XHMQV: Better Efficiency and Stronger Security for Signal's Initial Handshake based on HMQV*

Rune Fiedler[1], Felix Günther[2], Jiaxin Pan[3], and Runzhi Zeng[3]

[1] Technische Universität Darmstadt, Darmstadt, Germany
rune.fiedler@cryptoplexity.de
[2] IBM Research Europe – Zurich, Rüschlikon, Switzerland
mail@felixguenther.info
[3] University of Kassel, Kassel, Germany
{jiaxin.pan,runzhi.zeng}@uni-kassel.de

Abstract. The Signal protocol is the most widely deployed end-to-end-encrypted messaging protocol. Its initial handshake protocol X3DH allows parties to asynchronously derive a shared session key without the need to be online simultaneously, while providing implicit authentication, forward secrecy, and a form of offline deniability. The X3DH protocol has been extensively studied in the cryptographic literature and is acclaimed for its strong "maximum-exposure" security guarantees, hedging against compromises of users' long-term keys and medium-term keys but also the ephemeral randomness used in the handshake. This maximum-exposure security is achieved by deriving keys from the concatenation of 3–4 Diffie–Hellman (DH) secrets, each combining two long-term, medium-term, or ephemeral DH shares.

Remarkably, X3DH's approach of concatenating plain DH combinations is sub-optimal, both in terms of maximum-exposure security and performance. Indeed, Krawczyk's well-known HMQV protocol (Crypto '05) is a high-performance, DH-based key exchange that provides strong security against long-term and ephemeral key compromise. One might hence wonder: why not base Signal's initial handshake on HMQV?

In this work, we study this question and show that a carefully adapted variant of HMQV, which we call XHMQV, indeed enables stronger security and efficiency while matching the constraints of Signal's initial handshake. Most notably, HMQV does not work as a drop-in replacement for X3DH, as the latter's asynchronicity requires the protocol to handle cases where one party runs out of ephemeral keys (pre-uploaded to the Signal server). Our XHMQV design hence augments HMQV with medium-term keys analogous to those used in X3DH. We prove that XHMQV provides security in all 3–4 compromise scenarios where X3DH does and *additionally* in 1–2 further scenarios, strengthening the handshake's maximum-exposure guarantees while using more efficient group operations. We further confirm that our XHMQV design achieves deniability guarantees comparable to X3DH. Our security model is the first

* A preliminary version of this paper appears in the proceedings of the *45th Annual International Cryptology Conference (CRYPTO 2025)*, © IACR 2025. This is the full version.

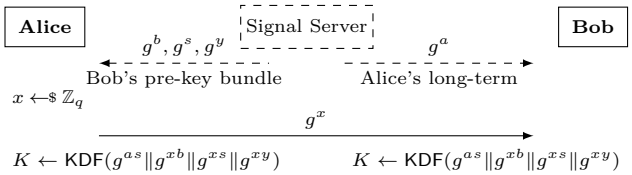

Fig. 1. Signal's X3DH handshake (simplified), involving long-term keys a and b of users Alice resp. Bob, semi-static key s, and ephemeral keys x and y.

to capture Signal's long-term key reuse between DH key exchange and signatures, which may be of independent interest.

1 Introduction

The Signal protocol [49] for end-to-end-encrypted messaging is used by billions of people daily, underlying major messaging apps such as Facebook Messenger, Google Messages, Skype, Signal, or WhatsApp. Its initial handshake protocol X3DH ("extended triple Diffie–Hellman") is used to establish a session key between two users that want to start a conversation. It is designed in a way that the two users do not have to be online simultaneously: users upload a "pre-key bundle" to the Signal server which initiators of a conversation can fetch to asynchronously run their part of the X3DH protocol and start sending messages, while responders can catch up later by completing their part of the handshake. X3DH further provides implicit authentication of the users, forward secrecy in case of future compromise of long-term keys, and a form of deniability allowing users to dispute having been involved in a conversation.

Sub-optimal solution for "maximum-exposure". Perhaps most importantly, X3DH aims at strong security guarantees, sometimes referred to as "maximum-exposure" security: its design hedges against compromises of users' long-term keys and medium-term keys as well as exposure of the ephemeral randomness used in the handshake. This maximum-exposure security is achieved by deriving the session key from the concatenation of 3–4 Diffie–Hellman (DH) secrets g^{as}, g^{xb}, g^{xs}, and (optionally) g^{xy}, each combining two long-term (g^a, g^b for users Alice and Bob), medium-term (g^s for user Bob only), or ephemeral DH shares (g^x, g^y). See Figure 1 for a (simplified) illustration of the X3DH handshake. Signal's handshake and related designs have been studied extensively in the cryptographic literature [11,12,35,53,27,28,19,8,24,6,13,23,29] and Signal is widely recognized as the "gold standard" for secure messaging.

Notably, X3DH uses only a partial coverage of the potential DH combinations, even when considering only the long-term and ephemeral keys. For instance, it does not include the long-/long-term combination g^{ab}, which is useful to hedge against compromise of ephemeral randomness and medium-term secrets. This is a security property that the current Signal handshake cannot achieve. A

Protocol	Group exp.		Key Indistinguishability						Deniability	
	Alice	Bob	LT-SS	E-LT	E-SS	E-E	LT-LT	LT-E	semi-hon.	malic.
X3DH	4 (3)	4 (3)	✓	✓	✓	(✓)	✗	✗	✓	✦
XHMQV	3 (2)	2	✓	✓	✓	(✓)	✓	(✓)	✓	✗

Table 1. Performance, security, and deniability properties of X3DH and XHMQV. Key indistinguishability conditions AA-BB demand security as long as Alice's AA and Bob's BB keys are uncompromised, where LT, SS, E denote long-term, medium-term/semi-static, and ephemeral keys, respectively. Parenthesized entries are for reduced mode (without Bob's ephemeral key present). We discuss the intricacies of malicious deniability below.

straightforward modification is to incorporate this additional long-term/long-term combination, yet simply concatenating plain DH values is *sub-optimal in terms of performance*. On the other hand, the well-known HMQV protocol by Krawczyk [36], building on the MQV protocol [43,39], is a high-performance, DH-based key exchange that provides strong security against long-term and ephemeral key compromise. This naturally raises the question:

Can Signal's initial handshake be (more) efficiently instantiated based on HMQV to provide (even) stronger security?

The short answer is: Yes, but it requires careful adaptation and rigorous proofs.

1.1 Contributions

In this work, we study the above question and introduce XHMQV, a carefully adapted variant of HMQV that enables stronger security and improved efficiency over X3DH, while adhering to the real-world constraints imposed on Signal's initial handshake. We emphasize that HMQV does not work as a drop-in replacement for X3DH, as the latter's asynchronicity requires the protocol to handle cases where one party runs out of ephemeral keys (pre-uploaded to the Signal server). Our XHMQV design hence augments HMQV with semi-static keys analogous to those used in X3DH.

We prove that XHMQV's session keys are secure under all 3–4 compromise scenarios targeted by X3DH and *additional* 1–2 scenarios (see Table 1 for details), and also that it matches Signal's deniability targets (we discuss technical subtleties below). Notably, our security model and proof is the first to capture the re-use of long-term keys in Signal for key exchange and signatures, which may be of independent interest.

In more detail, our contributions are as follows.

XHMQV*: HMQV-based Signal handshake.* We propose XHMQV, a new Signal handshake protocol based on Krawczyk's HMQV protocol [36], illustrated in

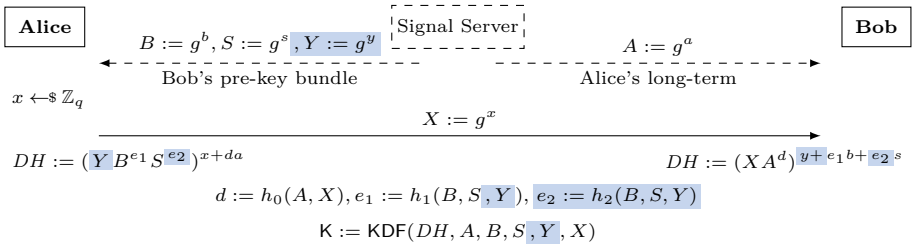

Fig. 2. Our XHMQV handshake protocol (simplified), involving long-term keys a and b of users Alice resp. Bob, semi-static key s, and ephemeral keys x and y, as well as hash functions h_0, h_1, h_2. If Bob's ephemeral key $Y := g^y$ is unavailable, the resulting reduced handshake mode omits elements highlighted in light blue.

Figure 2. The core idea is to augment HMQV with a semi-static key on Bob's side $S = g^s$, similar to the semi-static keys used in Signal's X3DH for delayed forward secrecy in case the Signal server runs out of pre-key bundles for Bob including a fresh ephemeral share $Y = g^y$. (Indeed, removing $S = g^s$ and all related computations from Figure 2 essentially yields HMQV, modulo interaction and hashing context.)

In terms of performance, XHMQV involves 3 or 2 group exponentiations (for Alice and Bob, respectively; in reduced mode only 2 for Alice) whereas X3DH involves 4 group exponentiations per party (in full mode, 3 in reduced mode). While XHMQV additionally involves group multiplications, these are generally much more efficient than exponentiations, resulting in an overall performance advantage.[4]

Stronger "maximal-exposure" security. Compared to X3DH, our XHMQV design ensures security under two additional compromise scenarios, namely compromise of everything except (1) both initiator and responder long-term keys, as well as (2) the initiator long-term key and the responder ephemeral randomness. In both cases, an adversary can trivially compute all DH combinations in the key derivation of X3DH to obtain the session key, whereas our XHMQV protocol inherits protection against such compromise from the original HMQV protocol.

We argue that these strengthened security guarantees are not only theoretical: The former scenario (long-term–long-term uncompromised) hedges against bad randomness used when generating ephemeral and semi-static keys (while long-term keys generated only once may benefit from better randomness sources).

[4] Note that we use multiplicative notation here, in line with prior Signal analyses. Signal's X3DH handshake is using elliptic-curve cryptography, specifically `X25519` based on `Curve25519` [5,38] for key exchange and XEdDSA signatures [46] over `Ed25519` [38]. Notably, X3DH is re-using long-term Diffie–Hellman keys for both DH key exchange and signing a user's pre-key bundle, involving a (cheap) conversion between `Curve25519` and `Ed25519`. When implementing XHMQV with `X25519`, one would likewise convert between `Curve25519` and `Ed25519` to benefit from the latter's fast, constant-time point addition.

The latter scenario (long-term–ephemeral uncompromised) mirrors the protection sought after by X3DH through combining initiator ephemeral and responder long-term keys. We stress that both additional security guarantees essentially come "for free" through employing HMQV's clever key derivation.

Security proof capturing reuse of long-term keys. We prove security for XHMQV in a computational, game-based security model capturing Bellare–Rogaway-style [4] key indistinguishability. To this end, we adopt the security model of Brendel et al. [8,7] and Fiedler and Günther [23], which in turn builds on the initial computational analysis of Signal by Cohn-Gordon et al. [11,12]. In this model, the adversary actively interacts with multiple users and sessions, is allowed to adaptively compromise long-term, semi-static, and ephemeral secrets, and is tasked with distinguishing from random the session key in a chosen *test session* which is not trivially compromised and "*clean*". Cleanliness of a session captures the "maximum-exposure" security aimed at by the XHMQV protocol, asking for security as long as certain combination of keys are uncompromised for the test session. The stronger security achieved by XHMQV compared to X3DH corresponds to our model demanding security in (two) more "clean" cases.

As a notable technical difference to prior models, our security model is the first to capture Signal's key reuse of long-term user keys for both in the `X25519` Diffie–Hellman key exchange and as signing key for XEdDSA [46] signatures. Prior work either modeled these signatures using separate signing keys [35,53,24,6,23,29] or not at all [11,12,8].

As a result of modeling the key reuse, we require a more fine-grained version of signature security. The technical challenge is that, since the long-term key in the DH key exchange is the same as the public key of the signature scheme, we cannot compute the session key when we construct a reduction to signature security, since we do not know the corresponding long-term secret key. To resolve this, we introduce a new notion of security, namely, one-per message unforgeability with a DDH oracle, where $\textsc{Ddh}(X = g^x, Y = g^y, Z)$ outputs whether $Z = g^{xy}$. This is a notion that can be satisfied by signatures using the CDH or DLog assumptions. In Section 2.2 we argue that both the Schnorr and (EC)DSA signature schemes satisfy this notion. As discussed in [22], the (weaker) one-per message unforgeability is sufficient for AKE protocols.

We give more technical details about why this additional DDH oracle can help us to simulate the session key. In our XHMQV protocol, our reduction to the signature security needs to compute $\mathsf{K} := \mathsf{KDF}(DH, A, B, S, X)$ for adversary $\mathcal{A}$ who is attacking XHMQV, where $DH = (XA^d)^{e_1 b + s}$, imagining the simplest reduced handshake mode. Since the long-term keys A and B are coming from the signature challenger, the reduction knows neither a nor b. Our solution is to always answer $\mathsf{KDF}(DH, A, B, S, X)$ with a random value k, although the reduction cannot compute DH. To prevent the adversary from noticing this, KDF is modeled as a random oracle and if at some point $\mathcal{A}$ queries KDF with the correct DH secret $DH = (XA^d)^{e_1 b + s}$, then the reduction will program $\mathsf{KDF}(DH, A, B, S, X)$ to k. To check whether $DH = (XA^d)^{e_1 b + s}$ holds, we need the DDH oracle.

Deniability. Signal aims for deniability of the initial handshake protocol [41, Section 4.4] in the following sense: the transcript of a protocol execution should not prove either party's involvement in the execution, even if the distinguisher gets access to both party's secret keys. In particular, this excludes the case of one party cooperating with the distinguisher during the protocol execution (online deniability), which Unger and Goldberg [51, Section 6.6] deem unachievable for asynchronous key exchange with forward secrecy. Hence, we focus on *offline deniability*. Arguably, Signal's requirement is restricted to semi-honest adversaries, i.e., adversaries that follow the protocol.

Fiedler and Janson [24] show that offline deniability against semi-honest adversaries holds for X3DH. Additionally, they show deniability for Alice against malicious adversaries limited to honestly generated long-term and semi-static keys. Furthermore, Vatandas, Gennaro, Ithurburn, and Krawczyk [53] show that X3DH is deniable against malicious adversaries based on their novel Extended Knowledge of Diffie–Hellman (EKDH) assumption. However, Fiedler and Langrehr [25] show that the EKDH assumption does not hold and that X3DH is not deniable as soon as simple auxiliary input is allowed.

Our protocol XHMQV matches the deniability guarantees of X3DH against semi-honest adversaries. Against malicious adversaries limited to honestly generated long-term and semi-static keys, our XHMQV does not provide deniability for Alice (unlike X3DH). Specifically, if Bob provably does not know his own ephemeral secret key, then the long-term–ephemeral contribution proves Alice's involvement. Arguably, the same deniability issue arises already in X3DH when Bob's semi-static keys are (also) allowed to be maliciously generated, which we deem realistic albeit not modeled in [24].[5] In that regard, XHMQV fares comparably to X3DH in terms of deniability against realistic malicious adversaries. Against malicious adversaries in general, we expect that the findings of Fiedler and Langrehr [25] discussed above likewise apply to XHMQV as to X3DH.

Post-quantum readiness. Our XHMQV design can be readily extended to protect against harvest-now-decrypt-later quantum attacks, following a similar approach as Signal's recent post-quantum revision PQXDH [37] of the initial handshake. Recall that PQXDH essentially adds a (signed) KEM flow to the X3DH design, mixing the derived KEM shared secret into the key derivation. Such a KEM flow and the derived KEM shared secret can be added in an identical fashion to our XHMQV design, lifting the resulting security to a quantum-safe version as for PQXDH. We refer to [6,24,23] for detailed analyses of the design and security of PQXDH.

Deployment considerations. Since our XHMQV protocol only requires a change in how session keys are derived, but does not change the wire format of the protocol,

[5] Even when assuming the Signal server does not collude in a deniability attack (and asks users to prove possession of keys they upload), we expect this is done only for long-term keys, but not semi-static keys. The honest key registration oracle of [24] however enforces that long-term *and semi-static* keys are honestly generated.

we deem XHMQV readily deployable through a next breaking protocol update of Signal. Observe that, in contrast to X3DH, the key derivation in XHMQV also includes as context the involved DH public keys; this is good practice and supports our proof technique. Similar suggestions have been made in the security analyses of Signal's PQXDH protocol [6,23] for a more robust design safeguarding against KEM re-encapsulation attacks [14].

1.2 Related Work and Further Discussion

Analyses of the HMQV protocol. The HMQV protocol was proven secure in [36] based on the Gap Diffie–Hellman assumption using the Forking Lemma [47]. In [34] Kiltz et al. gave a different security proof of HMQV with a modular approach. They first proposed a multi-user variant of the Computational DH problem, the Challenge-Response Gap DH with Corruption (CorrCRGapDH) assumption, and they proved that the CorrCRGapDH assumption tightly implies the security of HMQV. Then they showed that the CorrCRGapDH assumption is non-tightly implied by the Gap DH assumption [1] using the Forking Lemma. Although their proof has the same loss as the original one in [36], its modular approach allows us to focus on the core algebraic assumption for the security of HMQV. In this paper, we follow the same modular approach to simplify our proof.

Tightness of the security proof. Similar to prior work analyzing HMQV [34], X3DH [12], and PQXDH [23], our security bound is not tight, as illustrated in Table 2. However, none of them models the reuse of long-term keys for DH and signing operations. Our analysis of XHMQV is the first one to account for the key reuse. While the result of [34] for HMQV is the only one to allow for multiple Test queries, the protocol does not involve semi-static keys, nor does it take signatures on public keys or key reuse into account. The analysis of X3DH takes the semi-static keys into account, and the analysis of PQXDH additionally considers the signatures on the semi-static keys.

