns the same random element for two distinct queries, then G_3 sets $\mathsf{abort}_3 = \mathsf{true}$, aborts and returns a random bit. We set q as the number of queries to the random oracle during G_2, we have:

$$|\mathsf{Pr}\left[G_2 \text{ returns } 1\right] - \mathsf{Pr}\left[G_3 \text{ returns } 1\right]| \leq \mathsf{Pr}[\mathsf{abort}_3] \leq q^2/|\mathcal{K}|.$$

From this step, the abort conditions in games 2 and 3 ensure that the key K used in session $\pi^1_{i|j}$ if $P_{i|j}$ is the initiator is used only once for encryption (it can be used for decryption in the other session that matches the first two interactions and therefore uses the same ephemeral keys; this session is unique if it exists according to G_2). Furthermore, according to the condition of G_1, if the adversary submits the query to obtain K to the random oracle, then its winning probability drops to $1/2$. K is therefore perfectly random from their point of view as long as they still have a non-zero advantage in winning the game.

Game G_4**:** G_4 is the same as G_3 except that if the role of $P_{i|j}$ in $\pi^1_{i|j}$ is initiator, then it computes $\mathsf{C} \leftarrow \mathsf{Enc}_{\mathsf{K}}(\perp)$ instead of computing $\mathsf{C} \leftarrow \mathsf{Enc}_{\mathsf{K}}(\mathsf{U})$ at the second interaction. If during the simulation of an oracle the challenger has to decrypt C with the key K, they do not decrypt C and act as if the decryption returned U. Furthermore, if during the simulation of an oracle the challenger has to decrypt a ciphertext $C' \neq C$ with the key K, they act as if the decryption had failed.

We claim that:

$$|\mathsf{Pr}\left[G_3 \text{ returns } 1\right] - \mathsf{Pr}\left[G_4 \text{ returns } 1\right]| \leq \mathsf{Adv}^{\mathsf{AE}}_{\mathsf{SE}}(\lambda).$$

We prove this claim by reduction. Assume that there exists an adversary $\mathcal{A}$ such that $|\mathsf{Pr}\left[G_3 \text{ returns } 1\right] - \mathsf{Pr}\left[G_4 \text{ returns } 1\right]|$ is non-negligible, we build the following AE adversary $\mathcal{B}$. $\mathcal{B}$ simulates the game G_3 to $\mathcal{A}$, except that:

- In $\pi^1_{i|j}$, if $P_{i|j}$ is the initiator, when $\mathcal{B}$ must determine K with the random oracle for the query $(V^x \| Y^x \| X \| Y)$, it associates the query with the label *'undefined'*. Then, instead of computing $C \leftarrow \mathsf{Enc}_K(U)$, it sends $(M_0, M_1) = (U, \perp)$ (where U is the public key of $P_{i|j}$ to its challenger and receives the challenge ciphertext ctxt. It sets $C \leftarrow \mathsf{ctxt}$.
- Each time that $\mathcal{B}$ should compute the hash of query labeled with *'undefined'* to determine the value of a key K, it uses its encryption/decryption oracles $\mathsf{Enc}_K(\cdot)$ and $\mathsf{Dec}_K(\cdot)$ instead of directly using the hash K (If $\mathcal{B}$ must decrypt C, it cannot use the decryption oracle; instead, it considers that the decryption of C is U).
- If at any point the adversary $\mathcal{A}$ requests or has requested a query labeled with *'undefined'* to the random oracle, $\mathcal{B}$ aborts the game and returns a random bit. Note that this behaviour is consistent with the condition introduced in game G_1.
- $\mathcal{B}$ returns 1 if and only if $\mathcal{A}$ wins the game.

In this way, if $b = 0$, G_3 is perfectly simulated at $\mathcal{A}$, else G_4 is perfectly simulated at $\mathcal{A}$, except for abortions when $\mathcal{A}$ has no longer advantage, and therefore, the following concludes the proof of the claim:

$$\mathsf{Pr}\left[G_3 \text{ returns } 1\right] = \mathsf{Pr}\Big[\mathsf{Exp}^{\mathsf{AE}}_{\mathsf{SE},0,\mathcal{B}}(\lambda) = 1\Big];$$

$$\mathsf{Pr}\left[G_4 \text{ returns } 1\right] = \mathsf{Pr}\Big[\mathsf{Exp}^{\mathsf{AE}}_{\mathsf{SE},1,\mathcal{B}}(\lambda) = 1\Big].$$

Game G_5: G_5 is the same as G_4 except that at the end of the game, if the oracle $\mathsf{RevSessKeyPriv}$ generated a random session key k^* equals to either another session key randomly generated by the same oracle or an output of the random oracle, then G_5 sets $\mathsf{abort}_5 = \mathsf{true}$, aborts and returns a random bit. We set q (resp. q_r) as the number of queries to the random oracle (resp. $\mathsf{RevSessKeyPriv}$) during G_4, we have:

$$\left|\mathsf{Pr}\left[G_4 \text{ returns } 1\right] - \mathsf{Pr}\left[G_5 \text{ returns } 1\right]\right| \leq \mathsf{Pr}[\mathsf{abort}_5] \leq q_r(q + q_r)/|\mathcal{K}|.$$

At this stage, we observe that:

- If $\pi^1_{i|j}$ is the initiator, the keys and the ciphertext produced by $\pi^1_{i|j}$ no longer contain any information about the real identity of $P_{i|j}$ according (the keys generated by the RO appear perfectly random to the adversary according to G_1, and the ciphertext contains $\perp$ according to G_4). If it is the responder, no exchanged message depends on the identity of $P_{i|j}$ anymore (according to G_1). It is therefore no longer possible to directly exploit messages exchanged by $\pi^1_{i|j}$ to deduce its identity. Furthermore, if another oracle (for an uncorrupted party) is initialized with $\mathsf{Pid} = i|j$ (i.e., with the initiator role), G_1 guarantees that the encryption and session keys used by this oracle appear to have been generated randomly from the adversary's point of view.