Our result relies on a challenge-response version CRGapDH of the Gap Diffie–Hellman assumption introduced by Kiltz et al. [34] (under the name (2,1)-CorrCRGapDH assumption). Kiltz et al.'s use of the Forking Lemma to reduce CorrCRGapDH to GapDH means that, to base our results on the GapDH assumption, we require rewinding that comes with a square root security loss. However, the security proof for DLog-based signature schemes such as XEdDSA (used in X3DH) and Schnorr signatures is in the ROM and uses rewinding, which leads to a square root loss as well. Therefore, an analysis of X3DH (or any other protocol for Signal's initial handshake) that takes the signatures into account already incurs a square root loss. Hence, the use of the CRGapDH assumption does not incur much penalty for rewinding. We discuss the relation between the GapDH assumption and its challenge-response variant in more detail in Section 2.1.

Analyses of the Signal protocol. The Signal protocol (both the initial handshake X3DH and the Double Ratchet protocol) were first analyzed with a game-based

Protocol	#Test queries	Semi-static keys	Key reuse	Model	Security bound
HMQV (+context) [34]	T	✗	✗	ROM GGM	$\frac{(q_G+n_u+n_s+1)^2}{p} + \frac{2q_{RO}}{p} + \frac{3(n_u+n_s+1)^2(2q_{RO}+1)}{p}$
X3DH [12]	1	✓	✗	ROM	$\frac{(n_u+n_u\cdot n_{ss}+n_s)^2}{p} + n_u^2 \cdot n_s \cdot (4n_{ss}+2+n_s)\left(\frac{1}{p}+\epsilon_{\mathsf{GDH}}\right)$
PQXDH [23]	1	✓✓	✗	ROM	$\frac{(n_u+n_u n_{ss}+n_s)^2}{p} + \gamma_{\mathsf{coll}}(n_u n_{ss}+n_s) + n_s\cdot\delta_{\mathsf{corr}} + \epsilon_{LEAK^{+r}} + \left(n_u^2 n_{ss} + n_s n_u + n_u n_{ss} n_s\right)\cdot\epsilon_{\mathsf{GDH}} + 4n_u\cdot\epsilon_{\mathsf{SIG}} + \left(n_u n_{ss} n_s + n_s^2\right)\cdot\min(\epsilon_{\mathsf{GDH}},\epsilon_{\mathsf{CCA}}) + n_u n_s^2\cdot q_{RO}\cdot\epsilon_{\mathsf{CCA}}$
XHMQV **(this work)**	1	✓✓	✓	ROM	$\frac{(n_u+n_u n_{ss}+n_s)^2}{p} + n_u\cdot\epsilon_{\mathsf{EUF\text{-}opCMA\text{-}DDH}} + \left(2n_u^2+2n_u n_s+n_u^2 n_{ss}\right)\cdot\delta_{sim} + \left(n_u^2+n_u n_{ss}+n_u^2 n_{ss}+n_u n_s + n_s n_u n_{ss}+n_{ss}^2\right)\cdot\left(\epsilon_{\mathsf{CRGapDH}}+\frac{Q_{RO}}{p}\right)$

Table 2. Comparing the tightness of the bounds for HMQV (with additional context as input to the KDF), X3DH, PQXDH, and XHMQV (against classical adversaries). Column 2 denotes the number of allowed Test queries; column 3 if semi-static keys are modeled (✗ not at all, ✓ yes but without being signed, ✓✓ yes and signed); column 4 indicates if the key reuse of the long-term keys for both DH and signing operations is modeled; column 5 states the model; and the final column gives the bound. For HMQV, q_G denotes the number of queries to the group oracle (of the generic group model).

approach by Cohn-Gordon, Cremers, Dowling, Garratt, and Stebila [11,12] and with a tool-based approach by Kobeissi, Bhargavan, and Blanchet [35] in the symbolic and computational model. Signal's initial handshake has since been replaced with PQXDH [37], which extends X3DH with a KEM in order to keep the classical guarantees of X3DH. Bhargavan, Jacomme, Kiefer, and Schmidt [6] have analyzed PQXDH with a tool-based approach in the symbolic and computational model and Fiedler and Günther [23] with a game-based approach. Several post-quantum replacements for X3DH have been proposed: Hashimoto, Katsumata, Kwiatkowski, and Prest [27,28] proposed SC-AKE, SC-DAKE, and SC-DAKE' based on KEMs, ring signatures and NIZKs achieving varying levels of deniability. Brendel, Fiedler, Günther, Janson, and Stebila [8] proposed SPQR, which is based on KEMs and designated verifier signatures. Dobson and Galbraith [19] proposed SI-X3DH based on supersingular isogenies, which was broken by the SIDH attack [10,40,48]. Collins, Huguenin-Dumittan, Nguyen,

Rolin, and Vaudenay [13] proposed K-Waay based on KEMs and splitKEMs. Hashimoto, Katsumata, and Wiggers [29] proposed RingXKEM, which batches computation of pre-key bundles for better efficiency.

Deniability of X3DH was first analyzed by Vatandas, Gennaro, Ithurburn, and Krawczyk [53]. Fiedler and Janson [24] studied the deniability of PQXDH.

Key reuse. Haber and Pinkas [26] and An, Dodis, and Rabin [2] considered reuse of keys between signature schemes and public key encryption schemes and offer definitions for combined security. Degabriele, Lehmann, Paterson, Smart, and Strefler [17] and Paterson, Schuldt, Stam, and Thomson [44] looked at the same setting and define security for one scheme in the presence of an oracle for the other scheme, i.e., IND-CCA in the presence of a signing oracle and EUF-CMA in the presence of a decryption oracle. Patton and Shrimpton [45] generically compose two schemes while taking the context into account. Thormaker [50] analyzed the joint security of an X25519 based KEM and the Ed25519 signature scheme.

On the multi-test security from the multi-instance CorrCRGapDH. Kiltz et al. [34] proved that the security of HMQV can be tightly reduced to the (m, n)-CorrCRGapDH assumption, a multi-instance variant of CRGapDH. Moreover, their AKE security model considers a multi-test setting, namely, the adversary is allowed to query the Test oracle multiple times. This naturally raises the question of whether our XHMQV protocol can be proven secure with a tight reduction under the (m, n)-CorrCRGapDH assumption in the multi-test setting.

We leave it for future study for the following reasons: First, we aim to remain consistent with prior analyses of Signal's handshake protocols [12,23]. Second, we would need to carefully modify our proof such that the underlying (m, n)-CorrCRGapDH assumption does not use an n or m depending on T, the number of Test queries. Otherwise, T would appear in the security loss to the Gap DH assumption, since the reduction from (m, n)-CorrCRGapDH to Gap DH loses m^2n (cf. Theorem 1 and Figure 5 in [34]).

2 Preliminaries

We assume that all our algorithms are efficiently computable and probabilistic unless we state it.

2.1 Diffie–Hellman Assumptions

We use a weaker variant of the Challenge-Response Gap Diffie-Hellman with Corruption (CorrCRGapDH) assumption of Kiltz et al. [34] to prove our XHMQV protocol secure. In [34] Kiltz et al. gave a direct security proof of HMQV from CorrCRGapDH, and by using the Forking Lemma [47] they proved that the Gap DH (GapDH) assumption [1] implies the CorrCRGapDH assumption. Hence, by using the CorrCRGapDH assumption, we do not require the direct use of the

$\underline{\mathcal{G}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{A}, Q_{\mathrm{DDH}}, Q_{\mathrm{CH}}):}$

1 $a, b \leftarrow\!\!\$\ \mathbb{Z}_p,\ A := g^a, B := g^b$
2 $C \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{DDH},\mathrm{CH}}((\mathbb{G}, p, g), A, B)$
3 **return** $[\![C \in \mathcal{L}_{\mathsf{CRGapDH}}]\!]$

$\underline{\mathcal{G}^{\mathsf{GapDH}}_{(\mathbb{G},p,g)}(\mathcal{A}, Q_{\mathrm{DDH}}):}$

1 $a, b \leftarrow\!\!\$\ \mathbb{Z}_p,\ A := g^a, B := g^b$
2 $C \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{DDH}}((\mathbb{G}, p, g), A, B)$
3 **return** $[\![C = A^b]\!]$

$\underline{\mathrm{DDH}(X, Y, Z):}$

// at most Q_{DDH} queries
4 **return** $[\![Z = X^{\mathsf{DL}_g(Y)}]\!]$

$\underline{\mathrm{CH}(R \in \mathbb{G}):}$

// at most Q_{CH} queries
5 $h \leftarrow\!\!\$\ \mathbb{Z}_p$
6 $\mathcal{L}_{\mathsf{CRGapDH}} := \mathcal{L}_{\mathsf{CRGapDH}} \cup \{(R \cdot A^h)^b, (R \cdot B^h)^a\}$
7 **return** h

Fig. 3. Security game for the CRGapDH problem defined in Definition 1.

Forking Lemma in our security proof for XHMQV, simplifying the presentation. By [34, Theorem 1], the security of XHMQV is based on the GapDH assumption. The original CorrCRGapDH assumption is in the multi-user, multi-challenge setting, and ours is in the single-pair-of-user, single-challenge setting, and thus our assumption does not have any corruption query and is denoted by CRGapDH, which is $(2, 1)$-CorrCRGapDH in the notation of [34].

Let $\mathbb{G}$ be a cyclic group with prime order p and generator g. We write $(\mathbb{G}, p, g)$ as the group description of $\mathbb{G}$.

Definition 1 (Challenge-Response Gap Diffie–Hellman Problem). *We say the* $(t, \epsilon, Q_{\mathrm{DDH}}, Q_{\mathrm{CH}})$-CRGapDH *assumption holds in* $(\mathbb{G}, p, g)$ *if for any adversary* $\mathcal{A}$ *with running time at most* t *making at most* Q_{DDH} *queries to its* DDH *oracle and* Q_{CH} *queries to its* CH *oracle, we have that*

$$\mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{A}, Q_{\mathrm{DDH}}, Q_{\mathrm{CH}}) := \Pr\left[\mathcal{G}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{A}, Q_{\mathrm{DDH}}, Q_{\mathrm{CH}})\right] \leq \epsilon,$$

where $\mathcal{G}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{A}, Q_{\mathrm{DDH}}, Q_{\mathrm{CH}})$ *is defined in Fig. 3. For simplicity, when the numbers of oracle queries are clear, we may ignore* t, Q_{DDH}*, and* Q_{CH} *so that we can focus on security bounds. We may then write the game as* $\mathcal{G}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{A})$ *and the advantage function as* $\mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{A})$.

Lemma 1 (GapDH $\xrightarrow{\textbf{rewind}}$ CRGapDH [34, Lemma 6]). *For any adversary* $\mathcal{A}$ *that breaks the* $(t, \epsilon, Q_{\mathrm{DDH}}, Q_{\mathrm{CH}})$-CRGapDH *assumption, there exists an adversary* $\mathcal{B}$ *that breaks the* $(t', \epsilon', Q'_{\mathrm{DDH}})$-GapDH *assumption, where* $t' \approx 2t, \epsilon' \approx Q_{\mathrm{CH}}\epsilon^2$*, and* $Q'_{\mathrm{DDH}} \approx 2Q_{\mathrm{DDH}}$.

2.2 Signature

In Signal's X3DH and PQXDH protocols, user's semi-static keys are signed under their long-term keys, and the signatures included in the pre-key bundles. These signatures ensure that no one can upload a malicious semi-static key unless the user's long-term key is compromised.

We recall that a signature scheme $\mathsf{Sig} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ consists of the following algorithms:

$\mathcal{G}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}[(\mathbb{G},p,g)]}(\mathcal{A}, Q_{\text{SIGN}}, Q_{\text{DDH}})$:

1 $(\mathsf{vk} = g^{\mathsf{sk}} \in \mathbb{G}, \mathsf{sk} \in \mathbb{Z}_p) \leftarrow\!\!\$\ \mathsf{KGen}$
2 $(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\text{SIGN},\text{DDH}}((\mathbb{G}, p, g), \mathsf{vk})$
3 **if** $(m^*, \sigma^*) \notin \mathcal{L}_{\mathtt{sign}}$
$\quad \wedge\ \mathsf{Ver}(\mathsf{vk}, \sigma^*, m^*) = 1$
4 $\quad$ **return** 1
5 **else return** 0

SIGN(m):

// at most Q_{SIGN} queries
6 **if** $\exists\sigma$ s.t. $(m, \sigma) \in \mathcal{L}_{\mathtt{sign}}$
7 $\quad$ **return** σ
8 $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)$
9 $\quad$ **return** σ
10 $\mathcal{L}_{\mathtt{sign}} := \mathcal{L}_{\mathtt{sign}} \cup \{(m, \sigma)\}$
11 **return** σ

DDH(X, Y, Z):

// at most Q_{DDH} queries
12 **return** $[\![Z = X^{\mathsf{DL}_g(Y)}]\!]$

Fig. 4. Game EUF-opCMA-DDH for a signature scheme Sig.

- KGen generates a public and secret key pair $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{sk}, m)$ generates a signature σ of message m under secret key sk.
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma)$ deterministically outputs 1 if σ is a valid signature of m under vk or 0 otherwise.

We require a signature scheme to be correct, namely, for all $(\mathsf{vk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}$ and all messages m, we have $\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$.

Existential UnForgeability against Chosen-Message Attacks (EUF-CMA) is a standard security notion for signature schemes and is also used in proving security of Signal's PQXDH handshake when modeling signatures in the previous computational analysis of [23]. Here, we require only a weaker form of EUF-CMA, namely, one-per message unforgeability [22] where an adversary is restricted to ask at most one signature per message. This is because, in proving XHMQV (similar to the computational proofs for X3DH [11] and PQXDH [23]), we rule out collisions in honestly generated keys early on, meaning that each semi-static public key will only be singed once by a user. Since Signal's initial handshake protocol uses the same long-term key both for the DH key exchange and as secret key in its XEdDSA signature scheme, we consider the following customized one-per message security notion, which augments the regular notion with a decisional Diffie–Hellman oracle DDH.

Definition 2 (One-per message unforgeability with a DDH oracle). *Let* Sig *be a signature scheme based on a group description* $(\mathbb{G}, p, g)$*. We say* Sig *is* $(t, \epsilon, Q_{\text{SIGN}}, Q_{\text{DDH}})$-EUF-opCMA-DDH*-secure if for any adversary* $\mathcal{F}$ *with running time at most* t *making at most* Q_{SIGN} *queries to its* SIGN *oracle and* Q_{DDH} *queries to its* DDH *oracle, we have*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}[(\mathbb{G},p,g)]}(\mathcal{A}, Q_{\text{SIGN}}, Q_{\text{DDH}}) := \Pr\left[\mathcal{G}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}[(\mathbb{G},p,g)]}(\mathcal{A}, Q_{\text{SIGN}}, Q_{\text{DDH}})\right] \leq \epsilon,$$

where $\mathcal{G}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}[(\mathbb{G},p,g)]}(\mathcal{A}, Q_{\text{SIGN}}, Q_{\text{DDH}})$ *is defined in Figure 4. For simplicity, when the group and the numbers of oracle queries are clear, we may write the game as* $\mathcal{G}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}}(\mathcal{A})$ *and the advantage funcion as* $\mathsf{Adv}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}}(\mathcal{A})$.

The Schnorr signature scheme can be proven EUF-opCMA-DDH secure by using a variant of the Discrete Logarithm assumption, where an attacker has to solve a discrete logarithm when given a DDH oracle. This additional DDH oracle

$\underline{\mathcal{G}^{Real}_{\mathsf{Sig},Q}(\mathcal{A})}$:

1 $(\mathsf{vk},\mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Sig.KGen}$, $\mathcal{L}_{\mathsf{RO}} := \emptyset$
2 $(m_1,...,m_Q) \leftarrow\!\!\$\ \mathcal{A}()$
3 **for** $i \in \{1,...,Q\}$
4 $\quad \sigma_i \leftarrow\!\!\$\ \mathsf{Sign}^{\mathsf{RO}}(\mathsf{sk}, m_i)$
5 $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{RO}}(\mathsf{vk},\mathsf{sk},\{\sigma_i\}_{1\le i\le Q})$
6 **return** b_{guess}

$\underline{\mathcal{G}^{Sim}_{\mathsf{Sig},Q}(\mathcal{A})}$:

1 $(\mathsf{vk},\mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Sig.KGen}$, $\mathcal{L}_{\mathsf{RO}} := \emptyset$
2 $(m_1,...,m_Q) \leftarrow\!\!\$\ \mathcal{A}()$
3 **for** $i \in \{1,...,Q\}$
4 $\quad (\sigma_i, qry_i, h_i) \leftarrow\!\!\$\ \mathsf{SIMsign}(\mathsf{vk}, m_i)$
5 $\quad \mathcal{L}_{\mathsf{RO}} := \mathcal{L}_{\mathsf{RO}} \cup \{(qry_i, h_i)\}$
6 $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{RO}}(\mathsf{vk},\mathsf{sk},\{\sigma_i\}_{1\le i\le Q})$
7 **return** b_{guess}

$\underline{\mathsf{RO}(qry)}$ // In $\mathcal{G}^{Sim}_{\mathsf{Sig}}$:

8 **if** $\exists h$ s.t. $(qry,h) \in \mathcal{L}_{\mathsf{RO}}$
9 $\quad$ **return** h
10 $h \leftarrow\!\!\$\ \mathsf{RO.image}$
11 $\mathcal{L}_{\mathsf{RO}} := \mathcal{L}_{\mathsf{RO}} \cup \{(qry,h)\}$
12 **return** h

Fig. 5. Security game for the δ-simulatability in Definition 3. Here, $\mathcal{A}$ is stateful.

$\underline{\mathsf{DSA.Sign}^{\mathsf{RO}}(\mathsf{sk}=x,m)}$:

1 $r \leftarrow\!\!\$\ \mathbb{Z}_p$
2 $R := g^r$
3 $t := f(R)$
4 $h := \mathsf{RO}(m)$
5 $s := r^{-1}\cdot(h + xt)$
6 **return** (s,t)

$\underline{\mathsf{SIMsign}(\mathsf{vk}=X,m)}$:

1 $(a,b) \leftarrow\!\!\$\ \mathbb{Z}_p^2$
2 $R := X^a \cdot g^b$
3 $t := f(R)$
4 $h := b\cdot t\cdot a^{-1}$
5 $s := t\cdot a^{-1}$
6 **return** $((s,t),m,h)$ // Here qry is m.

$\underline{\mathsf{Schnorr.Sign}^{\mathsf{RO}}(\mathsf{sk}=x,m)}$:

1 $r \leftarrow\!\!\$\ \mathbb{Z}_p, R := g^r$
2 $h := \mathsf{RO}(R,m)$
3 $s := (r + x\cdot h) \in \mathbb{Z}_p$
4 **return** (s,R)

$\underline{\mathsf{SIMsign}(\mathsf{vk}=X,m)}$:

1 $h \leftarrow\!\!\$\ \mathbb{Z}_p$
2 $s \leftarrow\!\!\$\ \mathbb{Z}_p$
3 $R := g^s \cdot (X^{-1})^h$
4 **return** $((s,R),(R,m),h)$ // Here qry is (R,m).

Fig. 6. Signing (Sign) and simulated signing (SIMsign) algorithms for DSA and Schnorr signatures. For both schemes, the secret key is a random $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and public key is $X := g^x$.

is to simulate the DDH oracle in Fig. 4 and the remainder of the proof reduces forging a signature to computing a discrete logarithm as in [47,33]. This variant of the Discrete Log assumption is implied by the standard Gap DH assumption. We can show that the (EC)DSA scheme is also EUF-opCMA-DDH secure by following the proof in [22] and introducing a DDH oracle in the underlying assumption.

To stay as close as possible to the original Signal X3DH handshake, which uses long-term keys for both DH key exchange and signing, our XHMQV protocol does the same. We thus require the following δ-simulatability notion, adapted from [22, Definition 5]. Essentially, δ-simulatability states that there is an algorithm that can efficiently generate signatures which are statistically close to real signatures, when allowed programming the random oracle.

Definition 3 (δ-Simulatability, adapted from [22, Definition 5]). *Let* Sig *be a signature scheme whose signing algorithm uses a function* RO *modeled as a random oracle. Consider the two games defined in Figure 5. We say* Sig *is δ-simulatable if there exists an algorithm* SIMsign *such that for any (unbounded)*

adversary $\mathcal{A}$ *and* $Q \in \mathbb{N}$*, we have*

$$\left|\Pr\left[\mathcal{G}^{Real}_{\mathsf{Sig},Q}(\mathcal{A}) = 1\right] - \Pr\left[\mathcal{G}^{Sim}_{\mathsf{Sig},Q}(\mathcal{A}) = 1\right]\right| \leq \delta.$$

Both the DSA and the schnorr signature schemes satisfy the δ-simulatability definiton with negligible δ. Fig. 6 presents the SIMsign algorithms for the DSA and Schnorr signature schemes. The one for DSA is from [22, Figure 5]. The idea here is that the construction of h in the SIMsign of DSA ensures that h is distributed uniformly at random unless a is not invertible (which happens with probability $\frac{1}{p}$). By a union bound, we have $\delta \leq \frac{Q}{p}$. For the full proof of the simulatability of DSA scheme, we refer to [22, Lemma 1]. For the Schnorr scheme, we have $\delta = 0$.

3 Security Model

We analyze the security of our XHMQV protocol with a security model for authenticated key exchange, following the seminal Bellare–Rogaway (BR)–style key exchange model [4], adapted to the setting of Signal's initial handshake in a series of prior work [11,12,8,23]. Our analysis is game-based and yields a computational guarantee.

BR-style key exchange models allow an adversary to interact with multiple honest users with multiple sessions each. It can fully control the network and deliver, modify, inject and suppress messages (through a Send oracle). Furthermore, the adversary can (partially) compromise users: It can corrupt long-term keys (via the CorruptLTKey oracle), corrupt semi-static keys (via the CorruptSSKey oracle), reveal the ephemeral randomness of a session (via the RevealRand oracle), and reveal the session key of a session (via the RevealSessKey oracle). This fine-grained control over user compromise allows us to model the maximum exposure guarantees that Signal strives for. Lastly, the adversary picks a *test session* (via a Test oracle) for which it needs to distinguish between the real session key and a randomly sampled session key. Our model precludes the adversary from being able to trivially compute the session key in the test session. Formally, we do this via a "freshness" condition, which includes checking for protocol specific compromise scenarios with the so-called clean predicate. We give more details on this below.

Our model is directly based on the revision by [8,23], adapted mainly to capture the additional exposure scenarios that our XHMQV design protects against. We hence closely follow the exposition of [23]. Based on this line of work, we consider Bob, who first uploads his pre-key bundle, as responder, and Alice, who thereafter sends the first peer-specific message, as initiator.

3.1 Syntax and Notation

We follow the syntax of [23], who explicitly model the signatures on semi-static keys.[6]

[6] As [23] analyzed PQXDH, their model includes two types of semi-static keys (DH and KEM keys), while ours focuses on DH keys only.

- $\mathsf{KGenLT}() \xrightarrow{\$} (ltpk, ltsk)$: The *long-term key generation* algorithm that outputs a party's public-key/secret-key pair.
- $\mathsf{KGenSS}(ltsk) \xrightarrow{\$} (sspk, sssk)$: The *semi-static key generation* algorithm that takes as input a long-term secret key $ltsk$ and outputs a public-key/secret-key pair.
- $\mathsf{Run}(ltsk, \boldsymbol{sssk}, \boldsymbol{ltpk}, \pi, m) \xrightarrow{\$} (\pi', m')$: The *session execution algorithm* that takes as input a party's long-term secret key $ltsk$, that party's semi-static secret keys $\boldsymbol{sssk}$, all parties' long-term public keys $\boldsymbol{ltpk}$, a session state π, and an incoming message m, and outputs an updated session state π' and a (possibly empty) outgoing message m'. The session sending the first message is set up by calling Run with a distinguished message $m = (\mathsf{create}, (\mathsf{ssid}, \mathsf{type}))$, where ssid indicates the semi-static key to be used and type whether a full ($\mathsf{type} = \mathsf{full}$) or reduced ($\mathsf{type} = \mathsf{reduced}$) handshake should be performed.

Parties and sessions. In the model, each *party* $P \in [n_p]$ holds a long-term public-key/secret-key pair generated by KGenLT and may run multiple instances of the protocol (simultaneously or sequentially); we denote the ith such *session* of party P by π_P^i. The security game maintains the following information for each session:

- $\mathsf{oid} \in [n_p]$: The identity of the session owner.
- $\mathsf{pid} \in [n_p] \cup \{\star\}$: The identity of the intended peer, which may initially be unknown (indicated by $\star$).
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: The role of the party.
- $\mathsf{st}_{\mathsf{exec}} \in \{\bot, \mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$: The status of this session's execution.
- $\mathsf{sid} \in \{0,1\}^* \cup \{\bot\}$: A session identifier defining partnering.
- $\mathsf{cid} \in \{0,1\}^* \cup \{\bot\}$: A contributive identifier, defining a preliminary form of partnering (often as a substring or prefix of the session identifier) for the case the session is not yet bound to an authenticated peer [21].
- $\mathsf{K} \in \mathcal{K}_{\mathsf{KE}} \cup \{\bot\}$: The session key established in this session, initialized to $\bot$.
- $\mathsf{type} \in \{\mathsf{full}, \mathsf{reduced}\}$: Indicator whether an ephemeral pre-key was used ($\mathsf{type} = \mathsf{full}$) for key establishment, or not ($\mathsf{type} = \mathsf{reduced}$).
- $\mathsf{coins} \in \mathcal{R}_{\mathsf{KE}}$: The random coins from the randomness space $\mathcal{R}_{\mathsf{KE}}$ used in the execution of Run; set by the game and read-only thereafter.
- $\mathsf{sspk} \in \{0,1\}^* \cup \{\bot\}$: The semi-static public key used in this session.
- $\mathsf{state} \in \{0,1\}^*$: Potential session state of the protocol.

For bookkeeping in the security game we additionally introduce the following flags, which are not accessible by the protocol sessions:

- $\mathsf{revrand} \in \{\mathsf{true}, \mathsf{false}\}$ indicates whether the random coins $\pi.\mathsf{coins}$ have been revealed via a RevealRand query. The default value is false.
- $\mathsf{pcorr} \in \{\mathsf{true}, \mathsf{false}\}$ indicates whether the peer's long-term key was corrupted at the point in time when this session accepted. The default value is false.

Session partnering. We say that two sessions π_U^i, π_V^j are *partnered* if they agree on the session identifier, i.e., if $\pi_U^i.\mathsf{sid} = \pi_V^j.\mathsf{sid} \neq \bot$. *Contributive* identifiers indicate that two sessions may eventually derive the same session key but are not fully partnered (yet).

3.2 Security Game

We formalize security as *key indistinguishability* through game $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A})$ in Figure 7. The game first fixes a bit $b_{\mathsf{test}} \leftarrow\!\!\$\ \{0,1\}$. Next, it samples long-term and semi-static keys for all users (one long-term key and n_{ss} for each of the n_p users). Then, the adversary $\mathcal{A}$ gets all public keys $\boldsymbol{ltpk}, \boldsymbol{sspk}$ and can access several oracles to control interaction with and between honest users.

- $\textsc{Send}(U, i, m)$: Sends message m to session π^i_U, which corresponds to executing $\mathsf{Run}(ltsk_U, \boldsymbol{sssk}_U, \boldsymbol{ltpk}, \pi^i_U, m)$, saving the updated session state π' as π^i_U, and returning the outgoing message m' to the adversary.
- $\textsc{CorruptLTKey}(U)$: Returns party U's long-term secret key $ltsk_U$ to the adversary; recorded through the flag $\mathsf{corrltk}_U$.
- $\textsc{CorruptSSKey}(U, \mathsf{ssid})$: Returns party U's semi-static secret key $sssk_U^{\mathsf{ssid}}$ to the adversary; recorded through the flag $\mathsf{corrssk}_{sspk_U^{\mathsf{ssid}}}$.[7]
- $\textsc{RevealRand}(U, i)$: Returns the randomness $\pi^i_U.\mathsf{coins}$ of session π^i_U to the adversary, recorded in the session through the flag $\mathsf{revrand}$.
- $\textsc{RevealSessKey}(U, i)$: Returns the session key $\pi^i_U.\mathsf{K}$ of session π^i_U to the adversary, recorded in the session through the flag $\mathsf{revealed}$.
- $\textsc{Test}(U, i)$: If a Test query has been made before or session π^i_U has not accepted, then return $\perp$. Otherwise; if $b_{\mathsf{test}} = 0$, return $\pi^i_U.\mathsf{K}$, otherwise return a randomly sampled session key from the protocol's key space $\mathcal{K}_{\mathsf{KE}}$. Record the test session as $\pi^* \leftarrow \pi^i_U$.

Finally, the adversary outputs a bit b_{guess}. If the adversary was able to violate soundness of the protocol, the game outputs 1. If the adversary can trivially compute the session key of the test session, we say that it violates *freshness* and the game outputs a random bit. Otherwise, the game returns whether the adversary's guess b_{guess} correctly decided the test bit b_{test}, i.e., if the adversary can distinguish between the real session key and a randomly sampled session key. The advantage of the adversary is its chance of winning minus its chance of randomly guessing correctly ($\frac{1}{2}$).

Definition 4 (Key Indistinguishability [23]). *Let* KE *be a key exchange protocol and* $\mathcal{A}$ *an adversary against the key indistinguishability (*KI*) game* $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A})$ *in Figure 7. We say that* KE *achieves* $(t, \epsilon, (q_{\mathrm{Snd}}, q_{\mathrm{CorrLT}}, q_{\mathrm{CorrSS}}, q_{\mathrm{RevR}}, q_{\mathrm{RevSK}}))$–key indistinguishability*, if for any adversary* $\mathcal{A}$ *against* $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A})$ *with running time at most* t *and making at most* q_{Snd}*,* q_{CorrLT}*,* q_{CorrSS}*,* q_{RevR}*, resp.* q_{RevSK} *queries to its* Send*,* CorruptLTKey*,* CorruptSSKey*,* RevealRand*, resp.* RevealSessKey *oracles, we have that* $\mathsf{Adv}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A}) = \Pr\left[\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A})\right] - \frac{1}{2} \leq \epsilon$*. Note that the model restricts the adversary to a single query to the* Test *oracle.*

[7] Following [23], we base the $\mathsf{corrssk}$ flag on the semi-static public key and not on the semi-static id ssid, since initiator sessions do not learn the ssid.

$\underline{\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A})}$:

1 $b_{\mathsf{test}} \leftarrow\!\!\$\ \{0,1\}$ // sample challenge bit
2 $\pi^* \leftarrow \bot$ // variable for test session
3 **for** $U \in [n_p]$ // generate long-term keys
4 $\quad (ltpk_U, ltsk_U) \leftarrow\!\!\$\ \mathsf{KGenLT}()$
5 $\quad$ **for** $s \in [n_{ss}]$ // generate semi-static keys
6 $\quad\quad (\boldsymbol{sspk}_U[s], \boldsymbol{sssk}_U[s]) \leftarrow\!\!\$\ \mathsf{KGenSS}(ltsk_U)$
7 $\boldsymbol{ltpk} \leftarrow \{ltpk_U\}_{U\in[n_p]}$
8 $\boldsymbol{sspk} \leftarrow \{\boldsymbol{sspk}_U\}_{U\in[n_p]}$
9 $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ // run adversary
10 **if** sound() = false: **return** true
// adversary wins if it breaks soundness
11 **if** fresh(π^*) = false: $b_{\mathsf{guess}} \leftarrow 0$
// attack invalid if test session is not fresh
12 **return** $[\![b_{\mathsf{guess}} = b_{\mathsf{test}}]\!]$ // determine win or loss

fresh(π^*):

13 **if** π^*.revealed = true: **return** false
// test session is revealed
14 **if** $\exists \pi^j_V \neq \pi^*$: (π^j_V.sid = π^*.sid $\wedge$ π^j_V.revealed = true): **return** false
// test session's partner is revealed
15 **return** $\mathsf{clean}^{\mathsf{KE}}_{\pi^*.\mathsf{type}}(\pi^*)$
// test session is clean wrt. to its handshake type (full resp. reduced)

sound():

16 **return** $\forall$ distinct π, π', π'' : (
17 $\quad (\pi.\mathsf{sid} = \pi'.\mathsf{sid} \neq \bot \implies \pi.\mathsf{K} = \pi'.\mathsf{K} \wedge \pi.\mathsf{type} = \pi'.\mathsf{type} \wedge \pi.\mathsf{cid} = \pi'.\mathsf{cid})$
// same session identifiers imply same key, type, contributive identifiers
18 $\quad$ **and** $(\pi.\mathsf{sid} = \pi'.\mathsf{sid} \neq \bot \wedge \pi.\mathsf{role} = \mathsf{initiator} \implies \pi'.\mathsf{role} = \mathsf{responder})$
// session identifiers of two initiator sessions never collide
19 $\quad$ **and** $(\pi.\mathsf{sid} = \pi'.\mathsf{sid} = \pi''.\mathsf{sid} \neq \bot \implies \pi.\mathsf{type} = \mathsf{reduced})$)
// session identifiers of three sessions only collide in reduced mode

---

Send(U, i, m):

20 **if** $\pi^i_U = \bot$:
// initiate session: for responders, we have m = (create, (ssid, type))
21 $\quad \pi^i_U.\mathsf{oid} \leftarrow U$ // set owner identity
22 $\quad$ **if** $m = (\mathsf{create}, \dots)$: $\pi^i_U.\mathsf{role} \leftarrow \mathsf{responder}$
// set responder role
23 $\quad$ **else** $\pi^i_U.\mathsf{role} \leftarrow \mathsf{initiator}$
// set initiator role (m is first message)
24 $\quad \pi^i_U.\mathsf{coins} \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{KE}}$ // sample session randomness
25 $\quad \pi^i_U.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
26 $(\pi^i_U, m') \leftarrow \mathsf{Run}(ltsk_U, \boldsymbol{sssk}_U, \boldsymbol{ltpk}, \pi^i_U, m)$
// run session, random coins in π^i_U
27 **if** $\pi^i_U.\mathsf{st}_{\mathsf{exec}} = \mathsf{accepted}$:
// flag if peer was corrupted upon acceptance
28 $\quad \pi^i_U.\mathsf{pcorr} \leftarrow \mathsf{corrltk}_{\pi^i_U.\mathsf{pid}}$
29 **return** $(m', \pi^i_U.\mathsf{st}_{\mathsf{exec}})$ // return message and session state

Test(U, i):

30 **if** $\pi^i_U = \bot$ **or** $\pi^i_U.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{accepted}$ **or** $\pi^* \neq \bot$: **return** $\bot$
// session does not exist, has not accepted yet, or test already asked
31 $\pi^* \leftarrow \pi^i_U$ // record test session
32 $\mathsf{K}_0 \leftarrow \pi^i_U.\mathsf{K}$
33 $\mathsf{K}_1 \leftarrow\!\!\$\ \mathcal{K}_{\mathsf{KE}}$
34 **return** $\mathsf{K}_{b_{\mathsf{test}}}$ // return real-or-random challenge key

CorruptLTKey(U):

35 $\mathsf{corrltk}_U \leftarrow \mathsf{true}$ // mark long-term key corrupted
36 **return** $ltsk_U$ // return long-term secret key

CorruptSSKey(U, ssid):

37 $\mathsf{corrssk}_{sspk^{\mathsf{ssid}}_U} \leftarrow \mathsf{true}$ // mark semi-static key corrupted
38 **return** $\boldsymbol{sssk}_U[\mathsf{ssid}]$ // return semi-static secret key

RevealRand(U, i):

39 **if** $\pi^i_U = \bot$: **return** $\bot$ // session does not exist
40 $\pi^i_U.\mathsf{revrand} \leftarrow \mathsf{true}$ // mark randomness revealed
41 **return** $\pi^i_U.\mathsf{coins}$ // return session's random coins

RevealSessKey(U, i):

42 **if** $\pi^i_U = \bot$ **or** $\pi^i_U.\mathsf{st}_{\mathsf{exec}} \neq \mathsf{accepted}$: **return** $\bot$
// session does not exist or has not yet derived session key
43 $\pi^i_U.\mathsf{revealed} \leftarrow \mathsf{true}$ // mark session key revealed
44 **return** $\pi^i_U.\mathsf{K}$ // return session key

Fig. 7. Key indistinguishability (KI) game for key exchange protocol KE (top), in which adversary $\mathcal{A}$ has access to oracles Send, Test, CorruptLTKey, CorruptSSKey, RevealRand, and RevealSessKey (bottom), and wrt. to (protocol-specific) clean predicates $\mathsf{clean}^{\mathsf{KE}}$ for KE. The clean predicates satisfied by our XHMQV protocol are given in Figure 8.