- If $\pi^1_{i|j}$ has a matching conversation, then it is with a unique partner oracle (because the ephemeral keys are all different according to G_2). In this case, according to the definition of the RevSessKeyPriv oracle, the revealing of the session key of the $\pi^1_{i|j}$ and the oracle that has a matching conversation with it will be the same (even if one of the parties reject the session). It will therefore not be possible to use RevSessKeyPriv as a distinguisher to learn information about $\pi^1_{i|j}$ identity. This prevents trivial attacks where the adversary engages an initiator oracle with partner P_i, and forwards messages from that oracle to $\pi^1_{i|j}$ (in this case, $\pi^1_{i|j}$ would accept the session if its identity is P_i, and fail if its identity is P_j).
- In all other cases, regardless of $P_{i|j}$ actual identity, if no oracle π^s_k has a matching conversation with $\pi^1_{i|j}$, then no oracle has the same context as $\pi^1_{i|j}$ and therefore no oracle shares the same session key, because the context is hashed to compute the session key, and all hash values are different according to G_3. It should also be noted that, by definition, even if a session oracle refuses the session after receiving the last message C, RevSessKeyPriv returns a random key that is different from those of the other oracles according to G_5. Thus, the key returned by RevSessKeyPriv on $\pi^1_{i|j}$ will be different from the key returned on any other session oracle. It is therefore not possible to use RevSessKeyPriv to distinguish the identity of $P_{i|j}$ in this case. This prevents attacks where the adversary would try to forge a end-of-session cipher C' containing the key of P_i in order to replace the real cipher C containing the key of $P_{i|j}$, in order to test whether the partner oracle of $\pi^1_{i|j}$ returns the same session key as $\pi^1_{i|j}$ (which would mean that $P_{i|j} = P_i$) or not.

Recall the winning conditions for the MITM-privacy experiment: P_i and P_j are never corrupted, and $\mathsf{Pid}^1_{i|j} = \perp$ or $\mathsf{Pid}^1_{i|j} = k$ while P_k is never corrupted. If $\mathsf{Pid}^1_{i|j} = \perp$, then the session key revealed by RevSessKeyPriv is random by definition. If $\mathsf{Pid}^1_{i|j} = k$ while P_k is never corrupted, then the session key of $\pi^1_{i|j}$ appears perfectly random to $\mathcal{A}$ according to G_1. As long as game G_5 has not been aborted, $\mathcal{A}$ has no further information that can be exploited to guess the identity of $P_{i|j}$, neither by exploiting messages sent/received by $\pi^1_{i|j}$, nor by comparing session keys, so its advantage is zero, and it cannot guess better than guessing at random:

$$\mathsf{Pr}\left[G_5 \text{ returns } 1\right] = 1/2.$$

By considering that $|\mathbb{G}| > 2^\lambda$ and $|\mathcal{K}| > 2^\lambda$, we can therefore conclude that:

$$\left|\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{MITM\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda) = 1\right] - 1/2\right| \leq \ell\mu^2 \mathsf{Adv}^{\mathsf{SDH}}(\lambda) + ((\mu\ell)^2 + q^2 + q_r(q+q_r))/2^\lambda + \mathsf{Adv}^{\mathsf{AE}}_{\mathsf{SE}}(\lambda).$$

Theorem 2. *Our protocol Π_{SDH}, instantiated with a secure AE scheme, is forward-private under the SDH assumption in the ROM.*

The proof of forward-privacy follows the same idea. Without interfering in the communications between two uncorrupted users, the adversary cannot deduce

g^{xy}, and the key K therefore appears to be completely random, perfectly hiding the identity U.

Proof (Theorem 2). We use the following sequence of games.
Game G_0**:** This is the experiment $\mathsf{Exp}^{\mathsf{Forward\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda)$. We have that:

$$\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Forward\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda)=1\right]=\mathsf{Pr}\left[G_0 \text{ returns } 1\right].$$

Game G_1**:** G_1 is the same as G_0 except that the challenger picks $(i_*,j_*,k_*,r_*)\xleftarrow{\$}[\mu]^3\times[\ell]$, and if $i|j\neq i_*|j_*$ or if $\pi^{r_*}_{k_*}$ is not the partner oracle of $\pi^1_{i_*|j_*}$, then G_1 aborts and returns a random bit (note that according to the winning condition of the experiment, the returned oracle $\pi^1_{i|j}$ must have a partner oracle at the end of the game). We have that:

$$\mathsf{Pr}\left[G_0 \text{ returns } 1\right]\leq \ell\mu^3\mathsf{Pr}\left[G_1 \text{ returns } 1\right].$$

Game G_2**:** This hop is identical to that from G_2 to G_3 in the previous proof.

Game G_3**:** G_3 is the same as G_2 except that if the adversary sends a query containing $g^{\mathsf{esk}^1_{i_*|j_*}\mathsf{esk}^{r_*}_{k_*}}$ (where $\mathsf{esk}^1_{i_*|j_*}$ and $\mathsf{esk}^{r_*}_{k_*}$ are the ephemeral keys x and y used in the two sessions) to the random oracle, then G_3 sets $\mathsf{abort}_3=\mathsf{true}$, aborts and returns a random bit. We claim that:

$$|\mathsf{Pr}\left[G_2 \text{ returns } 1\right]-\mathsf{Pr}\left[G_3 \text{ returns } 1\right]|\leq \mathsf{Pr}[\mathsf{abort}_3]\leq \mathsf{Adv}^{\mathsf{SDH}}(\lambda)$$

We prove the claim by reduction. Let $\mathcal{A}\in\mathsf{Poly}(\lambda)$ be an adversary such that $\mathsf{Pr}[\mathsf{abort}_3]$ is non-negligible. We build the following algorithm $\mathcal{B}\in\mathsf{Poly}(\lambda)$ against SDH. $\mathcal{B}(A,B)$ simulates G_2 to $\mathcal{A}$, except that:

- $\mathcal{B}$ sets $\mathsf{epk}^1_{i_*|j_*}\leftarrow A$ and $\mathsf{epk}^{r_*}_{k_*}\leftarrow B$, where $\mathsf{epk}^1_{i_*|j_*}$ and $\mathsf{epk}^{r_*}_{k_*}$ are the ephemeral public keys used in $\pi^1_{i_*|j_*}$ and $\pi^{r_*}_{k_*}$, noted in the protocol X (resp. Y) for the initiator (resp. responder).
- Recall that $\mathcal{A}$ must ensure that $\pi^1_{i_*|j_*}$ and $\pi^{r_*}_{k_*}$ are partners (and so $\mathsf{Pid}^1_{i_*|j_*}=k_*$) in order to maintain a non-negligible advantage according to G_1. As soon as this condition can no longer be satisfied, $\mathcal{B}$ aborts the game by returning a random bit. When it simulates $\pi^1_{i_*|j_*}$, $\mathcal{B}$ computes $(\mathsf{epk}^1_{i_*|j_*})^{\mathsf{sk}_{k_*}}$ instead of $\mathsf{pk}_{k_*}^{\mathsf{esk}^1_{i_*|j_*}}$. Moreover, when $\mathcal{B}$ should compute $(\mathsf{epk}^{r_*}_{k_*})^{\mathsf{esk}^1_{i_*|j_*}}$, it replaces it with $T=(A,B)$. It then labels the queries to the random oracle containing T as *'incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query).
- In a similar way, when it simulates $\pi^{r_*}_{k_*}$, $\mathcal{B}$ computes $(\mathsf{epk}^{r_*}_{k_*})^{\mathsf{sk}_{i_*|j_*}}$ instead of $\mathsf{pk}_{i_*|j_*}^{\mathsf{esk}^{r_*}_{k_*}}$. Moreover, when $\mathcal{B}$ should compute $(\mathsf{epk}^1_{i_*|j_*})^{\mathsf{esk}^{r_*}_{k_*}}$, it replaces it with $T=(A,B)$. It then labels the queries to the random oracle containing T as *'incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query).

- Each time $\mathcal{B}$ must process a query to the random oracle that matches the pattern of a previous query labelled *'incomplete'*, it checks whether the group element Z of the query corresponding to the tuple $T = (A, B)$ in the incomplete query satisfies $\mathsf{stDH}_b(A, Z) = 1$. If so, it aborts the experiment and returns Z, otherwise, it processes the query as usual.
- To maintain consistency in responses, when $\mathcal{B}$ stores a new incomplete query, it must also check that no previous query matches this incomplete query; if so, it must return the corresponding hash of that previous query.

To keep an advantage in the game, the adversary must not interfere with communication between $\pi^1_{i_*|j_*}$ and $\pi^{r_*}_{k_*}$ (they must be exclusive partner oracles), and therefore cannot modify the ephemeral keys exchanged. In this way, G_2 is perfectly simulated at $\mathcal{A}$ except for abortions when $\mathcal{A}$ can no longer win the game or when $\mathcal{B}$ is guaranteed to win their SDH experiment, and therefore $\mathsf{Pr}\left[\mathsf{abort}_3\right] \leq \mathsf{Adv}^{\mathsf{SDH}}(\lambda)$, which concludes the proof of the claim.

From this step, the abort conditions in games 2 and 3 ensure that while the adversary keeps an advantage, the keys generated by the session seem to their fully random, and are never reused in another session (except the key shared with $\pi^{r_*}_{k_*}$). As a result, the latest information that the adversary can exploit about the identity of $P_{i|j}$ is the message encrypted with the key K, which contains its public key (if $P_{i|j}$ plays the role of initiator).

Game G_4 and G_5: These hops are identical to those from G_3 to G_4 and from G_4 to G_5 in the previous proof.

At this stage, we observe that:

- If $\pi^1_{i_*|j_*}$ is the initiator, the keys and the ciphertext produced by $\pi^1_{i_*|j_*}$ no longer contain any information about the real identity of $P_{i_*|j_*}$ (the keys generated by the RO appear perfectly random to the adversary according to G_3, and the ciphertext contains $\perp$ according to G_4). If it is the responder, no exchanged message with its partner oracle depends on the identity of $P_{i_*|j_*}$ anymore. It is therefore no longer possible to directly exploit messages exchanged by $\pi^1_{i_*|j_*}$ to deduce its identity. Furthermore, the winning conditions of the forward-privacy ensure that no oracle besides $\pi^{r_*}_{k_*}$ may be instructed to start a protocol run with intended partner $P_{i_*|j_*}$.
- As in the previous proof, if $\pi^1_{i_*|j_*}$ has a matching conversation, then the revealing of the session key of $\pi^1_{i_*|j_*}$ and the oracle that has a matching conversation with it will be the same. In all other cases, regardless of $P_{i_*|j_*}$ actual identity, if no oracle π^s_k has a matching conversation with $\pi^1_{i_*|j_*}$, then no oracle shares the same session key. It is therefore not possible to use $\mathsf{RevSessKeyPriv}$ to distinguish the identity of $P_{i_*|j_*}$.

Recall the winning conditions for the forward-privacy experiment: the returned oracle $\pi^1_{i|j}$ has a partner oracle π^r_k at the end of the game, and no oracle

besides π_k^r may be instructed to start a protocol run with intended partner $P_{i|j}$. As long as game G_5 has not been aborted, $\mathcal{A}$ has no further information that can be exploited to guess the identity of $P_{i|j}$, neither by exploiting messages exchanged by $\pi_{i|j}^1$ and π_k^r, nor by comparing session keys, so its advantage is zero, and it cannot guess better than guessing at random:

$$\mathsf{Pr}\left[G_5 \text{ returns } 1\right] = 1/2.$$

By considering that $|\mathbb{G}| > 2^\lambda$ and $|\mathcal{K}| > 2^\lambda$, we can therefore conclude that:

$$\left|\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Forward\text{-}privacy}}_{\Pi_{\mathsf{SDH}},\mathcal{A}}(\lambda) = 1\right] - 1/2\right| \leq \ell\mu^3 \left(\mathsf{Adv}^{\mathsf{SDH}}(\lambda) + (q + q^2 + q_r(q + q_r))/2^\lambda + \mathsf{Adv}^{\mathsf{AE}}_{\mathsf{SE}}(\lambda)\right).$$

Theorem 3. *Our protocol Π_{SDH} instantiated with a secure AE scheme is key secret under the SDH assumption in the ROM.*

Recall that a very similar protocol (Π) has already been proven key secret in [6], the only difference being the use of the key $K = \mathsf{H}(g^{xv}\|g^{xy}\|X\|Y)$ to encrypt the public key U (U is not a sensitive value for key-secrecy). Since the key K comes from the RO, it does not provide the adversary with any information about the values g^{xv} and g^{xy}, and is therefore of no use in guessing the session key. Since it essentially follows the same spirit as the proof of the protocol Π in [6], we have deferred it to the Appendix B.