Soundness. The model captures soundness, i.e., that session identifiers correctly reflect protocol executions, via the sound predicate. Specifically, it excludes the following three cases:

Fig. 8. Clean predicates for the XHMQV protocol. Predicates satisfied by XHMQV but not X3DH are highlighted.

(i) Two sessions accept with the same session identifier, but derive different session keys, indicate different handshake types (full vs. reduced), or do not agree on their contributive identifiers (Fig. 7, line 17).
(ii) Two initiator sessions accept with the same session identifier (Fig. 7, line 18).
(iii) Three sessions accept with the same session identifier in full handshake type (Fig. 7, line 19).

Freshness. The security model excludes cases which allow the adversary to win trivially: If the adversary has revealed the test session (line 13), if it has revealed a session partnered to the test session (line 14), or if it can compute the session key with the keys it has compromised via queries to the CorruptLTKey, CorruptSSKey, RevealRand oracles (line 15). The clean predicates are protocol-specific since different protocols require different (combinations of) keys to compute the session key.

3.3 Clean Predicates for XHMQV

In our XHMQV protocol, key derivation combines both of Alice's keys (long-term and ephemeral) with each of Bob's keys (long-term, semi-static, and ephemeral), yielding six combinations in a full handshake. If Bob's ephemeral key is unavailable (because the Signal server ran out of pre-key bundles for Bob), the number of combinations accordingly reduce to four. As long as one of these six (or four) combinations is not known to the adversary, it should not be able to compute the session key.

Formally, we model this with the $\mathsf{clean}^{\mathsf{XHMQV}}$ predicates given in Figure 8, more specifically $\mathsf{clean}^{\mathsf{XHMQV}}_{\mathsf{full}}$ and $\mathsf{clean}^{\mathsf{XHMQV}}_{\mathsf{reduced}}$ capturing the expected security of a full resp. reduced handshake. These predicates ensure that at least one of the combinations long-term–semi-static ($\mathsf{clean}_{\mathsf{LTSS}}$), ephemeral–long-term ($\mathsf{clean}_{\mathsf{ELT}}$), ephemeral–semi-static ($\mathsf{clean}_{\mathsf{ESS}}$), or long-term–long-term ($\mathsf{clean}_{\mathsf{LTLT}}$) is uncompromised. For a full handshake, the predicate is also satisfied if the ephemeral–ephemeral ($\mathsf{clean}_{\mathsf{EE}}$) or long-term–ephemeral ($\mathsf{clean}_{\mathsf{LTE}}$) combination is uncompromised.

To check that an ephemeral contribution is not compromised, we use the $\mathsf{clean}_{\mathsf{peerE}}$ sub-predicate: Here, we check that for an initiator session the randomness of a contributively partnered (i.e., via cid) responder session is not revealed or for a responder session the randomness of a partnered (via cid) initiator session is nor revealed.

For cases involving semi-static keys, i.e., $\mathsf{clean}_{\mathsf{LTSS}}$ and $\mathsf{clean}_{\mathsf{ESS}}$, simply requiring the semi-static key to be uncompromised is not sufficient: We additionally check that the responder's long-term key is uncompromised at the time when the initiator accepts via the pcorr flag or that the semi-static key was honestly generated. The first additional check was introduced by [23] to model that the responder's semi-static key is authenticated via a signature under the responder's long-term key. The second additional check is a novel strengthening to capture that even if the responder's long-term key is compromised, we expect security as long as the semi-static key is not compromised.

Compared to X3DH, the clean predicates of our XHMQV protocol give a strictly stronger guarantee: For all cases in which X3DH is clean, our XHMQV protocol is clean as well. Furthermore, if only the long-term–long-term or only the long-term–ephemeral combination is clean, then our clean predicate for XHMQV demands security, but X3DH does not satisfy this. This is the stronger maximum-exposure guarantee of our XHMQV protocol.

4 Our XHMQV Protocol

We now introduce our XHMQV protocol, using a group $(\mathbb{G}, p, g)$ and a signature scheme Sig (like in Signal, Sig may build on $\mathbb{G}$). Further, denoting by $\mathcal{K}$ the session key space, we use two key derivation functions $\mathsf{KDF}_f, \mathsf{KDF}_r \colon \{0,1\}^* \to \mathcal{K}$ for deriving session keys, and four hash functions $h_0, h_1, h_2, h_3 \colon \{0,1\}^* \to \mathbb{Z}_p$. (The indices indicate that in practice, we expect those to be instantiated using the same algorithm using domain separation.)

We fully define XHMQV in Fig. 9, along with a variant XHMQV$^+$. In XHMQV, long-term keys are used for both the DH key exchange key and for signing semi-static keys (as is done in X3DH). XHMQV$^+$ instead use independent signing keys, ensuring proper key separation. We will focus on and prove security for XHMQV, i.e., *with* key reuse, given this is closest to Signal' deployment. We emphasize that our analysis capturing this key reuse is in contrast to prior work, which either modeled the signatures using separate signing keys [35,53,24,6,23,29] or not at all [11,12,8]. Our results naturally apply to XHMQV$^+$, under weaker assumptions.

Adapting HMQV. To understand the design of XHMQV, we start by introducing how we adapt the idea from HMQV to the Signal setting. Let (A, a) and (X, x) be the long-term (identity) key pair and ephemeral key pair of Alice, and let (B, b), (S, s), and (Y, y) be the long-term, semi-static, and ephemeral key pairs of Bob, respectively. Following X3DH, a handshake session is either in the full mode or in the reduced mode.

Recall that in HMQV only involves two key pairs (long-term, ephemeral) on each side. In a full handshake, we hence need to integrate the additional semi-static key of the responder (S, s)) into the combination of DH secrets. Our approach is to view (S, s) as another "long-term secret" of Bob (in the HMQV sense), and integrate it into DH combinations in the following way (from Alice's view):

$$(YB^{e_1})^{x+da} /\!/ \mathsf{HMQV} \quad \longrightarrow \quad (YB^{e_1}S^{e_2})^{x+da} /\!/ \mathsf{XHMQV},$$

where d, e_1, e_2 are exponents derived using hash functions, following HMQV's design.

In a reduced handshake, (Y, y) is unavailable and the semi-static key pair (S, s), which is regularly rotated, serves to provide (delayed) forward secrecy. Here, we can essentially fall back to the HMQV equation, dropping (Y, y) and e_2, and using the semi-static key in place of the responder's ephemeral key (again, from Alice's view):

$$(YB^{e_1})^{x+da} /\!/ \mathsf{HMQV} \quad \longrightarrow \quad (B^{e_1}S)^{x+da} /\!/ \mathsf{XHMQV},$$

Let us discuss how to derive d, e_1, and e_2. These three exponents are chosen to prevent a linear relationship between Y, B, and S. We keep $d := h_0(A, X)$ as in HMQV and let $e_1 := h_1(B, Y, S)$ and $e_2 := h_2(B, Y, S)$. That is, both e_1 and e_2 include the long-term, semi-static, and ephemeral information of Bob. This choice is not only for excluding trivial attacks (e.g., if S is not involved in e_2, then the adversary can contruct a malicious Y to eliminate S), but also essential for our security proof, namely, allows us to construct reductions from CRGapDH assumption (cf. Definition 1).

Similar to X3DH, we sign semi-static keys in XHMQV using the long-term key, to protect against active attackers replacing them. While adding signature upgrades the protocol from weak forward secrecy security to full forward secrecy, the key reuse issue requires the signature scheme to have stronger security

than the standard unforgeability definition (cf. Definition 2). For completeness, we also provide a version that use independent signing key, XHMQV$^+$, see the highlighted differences in Fig. 9.

Security of XHMQV. The security theorem for XHMQV is given in Theorem 1 below. One can use it to derive a similar result for XHMQV$^+$ (except that we would require Sig to achieve EUF-opCMA instead of EUF-opCMA-DDH).

Theorem 1. *The* XHMQV *protocol given in Fig. 9 achieves* $(t, \epsilon, (q_{\text{Snd}}, q_{\text{CorrLT}}, q_{\text{CorrSS}}, q_{\text{RevR}}, q_{\text{RevSK}}))$–key indistinguishability *defined in Definition 4.*

Specifically, let n_u, n_{ss}, *and* n_s *be the numbers of users, semi-static keys per user, and sessions in the key exchange game* $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$, *respectively. Let* $(\mathbb{G}, p, g)$ *be a group and both* XHMQV *and the signature scheme* Sig *of* XHMQV *be built on this group. If the* $(t_0, \epsilon_{\mathsf{CRGapDH}}, Q_{\text{DDH}}, Q_{\text{CH}})$-CRGapDH *assumption holds in* $\mathbb{G}$ *and* Sig *has* $(t_1, \epsilon_{\mathsf{EUF\text{-}opCMA\text{-}DDH}}, Q_{\text{SIGN}}, Q'_{\text{DDH}})$-EUF-opCMA-DDH *security and* δ_{sim}*-simulatability, then for any adversary* $\mathcal{A}$ *with running time* t*, we have*

$$\begin{aligned}\epsilon \leq\ & \frac{(n_u + n_u n_{ss} + n_s)^2}{p} + n_u \cdot \epsilon_{\mathsf{EUF\text{-}opCMA\text{-}DDH}} \\ & + \left(2n_u^2 + 2n_u n_s + n_u^2 n_{ss}\right) \cdot \delta_{sim} \\ & + \Big(n_u^2 + n_u n_{ss} + n_u^2 n_{ss} \\ & \qquad + n_u n_s + n_s n_u n_{ss} + n_{ss}^2\Big) \cdot \left(\epsilon_{\mathsf{CRGapDH}} + \frac{Q_{RO}}{p}\right),\end{aligned}$$

where Q_{RO} *is the number of random oracle queries issued by* $\mathcal{A}$ *across* h_0, h_1, h_2, h_3, KDF_f, *and* KDF_r, *and* $t \approx t_0 \approx t_1$, $Q_{\text{CH}} \approx Q_{RO}$, $Q_{\text{DDH}} \approx Q_{RO}$, $Q'_{\text{DDH}} \approx Q_{RO}$, *and* $Q_{\text{SIGN}} \approx n_{ss}$. *Without loss of generality, we assume that* $\mathcal{A}$ *does not issue repeated queries since they return the same response, so we also have* $q_{\text{Snd}} \leq 2n_s, q_{\text{CorrLT}} \leq n_u, q_{\text{CorrSS}} \leq n_u \cdot n_{ss}, q_{\text{RevR}} \leq n_s$, *and* $q_{\text{RevSK}} \leq n_s$.

Remark 1 (Use the same KDF or hash function, with domain separation). For the key derivation functions KDF_f and KDF_r, as well as the hash functions h_0, h_1, h_2, and h_3, the indices indicate that in practice, we expect those to be instantiated using the same algorithm using domain separation [3]. This can be achieved, e.g., via prefixing labels, similar to domain separation already done in Signal [46, Section 2.5].

Remark 2 (On the "Another Look at HMQV"). Menezes [42] identified several vulnerabilities in HQMV, including small-subgroup and unknown key-share attacks, and proposed corresponding countermeasures. These considerations are also reflected in our XHMQV design. For instance, XHMQV includes the session context in key derivation process to prevent unknown key-share attacks. In real-world implementations, XHMQV must additionally validate all group elements, ensuring the correct order or group membership of long-term public keys, semi-static public keys, and ephemeral pre-keys.

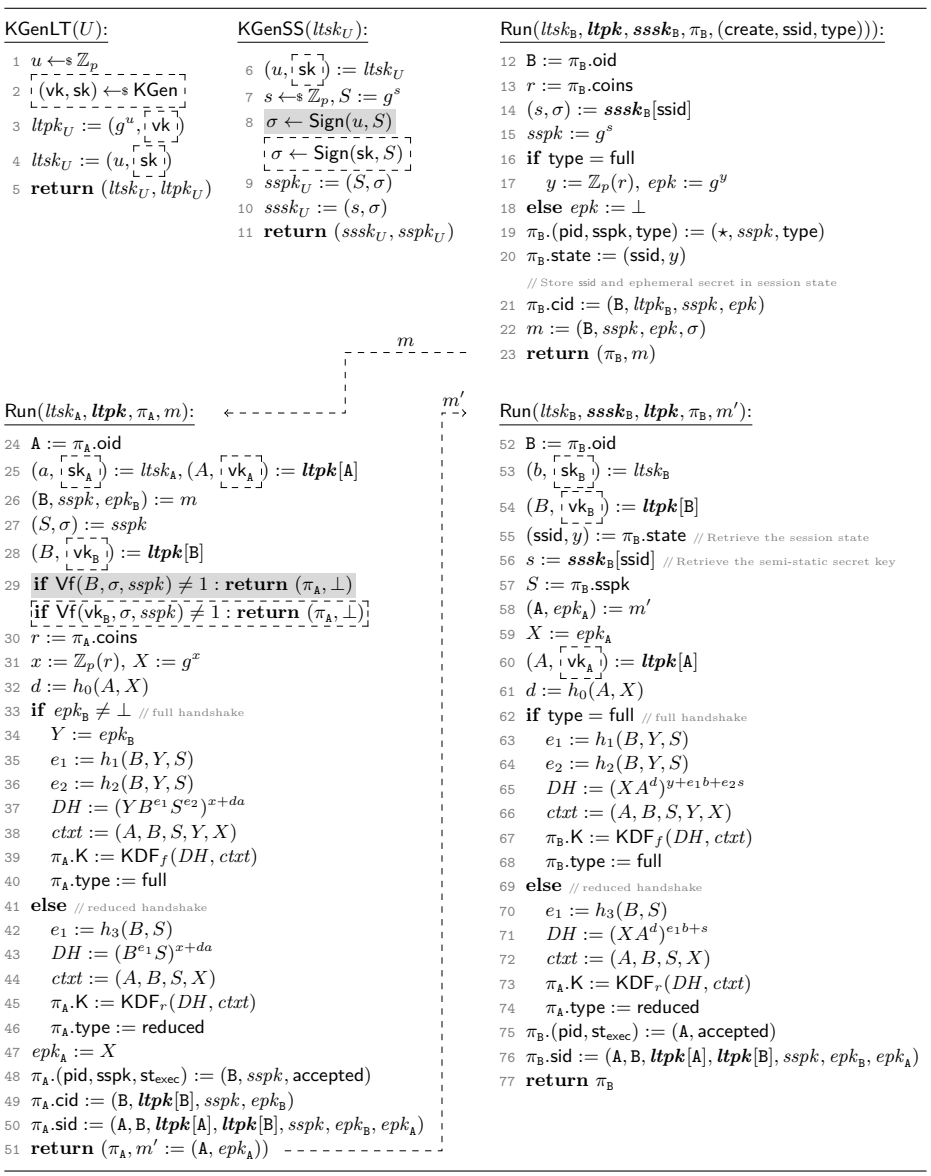

Fig. 9. Our XHMQV protocol and its variant XHMQV$^+$. Elements in gray boxes are only present in XHMQV, those in dashed boxes are only present in XHMQV$^+$.

5 Security Proof of XHMQV (Theorem 1)

We start with $\mathbf{G}_0$ which is the original key exchange security game $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A})$ with two modifications to simplify our proof: (1) For each session π^i_{U}, we add a variable $\pi^i_{\mathrm{U}}.ctxt$ into the session to store the context value $ctxt$ used in computing the session key. By using this variable, we can easily verify if a hash query to KDF_f or KDF_r matches the public keys (i.e., long-term, semi-static, and ephemeral keys) used in a session. (2) We use random oracles to simulate all hash functions. Since the proof is in the ROM, these changes are conceptual, so we have

$$\mathsf{Adv}^{\mathsf{KI}}_{\mathsf{KE}}(\mathcal{A}) = \mathsf{Adv}^{\mathbf{G}_0}_{\mathsf{XHMQV}}(\mathcal{A}).$$

GAME $\mathbf{G}_1$: This game excludes collisions of DH keys. Specifically, we abort the game (i.e., overwrite $\mathcal{A}$'s output with 0) if any two honestly generated DH key pairs collide. In the game, there are at most n_u long-term key pairs, n_{ss} semi-static key pairs for each user, and n_s ephemeral DH key pairs. By the birthday bound (over $\mathbb{G}$ with $|\mathbb{G}| = p$), we have

$$\mathsf{Adv}^{\mathbf{G}_0}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathbf{G}_1}_{\mathsf{XHMQV}}(\mathcal{A}) + \frac{(n_u + n_u \cdot n_{ss} + n_s)^2}{p}$$

GAME $\mathbf{G}_2$: We introduce a new abort event `Forge` in $\mathbf{G}_2$: If $\mathcal{A}$ forges a signature on a semi-static key with respect to an uncorrupted long-term key, then we abort the game. We denote this event as `Forge`. If `Forge` does not happen, then the distribution of $\mathbf{G}_2$ is the same as the one of $\mathbf{G}_1$. In the following, we construct an adversary $\mathcal{F}$ against the EUF-opCMA-DDH security of Sig to bound $\Pr[\texttt{Forge}]$. In the description of $\mathcal{F}$, we underline how we embed the challenge into the game and how we get the forgery if `Forge` happens.

Reduction $\mathcal{F}^{\mathrm{DDH},\mathrm{SIGN}}(\mathsf{vk}^* = B^*)$:

– Initialize $\mathbf{G}_2$:
1. Initialize two patch key lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$ (for full sessions) and $\mathcal{L}_{\mathsf{patch\text{-}r}}$ (for reduced sessions) to store all keys that are needed to be patched to the random oracle. For example, if there is a tuple (A, B, S, Y, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}f}}$, then it means that there is a session π such that $\pi.\mathsf{K} = K$ and $\pi.ctxt = (A, B, S, Y, X)$.
2. Initialize a candidate key list (for full sessions) $\mathcal{L}_{\mathsf{skey\text{-}f}}$ to deal with the case that the adversary first computes the key itself (and queries the RO) and finishes the session later. Similarly, initialize and a key candidate list $\mathcal{L}_{\mathsf{skey\text{-}r}}$ for reduced sessions. For example, if there is a tuple (A, B, S, Y, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}f}}$, then it means that $\mathcal{A}$ has queried KDF_f on (A, B, S, Y, X, DH) (where DH is the correct XHMQV secret). Here (A, B, S, Y, X) may correspond to the context of a session (that is not yet completed). If $\mathcal{A}$ completes such a session, then we can set its session key as K.
3. Guess a party $\mathtt{B}^*$ and set its long-term public key to B^*. When generating semi-static keys S of $\mathtt{B}^*$, $\mathcal{F}$ queries $\mathrm{SIGN}(S)$ to get the signatures. Note that we exclude collisions of all DH key pairs, so all S are unique and we only need to query $\mathrm{SIGN}(S)$ once for each S.

4. For all other parties, run KGenLT and KGenSS faithfully. After this step, $\mathcal{F}$ has $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$ (except for the long-term secret key of $\mathtt{B}^*$).
5. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceed the game as in $\mathbf{G}_1$. The simulation of random oracles and game oracles are simulated in the way described below.
6. If the adversary never forges a valid signature on one of $\mathtt{B}^*$'s semi-static keys before corrupting $\mathtt{B}^*$, then abort the simulation and return a random forgery.

– Simulation of oracles defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$:
  • All oracles except Send and CorruptLTKey are simulated as defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$. For CorruptLTKey, $\mathcal{F}$ aborts if $\mathcal{A}$ queried for the long-term secret key of $\mathtt{B}^*$ (in which case `Forge` cannot happen afterwards) and answers with the corresponding secret key for all other users. For Send queries, if the query does not involve $\mathtt{B}^*$, then we also simulate the oracles in the same way as in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$.
  • Send$(\mathtt{B}^*, i, m)$: We only describe the changes to the Run algorithm and the setup of session keys.
    ∗ $\mathsf{Run}(ltsk_{\mathtt{B}^*}, \boldsymbol{sssk}_{\mathtt{B}^*}, \boldsymbol{ltpk}, \pi^i_{\mathtt{B}^*}, (\mathsf{create}, \mathsf{ssid}, \mathsf{type}))$ – responder creation: Simulated as specified.
    ∗ $\mathsf{Run}(ltsk_{\mathtt{B}^*}, \boldsymbol{sssk}_{\mathtt{B}^*}, \boldsymbol{ltpk}, \pi^i_{\mathtt{B}^*}, m = (\mathtt{U}, S, Y, \sigma))$ – initiator: We describe handling a full handshake ($Y \neq \perp$); handling the reduced mode works analogously.
      1. If σ is a valid signature of $\mathtt{B}^*$ and $S \notin \boldsymbol{sspk}_{\mathtt{B}^*}$, then we abort the simulation and output (S, σ).
      2. Generate the ephemeral key pair (X, x).
      3. Let U be the long-term public key of $\mathtt{U}$. If there exists an entry (B, U, S, Y, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}f}}$, then retrieve this K. Otherwise, sample a random K and record (B, U, S, Y, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}f}}$.
      4. Set $\pi^i_{\mathtt{B}^*}.\mathsf{K} := K$.
    ∗ $\mathsf{Run}(ltsk_{\mathtt{B}^*}, \boldsymbol{sssk}_{\mathtt{B}^*}, \boldsymbol{ltpk}, \pi^i_{\mathtt{B}^*}, m' = (\mathtt{U}, epk_{\mathtt{U}} = X))$ – responder completion: We describe handling a full handshake; handling the reduced mode works analogously.
      1. Get the semi-static secret s and the ephemeral secret y (and their public keys) of this session. Let U be $\mathtt{U}$'s long-term public key.
      2. If there exists an entry (U, B, S, Y, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}f}}$, then retrieve this K. Otherwise, sample a random K and record (U, B, S, Y, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}f}}$.
      3. Set $\pi^i_{\mathtt{B}^*}.\mathsf{K} := K$.
  • Send$(\mathtt{U}, i, m = (\mathtt{B}^*, S, \sigma))$: If σ is a valid signature and $S \notin \boldsymbol{sspk}_{\mathtt{B}^*}$, then we abort the simulation and output (S, σ).

– Simulation of random oracles h_0, h_1, h_2, h_3, KDF_f, and KDF_r: We only discuss new queries. For repeated queries, $\mathcal{F}$ returns the previously recorded values.
  • $\mathsf{KDF}_f(DH, A, B, S, Y, X)$:
    1. If $\mathrm{DDH}(XA^d, YB^{e_1}S^{e_2}, DH)$ outputs 1:
       ∗ If there exists (A, B, S, Y, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}f}}$, retrieve this K.

    - If not, then sample a random key K and record (A, B, S, Y, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}f}}$.
    - Set $\mathsf{KDF}_f(DH, A, B, S, Y, X) := K$
    - Return K.
  2. Otherwise, simulate via lazy sampling.
- $\mathsf{KDF}_r(DH, A, B, S, X)$:
  1. If $\textsc{Ddh}(XA^d, SB^{e_1}, DH)$ outputs 1:
    - If there exists an entry (A, B, S, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}r}}$, retrieve this K.
    - If not, then sample a random key K and record (A, B, S, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}r}}$.
    - $\mathsf{KDF}_r(DH, A, B, S, X) := K$.
    - Return K.
  2. Otherwise, simulate via lazy sampling.

If `Forge` occurs, meaning that $\mathcal{A}$ forges a signature for a party's semi-static key before corrupting the party, then $\mathcal{F}$ can use this forgery to win the unforgeability game if $\mathtt{B}^*$ is the party whose signature $\mathcal{A}$ has forged. Since the guess of $\mathcal{B}^*$ is independent of $\mathcal{A}$'s view, we have

$$\frac{1}{n_u} \cdot \Pr[\mathtt{Forge}] \leq \Pr\left[\mathsf{Adv}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}}(\mathcal{F})\right]$$

Therefore, we have

$$\begin{aligned}\mathsf{Adv}^{\mathbf{G}_1}_{\mathsf{XHMQV}}(\mathcal{A}) &\leq \mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A}) + \Pr[\mathtt{Forge}] \\ &\leq \mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A}) + n_u \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}opCMA\text{-}DDH}}_{\mathsf{Sig}}(\mathcal{F}).\end{aligned}$$

Splitting $\mathbf{G}_2$ into clean cases: We now divide the proof into six sub-cases according to which clean predicate the test session π^* satisfies. Let $\mathbf{G}_2 \wedge \mathsf{clean}_X$ denote $\mathbf{G}_2$ conditioned on clean predicate clean_X being satisfied. We can bound the advantage of $\mathcal{A}$ in $\mathbf{G}_2$ by the sum of $\mathcal{A}$'s advantages in $\mathbf{G}_2 \wedge \mathsf{clean}_X$, for $\mathsf{clean}_X \in \{\mathsf{clean}_{\mathsf{LTLT}}, \mathsf{clean}_{\mathsf{LTE}}, \mathsf{clean}_{\mathsf{LTSS}}, \mathsf{clean}_{\mathsf{ELT}}, \mathsf{clean}_{\mathsf{ESS}}, \mathsf{clean}_{\mathsf{ELT}}\}$ being one of the clauses in $\mathsf{clean}^{\mathsf{XHMQV}}$:

$$\mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \sum_{\substack{\mathsf{clean}_X \in \{\mathsf{clean}_{\mathsf{LTLT}}, \mathsf{clean}_{\mathsf{LTE}}, \mathsf{clean}_{\mathsf{LTSS}}, \\ \mathsf{clean}_{\mathsf{ELT}}, \mathsf{clean}_{\mathsf{ESS}}, \mathsf{clean}_{\mathsf{EE}}\}}} \mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_X}_{\mathsf{XHMQV}}(\mathcal{A}).$$

Note that predicates $\mathsf{clean}_{\mathsf{LTE}}$ and $\mathsf{clean}_{\mathsf{EE}}$ only apply to a test session in full handshake mode ($\pi^*.\mathsf{type} = \mathsf{full}$).

We bound the probabilities of each predicate in Appendix A. For space reasons, we summarize the resulting bound in the following lemma and only give a proof sketch illustrating the common proof strategy across the clean cases.

Lemma 2. *With the notations and assumptions from Theorem 1, let $\epsilon_{\mathsf{CRGapDH}}$ be the advantage bound for the* CRGapDH *assumption in $\mathbb{G}$ and δ_{sim} be the bound*

for simulatability of Sig*, we have*

$$\begin{aligned}
\mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \quad & n_u n_u \quad \cdot (2\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{RO}/p) && /\!/\ \mathsf{clean}_{\mathsf{LTLT}} \\
+\ & n_u n_s \quad \cdot (\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{RO}/p) && /\!/\ \mathsf{clean}_{\mathsf{LTE}} \\
+\ & n_u^2 n_{ss} \quad \cdot (\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{RO}/p) && /\!/\ \mathsf{clean}_{\mathsf{LTSS}} \\
+\ & n_u n_s \quad \cdot (\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{RO}/p) && /\!/\ \mathsf{clean}_{\mathsf{ELT}} \\
+\ & n_s n_u n_{ss} \cdot (\epsilon_{\mathsf{CRGapDH}} + Q_{RO}/p) && /\!/\ \mathsf{clean}_{\mathsf{ESS}} \\
+\ & n_s^2 \quad \cdot (\epsilon_{\mathsf{CRGapDH}} + Q_{RO}/p)\,. && /\!/\ \mathsf{clean}_{\mathsf{EE}}
\end{aligned}$$

Let us illustrate the proof strategy at the example of the $\mathsf{clean}_{\mathsf{LTLT}}$ case, i.e., bounding $\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTLT}}}_{\mathsf{XHMQV}}(\mathcal{A})$. We proceed via the following steps:

1. We first guess the initiator and responder party of the test session π^*, picking two user identities $\mathtt{A}^*$ and $\mathtt{B}^*$ uniformly at random and aborting if that guess was incorrect. This leads to the multiplicative security loss of n_u^2.
2. We then use the δ-simulatability of Sig to replace signatures of $\mathtt{A}^*$ and $\mathtt{B}^*$. More specifically, we change the game such that all signatures of these two users are generated by running $\mathsf{SIMsign}$ and programming random oracles. This step allows us to simulate signatures only using public keys, which is essential for later reductions because we do not have the secret keys when constructing the $\mathsf{CRGapDH}$ reduction.
3. We then use the $\mathsf{CRGapDH}$ assumption over $\mathbb{G}$ to exclude the case that the adversary queries the random oracle on the hash input of π^*'s session key. To this end, we will embed the $\mathsf{CRGapDH}$ challenge into the public keys of the two users $\mathtt{A}^*$ and $\mathtt{B}^*$. Roughly, if $\mathcal{A}$ queries KDF_f or KDF_r on $(DH, ctxt)$, where $ctxt = \pi^*.ctxt$ and DH is the correct XHMQV's DH shared secret, then we can extract a $\mathsf{CRGapDH}$ solution from such a query.

The other proof cases have a similar structure: First, we guess where to embed the $\mathsf{CRGapDH}$ challenge, leading to different guessing losses depending on the clean case. Second, for all clean cases involving a long-term key, i.e., the first four cases, we use the simulatability of Sig to replace the long-term key, resulting in a δ_{sim} term. Thirdly, we reduce to $\mathsf{CRGapDH}$, obtaining the term $\epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p$. By combining all probability bounds, we obtain

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{KI}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq\ & \frac{(n_u + n_u n_{ss} + n_s)^2}{p} + n_u \cdot \epsilon_{\mathsf{EUF\text{-}opCMA\text{-}DDH}} \\
& + \left(2n_u^2 + n_u n_{ss} + n_u^2 n_{ss} + n_u n_s\right) \cdot \delta_{sim} \\
& + \Big(n_u^2 + n_u n_{ss} + n_u^2 n_{ss} \\
& \qquad + n_u n_s + n_s n_u n_{ss} + n_{ss}^2\Big) \cdot \left(\epsilon_{\mathsf{CRGapDH}} + \frac{Q_{\mathrm{RO}}}{p}\right),
\end{aligned}$$

This concludes our proof of Theorem 1. □

6 Deniability of XHMQV

Signal strives for deniability of its initial handshake protocol, as discussed in their descriptions of X3DH and PQXDH [41,37, both Section 4.4]. The deniability

property that Signal aims for considers transcripts of protocol executions. Given a transcript, no third party should be able to tell if the transcript is legitimate or if it was created using solely one party's (long-term and semi-static) secret keys; even if the third party has access to both parties' secret keys. Precisely this notion was formalized by Brendel, Fiedler, Günther, Janson, and Stebila [8] in a game-based setting.

We consider types of *offline deniability*, i.e., the distinguisher has to decide after the fact. In contrast, *online deniability* [20,51,52] allows the distinguisher to interact during the protocol execution. However, Unger and Goldberg's "Iron Triangle" [51, Section 6.6] presumes it is impossible to combine asynchronicity, forward secrecy, and online deniability in a key exchange protocol.

Deniable key exchange. Prior work has produced several deniability definitions for key exchange [18,32,15,16,54,30,53,31,28,8,24,13]. Most definitions follow the same blueprint[8]: An adversary queries a challenge oracle, which either acts as a real user (by executing the Run algorithm and using that user's secret key) or *pretends* to be that user (by executing the Fake algorithm or the simulator, depending on the formalism, but without that user's secret key) to answer the adversary. Afterwards, the distinguisher gets the transcript of the adversary's interaction with the challenge oracle and has to guess if the challenge oracle used Run or Fake. Intuitively, this guarantees that a transcript does not prove involvement of a party since the transcript could have been produced by the Fake algorithm.

Fiedler and Janson [24] provide a modular definition, which eases comparing deniability notions, with the following parameters:

1. adversarial power (is the adversary *malicious*, i.e., it may deviate from the protocol flow, or *semi-honest*?);
2. power of the Fake algorithm (can anybody fake a transcript (1-*out-of*-∞ deniability) or only the peer (1-*out-of*-2 deniability)? When producing a message for Bob, does the Fake algorithm need access to another session of Bob?);
3. power of the distinguisher (does the distinguisher get all honestly generated secret keys (*big brother model*)?);
4. auxiliary inputs (do we assume the adversary, the Fake algorithm, and potentially the distinguisher get common extra input, e.g., transcripts of honest protocol executions?).

We recount the full definition in Appendix B.

Deniability of Signal's initial handshake. Vatandas, Gennaro, Ithurburn, and Krawczyk [53] show deniability of HMQV and X3DH against malicious adversaries by relying on the novel Extended Knowledge of Diffie–Hellman assumption

[8] We describe the blueprint for the game-based setting, since our proof uses the game-based setting as well. For definitions in the simulation-based setting the formalism differs but the idea remains the same.

(EKDHA). For X3DH, they additionally rely on the auxiliary input to obtain a signature on Bob's semi-static key. Unfortunately, Fiedler and Langrehr [25] disprove the EKDHA and deniability of X3DH against malicious adversaries by relying on simple auxiliary input. Fiedler and Janson [24, Table 2] compare the deniability guarantees for X3DH and PQXDH by considering Alice and Bob separately: To achieve deniability for Bob, they model two ways for the Fake algorithm to obtain a signature on Bob's semi-static key and for PQXDH additionally on an ephemeral key that the distinguisher never sees.[9] Against malicious adversaries, they require the EKDHA for both protocols and for PQXDH additionally plaintext awareness of the KEM.[10] The two following results hold even in the big brother model: Against semi-honest adversaries, Fiedler and Janson show that either party can produce the transcript (1-out-of-2 deniability) by relying on the commutativity of DH operations (i.e., $(g^x)^y = (g^y)^x$). Similarly, against malicious adversaries restricted to honestly generated long-term and semi-static keys, they show deniability for Alice without assumptions: There, the Fake algorithm uses Bob's semi-static secret key to compute the long-term–semi-static contribution and a freshly sampled ephemeral key for Alice to compute the other three contributions. With this approach the Fake algorithm cannot compute a long-term–ephemeral contribution (which is not present in X3DH and PQXDH), but, looking ahead, prevents us from proving deniability for XHMQV in this setting. Though, it is not clear to us which scenario we model by forcing the adversary to sample semi-static keys honestly but not the ephemeral keys.