5 Protocol Π_{CDH}

We are now building a variant of our protocol that can be proven under CDH, which is weaker than SDH and is generally considered more standard. This protocol uses signatures of knowledge of a discrete logarithm given a group element and a base. We first recall the formal definition of signature of knowledge.

Definition 7 (Signature Of Knowledge (SOK) [5]). *Let $\mathcal{R}$ be a binary relation and let $\mathcal{L}$ be a language such that $s \in \mathcal{L} \Leftrightarrow (\exists w, (s, w) \in \mathcal{R})$. A* signature of knowledge *for $\mathcal{L}$ is a couple of algorithms $(\mathsf{SOK}, \mathsf{Verify})$ such that the algorithm $\mathsf{SOK}_m\{w : (s, w) \in \mathcal{R}\}$ outputs a signature sok of a message m using w, and the algorithm $\mathsf{Verify}(s, m, \mathsf{sok})$ outputs a bit b such that $\mathsf{Verify}(s, m, \mathsf{SOK}_m\{w : (s, w) \in \mathcal{R}\}) = 1$ for any w such that $(s, w) \in \mathcal{R}$. A SOK must guarantee the following properties:*

- **Extractability.** *There exists a polynomial time knowledge extractor Ext and a negligible function $\mathsf{Adv}^{\mathsf{ext}}_{\mathsf{SOK}}(\lambda)$ such that for any algorithm $\mathcal{A}^{\mathsf{Sim}(\cdot)}(\lambda)$ having access to a simulator that forges signatures for chosen statement and that outputs a fresh tuple (s, m, sok) with $\mathsf{Verify}(s, m, \mathsf{sok}) = 1$, the extractor $\mathsf{Ext}^{\mathcal{A}}(\lambda)$ having access to $\mathcal{A}$ outputs w such that $(s, w) \in \mathcal{R}$ with probability at least $1 - \mathsf{Adv}^{\mathsf{ext}}_{\mathsf{SOK}}(\lambda)$.*

– **(Perfect) Zero-knowledge.** *A signature* sok leaks no information*, i.e., there exists a polynomial time algorithm* Sim *(called the* simulator*) such that* $\mathsf{SOK}_m\{w : (s, w) \in \mathcal{R}\}$ *and* $\mathsf{Sim}(m, s)$ *follow the same probability distribution for any* (m, s) *and any* w *such that* $(s, w) \in \mathcal{R}$*.*

Alice (initiator)		**Bob (responder)**
$\mathsf{sk_i} = u; \mathsf{pk_i} = U = g^u$		$\mathsf{sk_r} = v; \mathsf{pk_r} = V = g^v$
$\mathsf{sok}_i = \mathsf{SOK}_{P_i}\{u : U = g^u\}$		$\mathsf{sok}_j = \mathsf{SOK}_{P_j}\{v : V = g^v\}$
If sok_j is not a valid SOK, Alice aborts the protocol.		
$x \leftarrow \mathbb{Z}_p; \quad X \leftarrow g^x$		
$\mathsf{sok}_X \leftarrow \mathsf{SOK}_0\{x : X = g^x\}$	$\xrightarrow{X, \mathsf{sok}_X}$	$y \leftarrow \mathbb{Z}_p; \quad Y \leftarrow g^y$
		$\mathsf{sok}_Y \leftarrow \mathsf{SOK}_1\{y : Y = g^y\}$
		If sok_X is not a valid SOK, Bob aborts the protocol.
$h_1 \leftarrow (Y^x \| V^x)$	$\xleftarrow{Y, \mathsf{sok}_Y}$	$h_1 \leftarrow (X^y \| X^v)$
$\mathsf{K} \leftarrow \mathsf{H}(X \| Y \| h_1)$		$\mathsf{K} \leftarrow \mathsf{H}(X \| Y \| h_1)$
If sok_Y is not a valid SOK, Alice aborts the protocol.		
$\mathsf{C} \leftarrow \mathsf{Enc}_{\mathsf{K}}(\mathsf{U})$	$\xrightarrow{C}$	$U \leftarrow \mathsf{Dec}_{\mathsf{K}}(\mathsf{C})$
		If U is not a registered public key, or if sok_i is not a valid SOK, Bob aborts the protocol.
$h_2 \leftarrow Y^u$		$h_2 \leftarrow U^y$
$\mathsf{K_i} \leftarrow \mathsf{H}(\mathsf{ctxt} \| h_1 \| h_2)$		$\mathsf{K_r} \leftarrow \mathsf{H}(\mathsf{ctxt} \| h_1 \| h_2)$

Fig. 3. Private authenticated key exchange protocol based on CDH (Π_{CDH}).

The SOK of a discrete logarithm can be instantiated by the Schnorr protocol [17] rendered interactive with the Fiat-Shamir heuristic [8], and where the message is hashed with the rest of the elements. In this case, $\mathsf{SOK}_m\{w : W = g^w\}$ consists of choosing an r and returning $(R = g^r, \alpha = r + xH(g, W, R, m))$ (1 exponentiation), and $\mathsf{Verify}((g, W), m, \mathsf{sok})$ consists of verifying whether $g^\alpha = Ry^{H(g,W,R,m)}$ (2 exponentiations). Note that we implicitly consider in the SOK definition that any hash function will be simulated by a random oracle (RO). In particular, the Schnorr simulator Sim can program the random oracles (Sim picks (α, h), computes $R = y^h/g^\alpha$, and then programs the random oracle such that $h = H(g, W, R, m)$). However, the algorithm $\mathcal{A}$ in extractability only has black-box access to the random oracles, so it cannot program the random oracles and therefore cannot simulate valid proofs on its own on arbitrary statements to extract the witness.

Our Π_{CDH} protocol, shown in Figure 3, is the same as the Π_{SDH} protocol, except that each user uses their long-term key to sign their identity, and uses

their ephemeral key to sign their role (with $\texttt{initiator} = 0$ and $\texttt{responder} = 1$) at each session. Each user must therefore generate a SOK and verify a SOK at each session, which adds 3 exponentiations per user. Note that it is also necessary to verify the SOK of the long-term key, but this can be performed only once (when verifying the certificate of this key) and does not need to be repeated for other sessions with the same partner. The security of this protocol is given in the following theorem.