Deniability of post-quantum replacements for Signal's initial handshake. Hashimoto, Katsumata, Kwiatkowski, and Prest [28] propose SC-DAKE' based on KEMs, ring signatures, and NIZKs, which they prove deniable under the model of [18]. Dobson and Galbraith [19] propose SI-X3DH based on supersingular isogenies, broken by the SIDH attack [10,40,48], and sketch a deniability argument based on X3DH. Brendel et al. [8] propose SPQR based on KEMs and designated verifier signatures (DVS) and show its deniability against semi-honest adversaries in the big brother model for Alice with their novel deniability definition matching the requirements of X3DH [41, Section 4.4]. Collins, Huguenin-Dumittan, Nguyen, Rolin, and Vaudenay [9] propose K-Waay, which combines KEMs with a splitKEM and show its deniability under a novel definition close to that of [8].

Deniability of XHMQV*.* In the following, we show that our protocol XHMQV achieves the same deniability guarantees against semi-honest adversaries in the big brother model as X3DH and PQXDH.[11] For malicious adversaries limited to honestly generated (long-term and semi-static) keys our XHMQV protocol does not provide the deniability guarantees of X3DH. We cannot avoid this since the long-term–ephemeral contribution proves Alice's involvement if Bob provably

[9] The Fake learns the signature via the auxiliary input or with the oracle to another session of Bob.

[10] Note that Signal's implementation uses Kyber as KEM, which does not achieve plaintext awareness.

[11] SPQR is proven to achieve the same guarantee only for Alice, but not for Bob.

does not know his ephemeral secret key. For malicious adversaries (which are not artificially limited to honestly generated long-term and semi-static keys) the currently only known proof strategy for X3DH involves the broken EKDHA assumption. Hence, XHMQV is the same as X3DH and PQXDH for not having a (viable) proof of deniability against malicious adversaries.

We use the model of [24], formally given in Definition 5, which considers Bob as initiator and Alice as responder. We split the theorems by role, beginning with the case of Bob, for whom the Fake algorithm needs to obtain signatures on the semi-static keys. Following [24], we do this via the auxiliary input and note that it can similarly be done with the USER oracle that allows Fake to access another session with Bob. When answering for Alice, the Fake algorithm does not need auxiliary input or the USER oracle, since Alice does not sign anything. The proof strategy follows [24, Theorem 4.1]: The Fake algorithm uses its SK oracle to learn the peer's long-term and semi-static secret keys and the extracts the ephemeral secret key from the adversary's randomness. Using these secret keys it computes the session key. We give the proofs in Appendix B.

Theorem 2. *The* XHMQV *protocol as shown in Figure 9 is* $(n_p, n_{ss}, q_{\mathrm{O_A}}, q_{\mathrm{O_F}}, q_{\mathrm{O_D}}, t_{\mathcal{A}}, t_{\mathsf{Dist}}, \epsilon_{\mathsf{Dist}})$*-deniable with respect to* $(\{\textsc{RegHon}, \textsc{ChallHonInit}\}, \{\mathrm{SK}\}, \{\mathrm{SKs}, \mathrm{AUX}\})$*-oracles and auxiliary info sampled with* AuxPrep *yielding a valid pre-key bundle per* ssid *and user, and aux known to the distinguisher, where* n_p *is the number of parties and* n_{ss} *the number of semi-static keys per party, and* $q_{\mathrm{O_A}}, q_{\mathrm{O_F}}, q_{\mathrm{O_D}}, t_{\mathcal{A}}, t_{\mathsf{Dist}}$ *are arbitrary and* $\epsilon_{\mathsf{Dist}} = 0$.

Theorem 3. *The* XHMQV *protocol as shown in Figure 9 is* $(n_p, n_{ss}, q_{\mathrm{O_A}}, q_{\mathrm{O_F}}, q_{\mathrm{O_D}}, t_{\mathcal{A}}, t_{\mathsf{Dist}}, \epsilon_{\mathsf{Dist}})$*-deniable with respect to* $(\{\textsc{RegHon}, \textsc{ChallHonResp}\}, \{\mathrm{SK}\}, \{\mathrm{SKs}, \mathrm{AUX}\})$*-oracles and auxiliary info sampled with arbitrary* AuxPrep*, and aux known to the distinguisher, where* n_p *is the number of parties and* n_{ss} *the number of semi-static keys per party, and* $q_{\mathrm{O_A}}, q_{\mathrm{O_F}}, q_{\mathrm{O_D}}, t_{\mathcal{A}}, t_{\mathsf{Dist}}$ *are arbitrary and* $\epsilon_{\mathsf{Dist}} = 0$.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. R.F. was supported by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Berlin, Heidelberg (Apr 2001). https://doi.org/10.1007/3-540-45353-9_12
2. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Berlin, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_6
3. Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 3–32. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45724-2_1
4. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Berlin, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_21
5. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Berlin, Heidelberg (Apr 2006). https://doi.org/10.1007/11745853_14
6. Bhargavan, K., Jacomme, C., Kiefer, F., Schmidt, R.: Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/bhargavan
7. Brendel, J., Fiedler, R., Günther, F., Janson, C., Stebila, D.: Post-quantum asynchronous deniable key exchange and the Signal handshake. Cryptology ePrint Archive, Report 2021/769 (2021), https://eprint.iacr.org/2021/769
8. Brendel, J., Fiedler, R., Günther, F., Janson, C., Stebila, D.: Post-quantum asynchronous deniable key exchange and the Signal handshake. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part II. LNCS, vol. 13178, pp. 3–34. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-97131-1_1
9. Carlini, N., Hayes, J., Nasr, M., Jagielski, M., Sehwag, V., Tramèr, F., Balle, B., Ippolito, D., Wallace, E.: Extracting training data from diffusion models. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 5253–5270. USENIX Association (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/carlini
10. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 423–447. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_15
11. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the Signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy. pp. 451–466. IEEE Computer Society Press (Apr 2017). https://doi.org/10.1109/EuroSP.2017.27
12. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the Signal messaging protocol. Journal of Cryptology **33**(4), 1914–1983 (Oct 2020). https://doi.org/10.1007/s00145-020-09360-1
13. Collins, D., Huguenin-Dumittan, L., Nguyen, N.K., Rolin, N., Vaudenay, S.: K-waay: Fast and deniable post-quantum X3DH without ring signatures. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association (Aug

2024), https://www.usenix.org/conference/usenixsecurity24/presentation/collins
14. Cremers, C., Dax, A., Medinger, N.: Keeping up with the KEMs: Stronger security notions for KEMs and automated analysis of KEM-based protocols. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 1046–1060. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3670283
15. Cremers, C., Feltz, M.: One-round strongly secure key exchange with perfect forward secrecy and deniability. Cryptology ePrint Archive, Report 2011/300 (2011), https://eprint.iacr.org/2011/300
16. Dagdelen, Ö., Fischlin, M., Gagliardoni, T., Marson, G.A., Mittelbach, A., Onete, C.: A cryptographic analysis of OPACITY - (extended abstract). In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 345–362. Springer, Berlin, Heidelberg (Sep 2013). https://doi.org/10.1007/978-3-642-40203-6_20
17. Degabriele, J.P., Lehmann, A., Paterson, K.G., Smart, N.P., Strefler, M.: On the joint security of encryption and signature in EMV. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 116–135. Springer, Berlin, Heidelberg (Feb / Mar 2012). https://doi.org/10.1007/978-3-642-27954-6_8
18. Di Raimondo, M., Gennaro, R., Krawczyk, H.: Deniable authentication and key exchange. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 400–409. ACM Press (Oct / Nov 2006). https://doi.org/10.1145/1180405.1180454
19. Dobson, S., Galbraith, S.D.: Post-quantum Signal key agreement from SIDH. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022. pp. 422–450. Springer, Cham (Sep 2022). https://doi.org/10.1007/978-3-031-17234-2_20
20. Dodis, Y., Katz, J., Smith, A., Walfish, S.: Composability and on-line deniability of authentication. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 146–162. Springer, Berlin, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00457-5_10
21. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1197–1210. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813653
22. Fersch, M., Kiltz, E., Poettering, B.: On the one-per-message unforgeability of (EC)DSA and its variants. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. LNCS, vol. 10678, pp. 519–534. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70503-3_17
23. Fiedler, R., Günther, F.: Security analysis of Signal's PQXDH handshake. In: Jager, T., Pan, J. (eds.) PKC 2025, Part II. LNCS, vol. 15675, pp. 137–169. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91823-0_5
24. Fiedler, R., Janson, C.: A deniability analysis of Signal's initial handshake PQXDH. PoPETs **2024**(4), 907–928 (Oct 2024). https://doi.org/10.56553/popets-2024-0148
25. Fiedler, R., Langrehr, R.: On deniable authentication against malicious verifiers. Cryptology ePrint Archive, Paper 2025/470 (2025), https://eprint.iacr.org/2025/470
26. Haber, S., Pinkas, B.: Securely combining public-key cryptosystems. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001. pp. 215–224. ACM Press (Nov 2001). https://doi.org/10.1145/501983.502013

27. Hashimoto, K., Katsumata, S., Kwiatkowski, K., Prest, T.: An efficient and generic construction for Signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. In: Garay, J. (ed.) PKC 2021, Part II. LNCS, vol. 12711, pp. 410–440. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75248-4_15
28. Hashimoto, K., Katsumata, S., Kwiatkowski, K., Prest, T.: An efficient and generic construction for Signal's handshake (X3DH): Post-quantum, state leakage secure, and deniable. Journal of Cryptology **35**(3), 17 (Jul 2022). https://doi.org/10.1007/s00145-022-09427-1
29. Hashimoto, K., Katsumata, S., Wiggers, T.: Bundled authenticated key exchange: A concrete treatment of (post-quantum) Signal's handshake protocol. In: 34th USENIX Security Symposium, USENIX Security 2025. USENIX Association (Aug 2025), to appear. Available as Cryptology ePrint Archive Report 2025/040, https://eprint.iacr.org/2025/040.
30. Jiang, S.: Timed encryption with application to deniable key exchange. Theor. Comput. Sci. **560**, 172–189 (2014). https://doi.org/10.1016/J.TCS.2014.02.005, https://doi.org/10.1016/j.tcs.2014.02.005
31. Jiang, S., Chee, Y.M., Ling, S., Wang, H., Xing, C.: A new framework for deniable secure key exchange. Inf. Comput. **285**(Part), 104866 (2022). https://doi.org/10.1016/J.IC.2022.104866, https://doi.org/10.1016/j.ic.2022.104866
32. Jiang, S., Safavi-Naini, R.: An efficient deniable key exchange protocol (extended abstract). In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 47–52. Springer, Berlin, Heidelberg (Jan 2008). https://doi.org/10.1007/978-3-540-85230-8_4
33. Kiltz, E., Masny, D., Pan, J.: Optimal security proofs for signatures from identification schemes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 33–61. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53008-5_2
34. Kiltz, E., Pan, J., Riepel, D., Ringerud, M.: Multi-user CDH problems and the concrete security of NAXOS and HMQV. In: Rosulek, M. (ed.) CT-RSA 2023. LNCS, vol. 13871, pp. 645–671. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30872-7_25
35. Kobeissi, N., Bhargavan, K., Blanchet, B.: Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In: 2017 IEEE European Symposium on Security and Privacy. pp. 435–450. IEEE Computer Society Press (Apr 2017). https://doi.org/10.1109/EuroSP.2017.38
36. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Berlin, Heidelberg (Aug 2005). https://doi.org/10.1007/11535218_33
37. Kret, E., Schmidt, R.: The PQXDH key agreement protocol (Jan 2024), https://signal.org/docs/specifications/pqxdh/
38. Langley, A., Hamburg, M., Turner, S.: Elliptic Curves for Security. RFC 7748 (Informational) (Jan 2016). https://doi.org/10.17487/RFC7748, https://www.rfc-editor.org/rfc/rfc7748.txt
39. Law, L., Menezes, A., Qu, M., Solinas, J.A., Vanstone, S.A.: An efficient protocol for authenticated key agreement. DCC **28**(2), 119–134 (2003). https://doi.org/10.1023/A:1022595222606
40. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A direct key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 448–471. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_16

41. Marlinspike, M., Perrin, T.: The X3DH key agreement protocol (Nov 2016), https://signal.org/docs/specifications/x3dh/
42. Menezes, A.: Another look at HMQV. Cryptology ePrint Archive, Report 2005/205 (2005), https://eprint.iacr.org/2005/205
43. Menezes, A., Qu, M., , Vanstone, S.A.: Some new key agreement protocols providing mutual implicit authentication. 2nd Workshop on Selected Areas in Cryptography (SAC 1995) (1995)
44. Paterson, K.G., Schuldt, J.C.N., Stam, M., Thomson, S.: On the joint security of encryption and signature, revisited. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 161–178. Springer, Berlin, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_9
45. Patton, C., Shrimpton, T.: Security in the presence of key reuse: Context-separable interfaces and their applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 738–768. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_26
46. Perrin, T.: The XEdDSA and VXEdDSA signature schemes (Oct 2016), https://signal.org/docs/specifications/xeddsa/
47. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (Jun 2000). https://doi.org/10.1007/s001450010003
48. Robert, D.: Breaking SIDH in polynomial time. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 472–503. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_17
49. Signal: Technical information, https://signal.org/docs/
50. Thormarker, E.: On using the same key pair for Ed25519 and an X25519 based KEM. Cryptology ePrint Archive, Report 2021/509 (2021), https://eprint.iacr.org/2021/509
51. Unger, N., Goldberg, I.: Deniable key exchanges for secure messaging. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1211–1223. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813616
52. Unger, N., Goldberg, I.: Improved strongly deniable authenticated key exchanges for secure messaging. PoPETs **2018**(1), 21–66 (Jan 2018). https://doi.org/10.1515/popets-2018-0003
53. Vatandas, N., Gennaro, R., Ithurburn, B., Krawczyk, H.: On the cryptographic deniability of the Signal protocol. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 2020, Part II. LNCS, vol. 12147, pp. 188–209. Springer, Cham (Oct 2020). https://doi.org/10.1007/978-3-030-57878-7_10
54. Yao, A.C.C., Zhao, Y.: OAKE: a new family of implicitly authenticated Diffie-Hellman protocols. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 1113–1128. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516695

A Full Proof of Bounding $\mathbf{G_2}$ in Theorem 1

In this section, we formally bound the probability from Theorem 1:

$$\mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \sum_{\substack{\mathsf{clean}_X \in \{\mathsf{clean}_{\mathsf{LTLT}}, \mathsf{clean}_{\mathsf{LTE}}, \mathsf{clean}_{\mathsf{LTSS}}, \\ \mathsf{clean}_{\mathsf{ELT}}, \mathsf{clean}_{\mathsf{ESS}}, \mathsf{clean}_{\mathsf{EE}}\}}} \mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_X}_{\mathsf{XHMQV}}(\mathcal{A}).$$

We prove each case in different subsections below; concrete security bounds for each case can be found in these subsections. Here we only give the final security bound of $\mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A})$, i.e., the bound in Lemma 2 where $\epsilon_{\mathsf{CRGapDH}}$ denotes the advantage bound for the $\mathsf{CRGapDH}$ assumption in $\mathbb{G}$ and δ_{sim} the bound for simulatability of Sig:

$$\begin{aligned}
\mathsf{Adv}^{\mathbf{G}_2}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \quad & n_u n_u \quad \cdot (2\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p) && /\!/\ \mathsf{clean}_{\mathsf{LTLT}} \\
+\ & n_u n_s \quad \cdot (\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p) && /\!/\ \mathsf{clean}_{\mathsf{LTE}} \\
+\ & n_u^2 n_{ss} \quad \cdot (\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p) && /\!/\ \mathsf{clean}_{\mathsf{LTSS}} \\
+\ & n_u n_s \quad \cdot (\delta_{sim} + \epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p) && /\!/\ \mathsf{clean}_{\mathsf{ELT}} \\
+\ & n_s n_u n_{ss} \cdot (\epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p) && /\!/\ \mathsf{clean}_{\mathsf{ESS}} \\
+\ & n_s^2 \quad \cdot (\epsilon_{\mathsf{CRGapDH}} + Q_{\mathrm{RO}}/p)\,. && /\!/\ \mathsf{clean}_{\mathsf{EE}}
\end{aligned}$$

A.1 Case $\mathsf{clean}_{\mathsf{LTLT}}$

By the definition of $\mathsf{clean}_{\mathsf{LTLT}}$, the long-term keys of both $\pi^*.\mathsf{oid}$ and $\pi^*.\mathsf{pid}$ are uncompromised. We begin with $\mathcal{G}^{\mathsf{ltlt}}_0 := \mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTLT}}(\pi^*)$, so we have

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTLT}}(\pi^*)}_{\mathsf{XHMQV}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}^{\mathsf{ltlt}}_0}_{\mathsf{XHMQV}}(\mathcal{A})$$

We use game sequence $\mathcal{G}^{\mathsf{ltlt}}_1$-$\mathcal{G}^{\mathsf{ltlt}}_4$ to bound the winning advantage of $\mathcal{A}$ when π^* satisfies $\mathsf{clean}_{\mathsf{LTLT}}$. The transitions $\mathcal{G}^{\mathsf{ltlt}}_1$-$\mathcal{G}^{\mathsf{ltlt}}_3$ enable the game to simulate signatures without using long-term keys. We use the simulatability (cf. Definition 3) of the signature scheme to bound the probability difference. For $\mathcal{G}^{\mathsf{ltlt}}_3$-$\mathcal{G}^{\mathsf{ltlt}}_4$, we use the $\mathsf{CRGapDH}$ assumption to prove that if the adversary cannot query KDF_f (or KDF_r) on the correct hash input of π^*, then it cannot distinguish the real session key of π^* from a random key.

GAME $\mathcal{G}^{\mathsf{ltlt}}_1$: We guess the identities $\mathtt{A}^*$ and $\mathtt{B}^*$ of the initiator and responder involved in the test session (i.e., $(\pi^*.\mathsf{oid}, \pi^*.\mathsf{pid}) = (\mathtt{A}^*, \mathtt{B}^*)$). If this guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. Since there are at most n_u users involved in ths game and the guess is uniformly at random in $\mathcal{A}$'s view, we have

$$\mathsf{Adv}^{\mathcal{G}^{\mathsf{ltlt}}_0}_{\mathsf{XHMQV}}(\mathcal{A}) = n_u^2 \cdot \mathsf{Adv}^{\mathcal{G}^{\mathsf{ltlt}}_1}_{\mathsf{XHMQV}}(\mathcal{A})$$

For the sake of simplicity, let (a^*, A^*) and (b^*, B^*) be the long-term key pairs of $\mathtt{A}^*$ and $\mathtt{B}^*$, respectively. In the game transitions $\mathcal{G}^{\mathsf{ltlt}}_1$-$\mathcal{G}^{\mathsf{ltlt}}_3$, we change the simulation of h_{sig} (the random oracle used by the signature scheme) and the

$\underline{\mathsf{KGenSS}(ltsk_{\mathtt{B}^*} = b^*) \text{ for } \mathcal{G}_1^{\mathsf{ltlt}}}$:

$s \leftarrow\!\!\$\ \mathbb{Z}_p, S := g^s$
$\sigma \leftarrow \mathsf{Sign}^{h_{\mathsf{sig}}}(b^*, S)$
$sssk_U := s,\ sspk_U := (S, \sigma)$
return $(sssk_U, sspk_U)$

$\underline{h_{\mathsf{sig}}(qry)}$:

// Lazy sampling
if $\exists h$ s.t. $(qry, h) \in \mathcal{L}_{h_{\mathsf{sig}}}$
  return h
$h \leftarrow\!\!\$\ h_{\mathsf{sig}}.\mathsf{image}$
$\mathcal{L}_{h_{\mathsf{sig}}} := \mathcal{L}_{h_{\mathsf{sig}}} \cup (qry, h)$
return h

$\underline{\mathsf{KGenSS}(ltsk_{\mathtt{B}^*} = b^*) \text{ for } \mathcal{G}_2^{\mathsf{ltlt}}}$:

$s \leftarrow\!\!\$\ \mathbb{Z}_p, S := g^s$
$(\sigma, qry, h) \leftarrow\!\!\$\ \mathsf{SIMsign}(B^*, S)$
$\mathcal{L}_{\mathtt{sim}} := \mathcal{L}_{\mathtt{sim}} \cup \{(qry, h)\}$
~~$\sigma \leftarrow \mathsf{Sign}^{h_{\mathsf{sig}}}(b^*, S)$~~
$sssk_U := s,\ sspk_U := (S, \sigma)$
return $(sssk_U, sspk_U)$

$\underline{h_{\mathsf{sig}}(qry)}$:

if $\exists h$ s.t. $(qry, h) \in \mathcal{L}_{\mathtt{sim}}$
  return h // Program the RO s.t. $h_{\mathsf{sig}}(qry) = h$
$\cdots$ // As the lazy sampling in $\mathcal{G}_1^{\mathsf{ltlt}}$

Fig. 10. Code of KGenSS and h_{sig} for games $\mathcal{G}_1^{\mathsf{ltlt}}$ (left) and $\mathcal{G}_2^{\mathsf{ltlt}}$ (right)

generation of $\mathtt{A}^*$'s and $\mathtt{B}^*$'s semi-static keys before running $\mathcal{A}$.

$\underline{\text{GAME } \mathcal{G}_2^{\mathsf{ltlt}}}$: This game changes the random oracle h_{sig} (used by the signature scheme) and the key generation of semi-static keys for $\mathtt{B}^*$. Since the code is presented in Fig. 10, here we only give the high-level idea:

- When running $\mathsf{KGenSS}(ltsk_{\mathtt{B}^*})$, we use $\mathsf{SIMsign}$ to generate simulated signatures and some simulation information. The simulation information is recorded in $\mathcal{L}_{\mathtt{sim}}$ and will be used to program h_{sig} later.
- We use the outputs of $\mathsf{SIMsign}(B^*, \cdot)$ to simulate h_{sig}'s responses to queries that may be related to the signatures of $\mathtt{B}^*$'s semi-static keys.

In $\mathcal{G}_2^{\mathsf{ltlt}}$, the differences compared to $\mathcal{G}_1^{\mathsf{ltlt}}$ are (1) we use the signature from $\mathsf{SIMsign}$, and (2) we use the simulated hash value from $\mathsf{SIMsign}$ to simulate the random oracle h_{sig}. We construct a straight-forward reduction from δ-simulatability of Sig to bound these differences: Let $\mathcal{R}$ be an adversary against the simulatability of Sig. $\mathcal{R}$ plays against either $\mathcal{G}_{\mathsf{Sig}}^{Real}$ or $\mathcal{G}_{\mathsf{Sig}}^{Sim}$ and simulates either $\mathcal{G}_1^{\mathsf{ltlt}}$ or $\mathcal{G}_2^{\mathsf{ltlt}}$ for $\mathcal{A}$, respectively. $\mathcal{R}$ works as follows:

- $\mathcal{R}$ first samples b_{test} and a party id $\mathtt{B}^*$. It generates $\mathtt{B}^*$'s semi-static keys (s_i, S_i) for $1 \le i \le n_{ss}$, and outputs $(S_i)_{1 \le i \le n_{ss}}$ to the game. It then gets $(\mathsf{vk} = B^*, \mathsf{sk} = b^*, (\sigma_i)_{1 \le i \le [n_{ss}]})$ and random oracle access to RO.
- Then, $\mathcal{R}$ sets up all variables (long-term key pairs and semi-static key pairs of all other parties, session initialization, etc.) and oracles required in $\mathcal{G}_2^{\mathsf{ltlt}}$ and runs $\mathcal{A}$. All h_{sig} queries from $\mathcal{A}$ are forwarded to RO.
- When $\mathcal{A}$ outputs b_{guess}, $\mathcal{R}$ returns $[\![b_{\mathsf{test}} = b_{\mathsf{guess}}]\!]$.

It is straight-forward to see that if $\mathcal{R}$ is playing against $\mathcal{G}_{\mathsf{Sig}}^{Real}$, then it perfectly simulates $\mathcal{G}_1^{\mathsf{ltlt}}$ for $\mathcal{A}$. If $\mathcal{R}$ is playing against $\mathcal{G}_{\mathsf{Sig}}^{Real}$, then we have $h_{\mathsf{sig}}(qry') = h' = \mathsf{RO}(qry)$ if (σ', qry', h') is generated by $\mathsf{SIMsign}$. Hence, in this case $\mathcal{R}$ perfectly simulates $\mathcal{G}_2^{\mathsf{ltlt}}$. Therefore, we have

$$\left|\mathsf{Adv}_{\mathsf{XHMQV}}^{\mathcal{G}_1^{\mathsf{ltlt}}}(\mathcal{A}) - \mathsf{Adv}_{\mathsf{XHMQV}}^{\mathcal{G}_2^{\mathsf{ltlt}}}(\mathcal{A})\right| = \left|\Pr\left[\mathcal{G}_{\mathsf{Sig}}^{Real} = 1\right] - \Pr\left[\mathcal{G}_{\mathsf{Sig}}^{Sim} = 1\right]\right| \le \delta$$

and

$$\mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_2^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \delta.$$

From $\mathcal{G}_2^{\mathsf{ltlt}}$ on, we no longer need the long-term secret key of $\mathtt{B}^*$ to generate signatures on $\mathtt{B}^*$'s semi-static keys.

GAME $\mathcal{G}_3^{\mathsf{ltlt}}$: We apply the same approach used in $\mathcal{G}_2^{\mathsf{ltlt}}$ to $\mathtt{A}^*$. Namely, we use a modified KGenSS and change the simulation of h_{sig} so that we can simulate signatures on $\mathtt{A}^*$'s semi-static keys without using its long-term key. Similar to $\mathcal{G}_2^{\mathsf{ltlt}}$, we have

$$\mathsf{Adv}^{\mathcal{G}_2^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \delta,$$

and from $\mathcal{G}_3^{\mathsf{ltlt}}$ on, we no longer need the long-term secret key of $\mathtt{A}^*$.

GAME $\mathcal{G}_4^{\mathsf{ltlt}}$: We introduce a new abort event $\mathtt{QueryDH}_{\mathsf{ltlt}}$ in $\mathcal{G}_4^{\mathsf{ltlt}}$: Specifically, if

- $\mathcal{A}$ queries KDF_f on (DH, A^*, B^*, S, Y, X) , where $d = h_0(A^*, X), e_1 = h_1(B^*, Y, S), e_2 = h_2(B^*, Y, S)$, and $\textsc{Ddh}(X(A^*)^d, Y(B^*)^{e_1}S^{e_2}, DH) = 1$, and $(A^*, B^*, S, Y, X) = \pi^*.ctxt$, or
- $\mathcal{A}$ queries KDF_r on (DH, A^*, B^*, S, X), where $d = h_0(A^*, X), e_1 = h_3(B^*, S)$, and $\textsc{Ddh}(X(A^*)^d, S(B^*)^{e_1}, DH) = 1$ and $(A^*, B^*, S, X) = \pi^*.ctxt$,

then the game aborts (i.e., return 0 as $\mathcal{A}$'s bit guess). In other words, we abort the game if $\mathcal{A}$ queries KDF_f or KDF_r on any DH value that contains the information of $(B^*)^{a^*} (= (A^*)^{b^*})$.

To bound the probability of this event, we construct a $\mathsf{CRGapDH}$ adversary $\mathcal{B}_{\mathsf{ltlt}}$ such that if such an abort event happens, then $\mathcal{B}$ can output a solution to its $\mathsf{CRGapDH}$ challenge. By Definition 1, the input to $\mathcal{B}_{\mathsf{ltlt}}$ is (A^*, B^*) and $\mathcal{B}_{\mathsf{ltlt}}$ has access to oracles $\textsc{Ddh}$ and $\textsc{Ch}$. We underline how we embed the challenge into the game.

Reduction $\mathcal{B}_{\mathsf{ltlt}}^{\textsc{Ddh},\textsc{Ch}}(A^*, B^*)$:

- Initialize $\mathcal{G}_4^{\mathsf{ltlt}}$:
  1. Choose two users $\mathtt{A}^*$ and $\mathtt{B}^*$ uniformly at random and set the long-term public key of $\mathtt{A}^*$ and $\mathtt{B}^*$ as A^* and B^*, respectively.
  2. Initialize lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ for the simulation of session keys using $\textsc{Ddh}$ and random oracle patching (as in $\mathbf{G}_2$). We also initialize $\mathcal{L}_{\mathtt{sim}}$ to simulate the signatures on $\mathtt{A}^*$'s and $\mathtt{B}^*$'s semi-static keys (as in Fig. 10).
  3. For parties $\mathtt{A}^*$ and $\mathtt{B}^*$, run the modified KGenSS algorithm defined in Fig. 10. After this step, $\mathcal{B}_{\mathsf{ltlt}}$ has $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$ (except for the long-term secret keys of $\mathtt{A}^*$ and $\mathtt{B}^*$). For all other parties, follow the KGenLT and KGenSS defined in XHMQV faithfully.
  4. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceed the game as in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$. If the initiator and responder of the test session are not $\mathtt{A}^*$ and $\mathtt{B}^*$, respectively, then overwrite $\mathcal{A}$'s bit with 0.

     The random oracles and game oracles are simulated as described below.

– Simulation of oracles defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$:
  - All oracles except Send and CorruptLTKey are simulated in the same way defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$.
  - By $\mathsf{clean}_{\mathsf{LTLT}}$, the adversary does not query CorruptLTKey on $\mathtt{A}^*$ or $\mathtt{B}^*$. CorruptLTKey queries on other parties are simulated as usual (since we have long-term keys of all other parties).
  - For Send, if the query does not involve $\mathtt{A}^*$ and $\mathtt{B}^*$, then we also simulate the oracle in the same way as in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$.
  - Send$(\mathtt{A}^*, i, m)$ (We only present how to simulate the Run algorithms and set up the session keys):
    * Case $m = (\mathsf{create}, \mathsf{ssid}, \mathsf{type})$ (as the session responder): Simulate as usual.
    * Case $m = (\mathtt{U}, X)$ (as the session responder and $\mathsf{type} = \mathsf{full}$):
      1. Get the semi-static secret s, the ephemeral secret y (and their public keys) of this session. Let U be the long-term public key of the initiator party $\mathtt{U}$.
      2. If there exists an entry (U, A^*, S, Y, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}f}}$, then retrieve this K.
      3. Otherwise, sample a random K and record (U, A^*, S, Y, X, K) in the list $\mathcal{L}_{\mathsf{patch\text{-}f}}$.
      4. Set $\pi^i_{\mathtt{A}^*}.\mathsf{K} := K$.
    * Case $m = (\mathtt{U}, X)$ (as the session responder and $\mathsf{type} = \mathsf{reduced}$). This case is similar to the case of $\mathsf{type} = \mathsf{full}$. We present the details for completeness:
      1. Get the semi-static key pair (s, S) of this session. Let U be the long-term public key of $\mathtt{U}$.
      2. If there exists (U, A^*, S, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}r}}$, then retrieve this K.
      3. Otherwise, sample a random K and record (U, A^*, S, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}r}}$.
      4. Set $\pi^i_{\mathtt{A}^*}.\mathsf{K} := K$.
    * Case $m = (\mathtt{U}, S, Y, \sigma)$ (as the session initiator and $\mathsf{type} = \mathsf{full}$):
      1. Generate the ephemeral key pair (X, x) of this session. Let U be the long-term public key of the responder party $\mathtt{U}$.
      2. If there exists (A^*, U, S, Y, X, K) in $\mathcal{L}_{\mathsf{skey\text{-}f}}$, then retrieve this K.
      3. Otherwise, sample a random K and record (A^*, U, S, Y, X, K) in $\mathcal{L}_{\mathsf{patch\text{-}f}}$.
      4. Set $\pi^i_{\mathtt{A}^*}.\mathsf{K} := K$
    * Case $m = (\mathtt{U}, S, Y = \bot, \sigma)$ (as the session initiator) and $\mathsf{type} = \mathsf{reduced}$ works analogously to the case of $\mathsf{type} = \mathsf{full}$.
  - Send$(\mathtt{B}^*, j, m)$: The simulation is analogous to Send$(\mathtt{A}^*, i, m)$.

– Simulation of Random Oracles:
  - For fresh RO queries to $\mathsf{KDF}_f(A, B, S, Y, X, DH)$ and $\mathsf{KDF}_r(A, B, S, X, DH)$, where $(A, B) \neq (A^*, B^*)$, we simulate in the same way as in $\mathbf{G}_2$. Similarly, for fresh RO queries to h_0, h_1, h_2, and h_3 that do not involve $\mathtt{A}^*$ and $\mathtt{B}^*$, we simulate as usual.
  - $h_0(A^*, X)$: $h \leftarrow \mathrm{Ch}(X)$, $h_0(A^*, X) := h$.

- $h_1(B^*, Y, S)$: $e_2 := h_2(B^*, Y, S), R := YS^{e_2}, h \leftarrow \text{CH}(R), h_1(B^*, Y, S) := h.$
- $h_2(B^*, Y, S)$: Simulate as usual.
- $h_3(B^*, S)$: $h \leftarrow \text{CH}(S)$, $h_0(B^*, S) := h.$
- $\mathsf{KDF}_f(DH, A^*, B^*, S, Y, X)$:
  1. If $(A^*, B^*, S, Y, X) = \pi^*.ctxt$ and $\text{DDH}(X(A^*)^d, Y(B^*)^{e_1}S^{e_2}, DH) = 1$, then extract the value

$$DH' := \begin{cases} \left(DH/(X(A^*)^d)^{y+se_2}\right)^{e_1^{-1}}, & \pi^* \text{ is a responder session} \\ \left(DH/\left(Y(B^*)^{e_1}S^{e_2}\right)^x\right)^{d^{-1}}, & \pi^* \text{ is an initiator session} \end{cases}$$

     where $d := h_0(A^*, X), e_1 := h_1(B, Y, S)$, and $e_2 := h_2(B, Y, S)$. If π^* is an initiator session, then we have its ephemeral key x. If π^* is a responder session, then we have the ephemeral key y and the semi-static key s of the responder $\mathtt{B}^*$. Once such a DH' value is found, $\mathcal{B}_{\mathsf{ltlt}}$ aborts the simulation and returns DH' as the CRGapDH solution.
  2. Otherwise, simulate by lazy sampling.
- $\mathsf{KDF}_r(DH, A^*, B^*, S, X)$:
  1. If $(A^*, B^*, S, X) = \pi^*.ctxt$ and $\text{DDH}(X(A^*)^d, S(B^*)^{e_1}) = 1$, then extract the value

$$DH' := \begin{cases} \left(DH/(X(A^*)^d)^s\right)^{e_1^{-1}}, & \pi^* \text{ is a responder session} \\ \left(DH/\left((S(B^*)^{e_1})^x\right)\right)^{d^{-1}}, & \pi^* \text{ is an initiator session} \end{cases}$$

     where $d := h_0(A^*, X)$ and $e_1 := h_3(B, S)$. If π^* is an initiator session, then we have its ephemeral key x. If π^* is a responder session, then we have the semi-static key s of the responder $\mathtt{B}^*$. Once such a DH' value is found, $\mathcal{B}_{\mathsf{ltlt}}$ aborts the simulation and returns DH' as the CRGapDH solution.
  2. Otherwise, simulate by lazy sampling.