Theorem 4. *Our protocol Π_{CDH} instantiated with a secure AE scheme and the Schnorr SOK is MITM-private, forward-private and key-secret under the CDH assumption in the ROM.*

The detailed proof of this theorem is provided in the extended version of this work[2]. The proofs of the three properties follow essentially the same structure as those of the previous protocol, except for the CDH reductions. In the previous proofs, during these reductions, the SDH challenge (A, B) was used to instantiate a long-term key and an ephemeral key or two ephemeral keys. Since, before being hashed, these keys are used to compute Diffie-Hellman keys with several different key shares, and since it is sometimes impossible to compute these keys without knowing the discrete logarithm of A or B, we stored the key shares that would have had to be merged in the random oracle queries, then used the stDH oracle to check whether the attacker was sending the corresponding queries. For example, if an adversary's query corresponds to an incomplete query requiring the Diffie-Hellman key of key shares (X, B), we checked whether $\mathsf{stDH}_b(X, Z)$, where Z is the element sent by the adversary. In Π_{CDH}, in similar reductions, while (X, sok_X) is a fresh pair of ephemeral key and SOK (not already outputted by the challenger), we can instead use the knowledge extractor on sok_X to find x and compute $Z = B^x$. However, it should be noted that, since the discrete logarithm of A and B is unknown, the corresponding signatures must be simulated (these signatures are therefore non-extractable). Thus, the adversary could reuse these elements and signatures without knowing the corresponding discrete logarithms. However, since the signature signs the user's role (for ephemeral key) or the user's identity (for long-term key), they can only be reused in the same context. For example, if an oracle sends the ephemeral key A with a signature of the role `initiator`, the adversary cannot send the same ephemeral key back to the oracle, since they are supposed to play the role of the responder. For similar reasons, the adversary cannot register a user using such an ephemeral key as a public key. This prevents cases that cannot be simulated in the reductions.

6 Conclusion and Future Work

In this article, we have given two efficient round-optimal private key exchange protocols and proved their security in the ROM under SDH and CDH. There are many directions in which future research could go. Firstly, our privacy proofs are

[2] See https://hal.science/hal-05593610

not tight, and it would be interesting to improve them and prove the optimal loss reduction that can be achieved for privacy. Secondly, it would be interesting to investigate whether a similar approach could be taken with quantum-resistant primitives, such as in key exchange based on key encapsulation mechanisms. Finally, since our protocols are very similar to Noise, which is used in real-world applications, it would be interesting to see what using our protocols would entail in practice (e.g. for Signal [11], WhatsApp [18] or Wireguard[7]).

Acknowledgments

This work has been supported by the French National Research Agency through the PRIVASIQ project (ANR-23-CE39-0008).

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle diffie-hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer (2001). https://doi.org/10.1007/3-540-45353-9_12, https://doi.org/10.1007/3-540-45353-9_12
2. Aiello, W., Bellovin, S.M., Blaze, M., Canetti, R., Ioannidis, J., Keromytis, A.D., Reingold, O.: Just fast keying: Key agreement in a hostile internet. ACM Trans. Inf. Syst. Secur. **7**(2), 242–273 (2004). https://doi.org/10.1145/996943.996946, https://doi.org/10.1145/996943.996946
3. Arfaoui, G., Bultel, X., Fouque, P., Nedelcu, A., Onete, C.: The privacy of the TLS 1.3 protocol. IACR Cryptol. ePrint Arch. p. 749 (2019), https://eprint.iacr.org/2019/749
4. Canetti, R., Krawczyk, H.: Security analysis of ike's signature-based key-exchange protocol. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 143–161. Springer (2002). https://doi.org/10.1007/3-540-45708-9_10, https://doi.org/10.1007/3-540-45708-9_10
5. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4117, pp. 78–96. Springer (2006). https://doi.org/10.1007/11818175_5, https://doi.org/10.1007/11818175_5
6. Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness - enabling real-world deployments with theoretically sound parameters. IACR Cryptol. ePrint Arch. p. 737 (2019), https://eprint.iacr.org/2019/737
7. Dowling, B., Paterson, K.G.: A cryptographic analysis of the wireguard protocol. IACR Cryptol. ePrint Arch. p. 80 (2018), http://eprint.iacr.org/2018/080
8. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO

'86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986). https://doi.org/10.1007/3-540-47721-7_12, https://doi.org/10.1007/3-540-47721-7_12
9. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074, https://doi.org/10.1109/TIT.1985.1057074
10. Lyu, Y., Liu, S., Han, S., Gu, D.: Privacy-preserving authenticated key exchange in the standard model. In: ASIACRYPT 2022. pp. 210–240. Springer (2022)
11. Marlinspike, M., Perrin, T.: The x3dh key agreement protocol. https://signal.org/docs/specifications/x3dh/ (2016), signal Foundation Specification
12. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the TLS record protocol. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7073, pp. 372–389. Springer (2011). https://doi.org/10.1007/978-3-642-25385-0_20, https://doi.org/10.1007/978-3-642-25385-0_20
13. Ramacher, S., Slamanig, D., Weninger, A.: Privacy-preserving authenticated key exchange: Stronger privacy and generic constructions. In: Bertino, E., Schulmann, H., Waidner, M. (eds.) Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12973, pp. 676–696. Springer (2021). https://doi.org/10.1007/978-3-030-88428-4_33, https://doi.org/10.1007/978-3-030-88428-4_33
14. Ramacher, S., Slamanig, D., Weninger, A.: Privacy-preserving authenticated key exchange: Stronger privacy and generic constructions. IACR Cryptol. ePrint Arch. p. 1338 (2022), https://eprint.iacr.org/2022/1338
15. Schäge, S., Schwenk, J., Lauer, S.: Privacy-preserving authenticated key exchange and the case of ikev2. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 567–596. Springer (2020). https://doi.org/10.1007/978-3-030-45388-6_20, https://doi.org/10.1007/978-3-030-45388-6_20
16. Schnorr, C.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. Lecture Notes in Computer Science, vol. 435, pp. 239–252. Springer (1989). https://doi.org/10.1007/0-387-34805-0_22, https://doi.org/10.1007/0-387-34805-0_22
17. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725, https://doi.org/10.1007/BF00196725
18. WhatsApp Inc., Open Whisper Systems: Whatsapp encryption overview. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (2016), technical Whitepaper
19. Zhao, Y.: Identity-concealed authenticated encryption and key exchange. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi,

S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 1464–1479. ACM (2016). https://doi.org/10.1145/2976749.2978350, https://doi.org/10.1145/2976749.2978350

A Key-Secrecy Experiment

Definition 8 (Freshness). *An oracle π_i^s is fresh if (i) $\Psi_i^s = \mathsf{Accept}$, (ii) $\mathsf{RevSessKey}(i,s)$ has not been issued; (iii) no query $\mathsf{RevSessKey}(j,t)$ has been issued, where π_j^t is a partner of π_i^s; and (iv) $P_{\mathsf{Pid}_i^s}$ was not corrupted before π_i^s accepted if π_i^s has an origin-oracle, and not corrupted at all if π_i^s has no origin-oracle.*

Key secrecy. The oracle $\mathsf{Test}(i,s)$ is defined as follows: if $\Psi_i^s \neq \mathsf{Accept}$, return $\perp$. Else, return k_b where $k_0 = k_i^s$, and $k_1 \xleftarrow{\$} \mathcal{K}$ is a random key. After this query, oracle π_i^s is said to be tested. The experiment $\mathsf{Exp}^{\mathsf{key\text{-}secret}}_{\mathsf{PPAKE},\mathcal{A}}(\lambda)$ is defined as follows:

1. Let μ be the number of parties in the game and ℓ the number of sessions per user. $\mathcal{C}$ begins by drawing a random bit $b \xleftarrow{\$} \{0,1\}$ and generating key pairs $\{(\mathsf{sk}_i, \mathsf{pk}_i) \mid 1 \leq i \leq \mu\}$ as well as oracles $\{\pi_i^s \mid 1 \leq i \leq \mu, 1 \leq s \leq \ell\}$.
2. $\mathcal{C}$ now runs $\mathcal{A}$, providing all the public keys as input. During its execution, $\mathcal{A}$ may adaptively issue $\mathsf{Send}(i,s,m)$, $\mathsf{RevLTK}(i)$, $\mathsf{RevSessKeySec}(i,s)$ and $\mathsf{RegisterLTK}(i,\mathsf{pk}_i)$ queries any number of times and the $\mathsf{Test}(i,s)$ query once.
3. The game ends when $\mathcal{A}$ terminates with the output b' representing the guess of the secret bit b. If the tested oracle is not fresh, it outputs a random bit. Else, if the tested oracle remains fresh and $b' = b$, it outputs 1. Otherwise, it outputs 0.

We define the following three winning events in the game $\mathsf{Exp}^{\mathsf{key\text{-}secret}}_{\mathsf{PPAKE},\mathcal{A}}(\lambda)$:

1. **Event** $\mathsf{breakSound}$**:** This event occurs if there exist two partner oracles π_i^s and π_j^t such that their session keys differ, i.e., $k_i^s \neq k_j^t$. In other words, two partner oracles compute different session keys.
2. **Event** $\mathsf{breakUnique}$**:** This event occurs if there exists an oracle π_i^s that has two distinct partner oracles π_j^t and $\pi_{j'}^{t'}$. That is, one oracle has more than one partner oracle.
3. Let $\mathsf{guessKE}$ be the output of the game $\mathsf{Exp}^{\mathsf{key\text{-}secret}}_{\mathsf{PPAKE},\mathcal{A}}(\lambda)$. We define $\mathsf{breakKE}$ as the event where $\mathsf{guessKE} = 1$.

Definition 9. *Let* PPAKE *be a key exchange protocol. An adversary $\mathcal{A}$ breaks the key-secrecy of* PPAKE *if at least one of the events* $\mathsf{breakSound}$*,* $\mathsf{breakUnique}$*, or* $\mathsf{breakKE}$ *occurs during the game. The advantage of the adversary $\mathcal{A}$ in breaking the AKE security of protocol* PPAKE *is defined as:*

$$\mathsf{Adv}^{\mathsf{AKE}}_{\mathsf{PPAKE},\mathcal{A}}(\lambda) = \max\left\{\Pr[\mathsf{breakSound}], \Pr[\mathsf{breakUnique}], \left|\Pr[\mathsf{breakKE}] - \frac{1}{2}\right|\right\}.$$

B Proof of key-secrecy for Π_{SDH}

Lemma 2. *Let E be the event: "at the end of the* key-secrecy *experiment, for some tuple (s, i, j, R), P_j is never corrupted, $\mathsf{Pid}_i^s = j$, and the R^{th} query to the RO contains the value $g^{\mathsf{esk}_i^s \mathsf{sk}_j}$, where esk_i^s is the ephemeral key used in π_i^s ". For any PPT adversary $\mathcal{A}$ playing the* key-secrecy *experiment on Π_{SDH}, the probability that E occurs and $\mathcal{A}$ wins the expermient is negligible under the* SDH *assumption in the ROM.*

Proof. This lemma can be proved in a manner similar to Lemma 1. The only notable difference is that in this lemma, we do not require that P_i be uncorrupted, which is never used in the proof of Lemma 1 (however, we do need P_j to be uncorrupted because in the reduction to SDH, the public key of P_j is replaced by one of the inptus of the SDH experiment and therefore the corresponding secret key is unknown).

Proof. We show that Π_{SDH} is key secret using the following sequences of games.

Game G_0. In this game, the adversary interacts with honestly behaving oracles according to the key-secrecy security experiment. Oracles derive session keys exactly as specified in the protocol. The adversary's goal is to distinguish a real session key from a random one. We define:

$$\Pr[\mathsf{break}_{\mathsf{KE}}] = \Pr[G_0 \text{ returns } 1].$$

Game G_1. This game is identical to G_0 except that the challenger aborts if at the end of the game, for some tuple (s, i, j, R), P_j is not corrupted, $\mathsf{Pid}_i^s = j$, and the R^{th} query to the RO contains the value $g^{\mathsf{esk}_i^s \mathsf{sk}_j}$, where esk_i^s is the ephemeral key used in π_i^s.

By Lemma 2, we have that:

$$|\Pr[G_0 \text{ returns } 1] - \Pr[G_1 \text{ returns } 1]| \leq \ell\mu^2 \mathsf{Adv}^{\mathsf{SDH}}(\lambda).$$

Game G_2. This game is identical to G_1 except that the challenger aborts the experiment if two oracles in the same role (either initiators or responders) generate identical session contexts. This implies a collision in their ephemeral exponents, which is highly unlikely. Given at most ℓ sessions per user and μ users, this collision occurs with probability at most:

$$|\Pr[G_1 \text{ returns } 1] - \Pr[G_2 \text{ returns } 1]| \leq \frac{(\mu\ell)^2}{|\mathbb{G}|}.$$

Game G_3. This game is identical to G_2 except that the challenger guesses a tuple (i_*, s_*, j_*, t_*) such that if the adversary $\mathcal{A}$ ever queries the Test oracle, it will be on $\pi_{i_*}^{s_*}$ and if that oracle has an origin oracle, it is $\pi_{j_*}^{t_*}$ (Note that t_* is unique, according to G_2). If the adversary $\mathcal{A}$ fails its guess, the game aborts. We have:

$$\Pr[G_2 \text{ returns }] \leq \mu^2\ell^2 \cdot \Pr[G_3 \text{ returns } 1].$$

Game G_4**.** This game is identical to G_3 except that the game sets $\mathsf{abort}_4 = \mathsf{true}$, aborts and returns a random bit if $\pi_{j_*}^{t_*}$ is the origin oracle of $\pi_{i_*}^{s_*}$, $\pi_{i_*}^{s_*}$ is fresh and the adversary has sent a query to the random oracle containing an element equal to $g^{\mathsf{esk}_{i_*}^{s_*}\mathsf{esk}_{j_*}^{t_*}}$ during the game. We claim that :

$$|\Pr[G_3 \text{ returns } 1] - \Pr[G_4 \text{ returns } 1]| \leq \mathsf{Pr}[\mathsf{abort}_4] \leq \mathsf{Adv}^{\mathsf{SDH}}(\lambda).$$

We prove this claim by reduction. Let $\mathcal{A} \in \mathsf{Poly}(\lambda)$ be an adversary such that $\mathsf{Pr}[\mathsf{abort}_4]$ is non-negligible. We build the following algorithm $\mathcal{B} \in \mathsf{Poly}(\lambda)$ against SDH. $\mathcal{B}$(A,B) simulates G_4 to $\mathcal{A}$, except that:

- $\mathcal{B}$ sets $\mathsf{epk}_{j_*}^{t_*} \leftarrow B$, and sets $\mathsf{epk}_{i_*}^{s_*} \leftarrow A$, where $\mathsf{epk}_{i_*}^{s_*}$ and $\mathsf{epk}_{j_*}^{t_*}$ are the ephemeral public keys, respectively, used in $\pi_{i_*}^{s_*}$ and $\pi_{j_*}^{t_*}$. If $\overline{\mathsf{epk}}_{i_*}^{s_*} \neq \mathsf{epk}_{j_*}^{t_*}$, then $\mathcal{B}$ aborts the game (in this case $\pi_{j_*}^{t_*}$ is not the origin oracle of $\pi_{i_*}^{s_*}$). Note that $\mathcal{B}$ does not know the respective discrete logarithms $\mathsf{esk}_{j_*}^{t_*}$ and $\mathsf{esk}_{i_*}^{s_*}$ of $\mathsf{epk}_{j_*}^{t_*}$ and $\mathsf{epk}_{i_*}^{s_*}$. It is therefore necessary to specify how $\mathcal{B}$ simulates the oracles that would use these values, i.e. oracle $\pi_{j_*}^{t_*}$ for $\mathsf{esk}_{j_*}^{t_*}$ and the oracle $\pi_{i_*}^{s_*}$ for $\mathsf{esk}_{i*}^{s_*}$.
- In $\pi_{i_*}^{s_*}$, when $\mathcal{B}$ should compute $(\mathsf{pk}_{\mathsf{Pid}_{i_*}^{s_*}}^{s_*})^{\mathsf{esk}_{i_*}^{s_*}}$, it computes $(\mathsf{epk}_{i_*}^{s_*})^{\mathsf{sk}_{\mathsf{Pid}_{i_*}^{s_*}}^{s_*}}$ instead. In the case where $\mathcal{B}$ does not know the secret key $\mathsf{sk}_{\mathsf{Pid}_{i_*}^{s_*}}^{s_*}$ (i.e. $\mathsf{pk}_{\mathsf{Pid}_{i_*}^{s_*}}^{s_*}$ has been chosen by $\mathcal{A}$), then $\mathcal{B}$ aborts the game. Recall that if $\pi_{i_*}^{s_*}$ is fresh and has an origin oracle, then $P_{\mathsf{Pid}_{i_*}^{s_*}}$ was not corrupted before $\pi_{i_*}^{s_*}$ accepts, so $\mathcal{B}$ knows the secret key of $P_{\mathsf{Pid}_{i_*}^{s_*}}$.
- In $\pi_{i_*}^{s_*}$, when $\mathcal{B}$ should compute $(\mathsf{epk}_{j_*}^{t_*})^{\mathsf{esk}_{i_*}^{s_*}}$, it sets a tuple $T \leftarrow (A, B)$. If, afterward, $\mathcal{B}$ needs to compute the hash of an element containing $(\mathsf{epk}_{j_*}^{t_*})^{\mathsf{esk}_{i_*}^{s_*}}$ using the random oracle, it replaces it with T and labels the query as *'incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query). Each time $\mathcal{B}$ must process a query to the random oracle that matches the pattern of a previous query labelled *'incomplete'* containing T, it checks whether the group element Z of the query corresponding to the tuple $T = (A, B)$ in the incomplete query satisfies $\mathsf{stDH}_b(A, Z) = 1$. If so, it aborts the experiment and returns Z, otherwise, it processes the query as usual.
- In $\pi_{j_*}^{t_*}$, when $\mathcal{B}$ should compute $(A')^{\mathsf{esk}_{j_*}^{t_*}}$ for some A', it sets a tuple $T' \leftarrow (A', B)$. If, afterward, $\mathcal{B}$ needs to compute the hash of an element containing $(A')^{\mathsf{esk}_{j_*}^{t_*}}$ using the random oracle, it replaces it with T' and labels the query as *'incomplete'* when storing it in the list of random oracle queries (but still returns a random value corresponding to the query). Each time $\mathcal{B}$ must process a query to the random oracle that matches the pattern of a previous query labelled *'incomplete'* containing T', it checks whether the group element Z' of the query corresponding to the tuple $T' = (A', B)$ in the incomplete query satisfies $\mathsf{stDH}_b(A', Z') = 1$. If so, it replaces T' by Z' in the stored query (note that a query can contains several tuples T').

– To maintain consistency in responses, when $\mathcal{B}$ stores a new incomplete query, it must also check that no previous query matches this incomplete query in the same way; if so, it must return the corresponding hash of that previous query.

In this way, G_4 is perfectly simulated at $\mathcal{A}$ except for abortions when $\mathcal{A}$ can no longer win the game or when $\mathcal{B}$ is guaranteed to win their SDH experiment, and therefore $\mathsf{Pr}\left[\mathsf{abort}_4\right] \leq \mathsf{Adv}^{\mathsf{SDH}}(\lambda)$,

Game G_5**.** This game is identical to G_4 except that the challenger aborts the game if the RO outputs two times the same value on two different queries, or outputs k_1. We set q as the number of queries to the RO, we have:

$$|\Pr[G_5 \text{ returns } 1] - \Pr[G_4 \text{ returns } 1]| \leq \mathsf{Pr}[\mathsf{abort}_5] \leq (q+1)^2/|\mathcal{K}|.$$

Remember that in order to have a significant probability of winning, the tested oracle ($\pi_{i_*}^{s_*}$ according to G_3) must remain fresh, which means that the session keys of the tested oracle's partner oracles must not have been revealed. Note that given the G_2 game, since session keys are always different, the tested oracle can only have one partner, and given the G_5 game, two oracles that are not partners have a different context and necessarily have different session keys. As a result, $\mathcal{A}$ cannot deduce the tested session key by revealing the session keys of other oracles. Finally, for the adversary to maintain a significant advantage, they must be in one of these two situations required by freshness of the tested session $\pi_{i_*}^{s_*}$:

– $P_{\mathsf{Pid}_{i_*}^{s_*}}$ was not corrupted before $\pi_{i_*}^{s_*}$ accepted if $\pi_{i_*}^{s_*}$ has an origin-oracle. In this case, the game G_4 ensures that the adversary must not have sent the key $(\overline{\mathsf{epk}}_{i_*}^{s_*})^{\mathsf{esk}_{i_*}^{s_*}}$ in a query to the RO, which is necessary to generate the oracle's session key.
– $P_{\mathsf{Pid}_{i_*}^{s_*}}$ was not corrupted at all if $\pi_{i_*}^{s_*}$ has no origin-oracle. In this case, the game G_1 ensures that the adversary must not have sent the key $(\mathsf{pk}_{\mathsf{Pid}_{i_*}^{s_*}})^{\mathsf{esk}_{i_*}^{s_*}}$ in a query to the RO, which is necessary to generate the oracle's session key.

In both cases, the session key of $\pi_{i_*}^{s_*}$ is therefore perfectly random from the point of view of $\mathcal{A}$. It therefore has no way of guessing whether the key returned by the test oracle is the real session key or a randomly chosen key. We deduce that:

$$\Pr[G_5 \text{ returns } 1] = 1/2.$$

Combining all bounds, we obtain:

$$\Pr[\mathsf{break}_{\mathsf{KE}}] \leq (\ell\mu)^2 \left(\frac{1}{|\mathbb{G}|} + 2 \cdot \mathsf{Adv}^{\mathsf{SDH}}(\lambda) + \frac{(q+1)^2}{|\mathcal{K}|} \right).$$

Furthermore, the probability of having two partner oracles at the same oracle is bounded by the probability that two oracles generate the same ephemeral key, and therefore:

$$\Pr[\mathsf{break}_{\mathsf{Unique}}] \leq \frac{(\mu\ell)^2}{|\mathbb{G}|}.$$

Finally, let us consider two partner oracles π_i^s and π_j^t, i.e., which share the same context and transcript, and satisfy $\mathsf{Pid}_i^s = j$ and $\mathsf{Pid}_j^t = i$. We will have:

$$k_i^s = H(\mathsf{ctxt} \| (\mathsf{pk}_j)^{\mathsf{esk}_i^s} \| (\mathsf{epk}_j^t)^{\mathsf{esk}_i^s} \| (\mathsf{epk}_j^t)^{\mathsf{sk}_i}) = H(\mathsf{ctxt} \| g^{\mathsf{sk}_j \mathsf{esk}_i^s} \| g^{\mathsf{esk}_j^t \mathsf{esk}_i^s} \| g^{\mathsf{esk}_j^t \mathsf{sk}_i});$$
$$k_j^t = H(\mathsf{ctxt} \| (\mathsf{epk}_i^s)^{\mathsf{sk}_j} \| (\mathsf{epk}_i^s)^{\mathsf{esk}_j^t} \| (\mathsf{pk}_i)^{\mathsf{esk}_j^t}) = H(\mathsf{ctxt} \| g^{\mathsf{sk}_j \mathsf{esk}_i^s} \| g^{\mathsf{esk}_j^t \mathsf{esk}_i^s} \| g^{\mathsf{esk}_j^t \mathsf{sk}_i}).$$

We deduce that $k_j^t = k_i^s$, then:

$$\Pr[\mathsf{breakSound}] \leq \Pr[k_i^s \neq k_j^t] = 0.$$

To conclude, we obtain the following bound for the AKE advantage:

$$\mathsf{Adv}^{\mathsf{AKE}}_{\mathsf{PPAKE},\mathcal{A}}(\lambda) \leq (\ell\mu)^2 \left(\frac{1}{|\mathbb{G}|} + 2 \cdot \mathsf{Adv}^{\mathsf{SDH}}(\lambda) + \frac{(q+1)^2}{|\mathcal{K}|} \right).$$