To bound $\Pr[\mathtt{QueryDH}_{\mathsf{ltlt}}]$, we first prove that $\mathcal{B}_{\mathsf{ltlt}}$ can extract a correct CRGapDH solution from KDF_f and KDF_r (specifically, when the DDH oracle outputs 1 and the query matches $\pi^*.ctxt$). In KDF_f, if π^* is an initiator session, then the extracted DH value DH' equals $(Y(B^*)^{e_1}S^{e_2})^{a^*}$. By the simulation of $h_1(B^*, Y, S)$, we have $e_2 = h_1(B^*, Y, S) = h$ where $h \leftarrow \text{CH}(YS^{e_2})$. Therefore, we have

$$DH' = (Y(B^*)^{e_1}S^{e_2})^{a^*} = ((YS^{e_2}) \cdot (B^*)^h)^{a^*},$$

where $h = \text{CH}(YS^{e_2})$. So, DH' is a solution to CRGapDH. A similar argument applies to KDF_r and the case that π^* is a responder session. Therefore, as long as $\mathcal{B}_{\mathsf{ltlt}}$ aborts the game when simulating KDF_f or KDF_r, the extracted solution is a correct CRGapDH solution.

By definition, if $\mathtt{QueryDH}_{\mathsf{ltlt}}$ happens, then $\mathcal{A}$ must have queried $\mathsf{KDF}_f(DH, A^*, B^*, S, Y, X)$ or $\mathsf{KDF}_r(DH, A^*, B^*, S, X)$ which makes the DDH oracle output 1,

and $\mathcal{B}_{\mathsf{ltlt}}$ will extract a CRGapDH solution. Therefore, if $\mathtt{QueryDH}_{\mathsf{ltlt}}$ happens, then $\mathcal{B}_{\mathsf{ltlt}}$ solves its CRGapDH challenge, and thus we have

$$\begin{aligned}\mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) &\leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \Pr\left[\mathtt{QueryDH}_{\mathsf{ltlt}}\right] \\ &\leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{B}_{\mathsf{ltlt}}) + Q_{\mathrm{RO}}/p,\end{aligned}$$

where the last term counts in the probability that there exists a non-invertible h value from the CH oracle.

Now we can bound the adversary's advantage in $\mathcal{G}_4^{\mathsf{ltlt}}$. In this game, the session keys of all sessions are computed via lazy sampling. These keys are distributed uniformly at random in $\mathcal{A}$'s view until it queries KDF_f or KDF_r on the correct hash input. By the abort event $\mathtt{QueryDH}_{\mathsf{ltlt}}$, the adversary cannot query the correct hash input of the test session, so the session key of π^* is distributed uniformly at random in $\mathcal{A}$'s view, independent of the game's challenge bit b_{test}. Therefore, the probability that $b_{\mathsf{guess}} = b_{\mathsf{test}}$ is $\frac{1}{2}$, and we have

$$\mathsf{Adv}^{\mathcal{G}_4^{\mathsf{ltlt}}}_{\mathsf{XHMQV}}(\mathcal{A}) = 0.$$

We assume that the ϵ-CRGapDH assumption holds over $(\mathbb{G}, p, g)$. Combining all the probability bounds, we have the final bound for the case $\mathsf{clean}_{\mathsf{LTLT}}(\pi^*)$:

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTLT}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u^2 \cdot (2\delta + \epsilon + Q_{\mathrm{RO}}/p)$$

A.2 Case $\mathsf{clean}_{\mathsf{LTSS}}$

By the definition of $\mathsf{clean}_{\mathsf{LTSS}}$, we have (1) the initiator of π^* is uncompromised, (2) the semi-static key of π^*'s responder is uncompromised, and (3) if π^* is an initiator session, then either the responder's long-term key was uncompromised upon acceptance, or the semi-static key used in π^* is generated by the security game (rather than generated by the adversary).

By $\mathbf{G}_2$, we always have $\pi^*.sspk \in \boldsymbol{sspk}_{\pi^*.\mathsf{pid}}$ as long as $\mathsf{clean}_{\mathsf{LTSS}}(\pi^*)$. This is because, on the one hand, if $\mathsf{clean}_{\mathsf{LTSS}}(\pi^*)$ and $\pi^*.\mathsf{pcorr} = 0$, then we have $\pi^*.sspk \in \boldsymbol{sspk}$ unless $\mathcal{A}$ forged a valid signature on the semi-static key of an uncompromised party, which has been excluded in $\mathbf{G}_2$. On the other hand, by definition, we have $\mathsf{clean}_{\mathsf{LTSS}}(\pi^*) \wedge \pi^*.\mathsf{pcorr} \implies \pi^*.sspk \in \boldsymbol{sspk}_{\pi^*.\mathsf{pid}}$. Therefore, the third condition described above and the modification in $\mathbf{G}_2$ ensure that the semi-static key used in π^* has been generated by the game. This allows us to embed a CRGapDH challenge into the semi-static key.

We begin with $\mathcal{G}_0^{\mathsf{ltss}} := \mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTSS}}(\pi^*)$, so we have

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTSS}}(\pi^*)}_{\mathsf{XHMQV}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}_0^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

We use the game sequence $\mathcal{G}_1^{\mathsf{ltss}}$-$\mathcal{G}_4^{\mathsf{ltss}}$ to bound the winning advantage of $\mathcal{A}$ when π^* satisfies $\mathsf{clean}_{\mathsf{LTSS}}$. Similar to the proof of the case $\mathsf{clean}_{\mathsf{LTLT}}(\pi^*)$, we use game transitions $\mathcal{G}_1^{\mathsf{ltss}}$-$\mathcal{G}_3^{\mathsf{ltss}}$ and the simulatability (cf. Definition 3) of the

signature scheme to simulate signatures of the initiator of π^* without using its long-term key. Then, we use $\mathcal{G}_3^{\mathsf{ltss}}$-$\mathcal{G}_4^{\mathsf{ltss}}$ and $\mathsf{CRGapDH}$ to prove that the adversary cannot query KDF_f (or KDF_r) on the correct hash input of π^*, and thus it cannot distinguish whether π^*'s key is real or random.

GAME $\mathcal{G}_1^{\mathsf{ltss}}$: We guess the identity $\mathtt{A}^*$ of the initiator involved in the test session. If this guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote $\mathtt{A}^*$'s long-term key pair as (a^*, A^*). We have

$$\mathsf{Adv}^{\mathcal{G}_0^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u \cdot \mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_2^{\mathsf{ltss}}$: We guess the identity $\mathtt{B}^*$ of the responder involved in π^* and the identifier ssid^* of the repsonder $\mathtt{B}^*$'s uncorrupted semi-static key. The semi-static key pair of π^* is denoted as (s^*, S^*) (i.e., $\pi^*.\mathsf{ssid} = \mathsf{ssid}^*$ and $(s^*, S^*) = (\boldsymbol{sssk}_{\mathtt{B}^*}[\mathsf{ssid}^*], \boldsymbol{sspk}_{\mathtt{B}^*}[\mathsf{ssid}^*])$). Since there are at most n_u users involved in this game and at most n_{ss} semi-static keys for each user, we have

$$\mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u n_{ss} \cdot \mathsf{Adv}^{\mathcal{G}_2^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_3^{\mathsf{ltss}}$: In this game, we use the RO-simulatability of Sig to simulate $\mathtt{A}^*$'s signatures without using a^*. Similar to the case of $\mathsf{clean}_{\mathsf{LTLT}}(\pi^*) = 1$, we can use the δ-RO-simulatability of Sig to bound the probability difference, so we have

$$\mathsf{Adv}^{\mathcal{G}_2^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \delta$$

GAME $\mathcal{G}_4^{\mathsf{ltss}}$: We introduce an abort event $\mathtt{QueryDH}_{\mathsf{ltss}}$ in $\mathcal{G}_4^{\mathsf{ltss}}$: Specifically, if

- $\mathcal{A}$ queries KDF_f on (DH, A^*, B, S^*, Y, X) , where $d = h_0(A^*, X), e_1 = h_1(B, Y, S^*), e_2 = h_2(B, Y, S^*)$, $\textsc{Ddh}(X(A^*)^d, YB^{e_1}(S^*)^{e_2}, DH) = 1$, and $(A^*, B, S^*, Y, X) = \pi^*.ctxt$,
- $\mathcal{A}$ queries KDF_r on (DH, A^*, B, S^*, X), where $d = h_0(A^*, X)$, $e_1 = h_3(B, S^*)$, and $\textsc{Ddh}(X(A^*)^d, S^*B^{e_1}, DH) = 1$ and $(A^*, B, S^*, X) = \pi^*.ctxt$,

then the game aborts. In other words, we abort the game if $\mathcal{A}$ queries KDF_f or KDF_r on any DH value that contains the information of $(A^*)^{s^*} (= (S^*)^{a^*})$. We construct a $\mathsf{CRGapDH}$ adversary $\mathcal{B}_{\mathsf{ltss}}$ to bound the probability of $\mathtt{QueryDH}_{\mathsf{ltss}}$.

Reduction $\mathcal{B}_{\mathsf{ltss}}^{\textsc{Ddh},\textsc{Ch}}(A^*, S^*)$:

- Initialize $\mathcal{G}_4^{\mathsf{ltss}}$:
  1. Choose two users $\mathtt{A}^*$ and $\mathtt{B}^*$ uniformly at random. Pick $\mathsf{ssid}^* \leftarrow\!\!\$\ [n_{ss}]$. The long-term public key of $\mathtt{A}^*$ is set as A^*.
  2. Initialize lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ for the simulation of session keys using DDH and random oracle patching (as in $\mathbf{G}_2$). We also initialize $\mathcal{L}_{\mathtt{sim}}$ to simulate the signatures of $\mathtt{A}^*$'s and $\mathtt{B}^*$'s semi-static keys (as we did in Fig. 10).

3. For party $\mathtt{A}^*$, run the modified KGenSS algorithm defined in Fig. 10. After this step, $\mathcal{B}_{\mathsf{ltss}}$ has $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$ (except for the long-term secret key of $\mathtt{A}^*$). $\mathcal{B}_{\mathsf{ltss}}$ further replaces $\boldsymbol{sspk}_{\mathtt{B}^*}[\mathsf{ssid}^*]$ with S^*. For all other parties, follow the KGenLT and KGenSS defined in XHMQV faithfully.
4. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceeds the game as in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$. If the initiator and responder of the test session are not $\mathtt{A}^*$ and $\mathtt{B}^*$ respectively, then overwrite $\mathcal{A}$'s bit with 0.

The random oracles and game oracles are simulated as described below.

– Simulation of oracles defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$:
  - All oracles except Send, CorruptLTKey, and CorruptSSKey are simulated in the same way defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$.
  - By the definition of $\mathsf{clean}_{\mathsf{LTSS}}$, $\mathcal{A}$ does not query CorruptLTKey on $\mathtt{A}^*$, nor to CorruptSSKey on $(\mathtt{B}^*, \mathsf{ssid}^*)$. Queries to CorruptLTKey and CorruptSSKey on other parties and sessions are simulated as usual.
  - For a Send query, if the query does not involve either $\mathtt{A}^*$ or $\mathtt{B}^*$ with ssid^*, then we also simulate the oracle in the same way as in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$. If the query involves $\mathtt{A}^*$ or $\mathtt{B}^*$ with ssid^*, then we simulate in the same ways as in $\mathcal{B}_{\mathsf{ltlt}}$. Namely, we use $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ to make sure that we can always create session keys that are consistent with the random oracles.

– Simulation of Random Oracles:
  - For fresh RO queries to KDF_f and KDF_r that do not match π^*, we simulate in the same way as in $\mathbf{G}_2$. Similarly, for fresh RO queries to h_0, h_1, h_2, and h_3 that do not involve either A^* or S^*, we simulate as usual.
  - $h_0(A^*, X)$: $h \leftarrow \textsc{Ch}(X)$, $h_0(A^*, X) := h$.
  - $h_2(B, Y, S^*)$: $h \leftarrow \textsc{Ch}(Y)$, $h_0(B, Y, S^*) := h$.
  - h_1 and h_3: Simulate as usual.
  - $\mathsf{KDF}_f(DH, A^*, B^*, S, Y, X)$:
    1. If $(A^*, B^*, S^*, Y, X) = \pi^*.ctxt$ and $\textsc{Ddh}(X(A^*)^d, Y(B^*)^{e_1}(S^*)^{e_2}, DH) = 1$, then extract the value
    $$DH' := \begin{cases} \left(DH/(X(A^*)^d)^{y+b^*e_1}\right)^{e_2^{-1}}, & \pi^* \text{ is a responder session} \\ \left(DH/\left(Y(B^*)^{e_1}(S^*)^{e_2}\right)^x \cdot A^{de_1b}\right)^{d^{-1}}, & \pi^* \text{ is an initiator session} \end{cases}$$
    where $d := h_0(A^*, X), e_1 := h_1(B^*, Y, S^*)$, and $e_2 := h_2(B^*, Y, S^*)$. In this game, we have the long-term keys of all parties except $\mathtt{A}^*$, so we have the long-term key b^* of the party $\mathtt{B}^*$. If π^* is an initiator session, then we have its ephemeral key x. If π^* is a responder session, then we have the ephemeral key y. Once such a DH' is found, $\mathcal{B}_{\mathsf{ltss}}$ aborts the simulation and return DH' as the $\mathsf{CRGapDH}$ solution.
    2. Otherwise, simulate by lazy sampling.
  - $\mathsf{KDF}_r(DH, A^*, B^*, S, X)$:
    1. If $(A^*, B^*, S^*, X) = \pi^*.ctxt$ and $\textsc{Ddh}(X(A^*)^d, S^*(B^*)^{e_1}, DH) = 1$, then extract the value
    $$DH' := \begin{cases} DH/(X(A^*)^d)^{be_1}, & \pi^* \text{ is a responder session} \\ \left(DH/\left(S^*(B^*)^{e_1}\right)^x \cdot A^{de_1b}\right)^{d^{-1}}, & \pi^* \text{ is an initiator session} \end{cases}$$

where $d := h_0(A^*, X)$ and $e_1 := h_3(B^*, Y, S^*)$. In this game, we have the long-term key b^* of $\mathtt{B}^*$. If π^* is a responder session, then we have the ephemeral key y, and thus $\mathcal{B}_{\mathsf{ltss}}$ can calculate DH' and return DH' as the CRGapDH solution.
If π^* is an initiator session, then $DH' = \mathsf{DH}(A^*, S^*)$. However, DH' itself does not align with the format of a CRGapDH solution. To deal with it, $\mathcal{B}_{\mathsf{ltss}}$ first pick a random $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $R := g^r$, and queries $\textsc{Ch}(R)$ to obtain h. Then $\mathcal{B}_{\mathsf{ltss}}$ computes $DH'' := (S^*)^r \cdot (DH')^h$ and outputs DH'' as the CRGapDH solution. One can verify that $DH'' = (R(A^*)^h)^{s^*}$, where $h = \textsc{Ch}(R)$, which means that it is a valid CRGapDH solution.

2. Otherwise, simulate by lazy sampling.

Here we only prove that the simulation of KDF_f and KDF_r can extract the CRGapDH solution. In KDF_f, if π^* is an initiator session, then the extracted DH value DH' equals to $(Y(S^*)^{e_2})^{a^*}$, where $e_2 = \textsc{Ch}(B^*, Y, S^*) = h$ and h is from $\textsc{Ch}(Y)$, so DH' is a valid solution in this case. If π^* is a responder session, then $DH' = (X(A^*)^d)^{s^*}$ and d is from $\textsc{Ch}(X)$, and thus it is also a valid CRGapDH solution. A similar argument applies to KDF_r. Therefore, we have

$$\mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{B}_{\mathsf{ltss}}) + Q_{\mathrm{RO}}/p$$

The remain argument to bound $\Pr[\mathtt{QueryDH}_{\mathsf{ltss}}]$ is the same as in the case of $\mathsf{clean}_{\mathsf{LTLT}}$, so we have

$$\mathsf{Adv}^{\mathcal{G}_4^{\mathsf{ltss}}}_{\mathsf{XHMQV}}(\mathcal{A}) = 0.$$

We assume that the ϵ-CRGapDH assumption holds over $(\mathbb{G}, p, g)$. Combining all the probability bounds gives us the final bound of the case $\mathsf{clean}_{\mathsf{LTSS}}$:

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTSS}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq {n_u}^2 n_{ss} \cdot (\delta + \epsilon + Q_{\mathrm{RO}}/p)$$

A.3 Case $\mathsf{clean}_{\mathsf{LTE}}(\pi^*)$

In this case, π^* performs a full handshake as the ephemeral secret of the responder is involved. By definition of $\mathsf{clean}_{\mathsf{LTE}}$, the initiator's long-term key is uncompromised and the responder's session randomness is not revealed. We begin with $\mathcal{G}_0^{\mathsf{lte}} := \mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTE}}(\pi^*)$, so we have

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTE}}(\pi^*)}_{\mathsf{XHMQV}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}_0^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

We use game sequence $\mathcal{G}_1^{\mathsf{lte}}$-$\mathcal{G}_4^{\mathsf{lte}}$ to bound the winning advantage of $\mathcal{A}$ when π^* satisfies $\mathsf{clean}_{\mathsf{LTE}}$. Similar to the proof of the case $\mathsf{clean}_{\mathsf{LTLT}}(\pi^*)$, we use game transitions $\mathcal{G}_1^{\mathsf{lte}}$-$\mathcal{G}_3^{\mathsf{lte}}$ and the simulatability (cf. Definition 3) of the signature scheme to simulate signatures of the initiator of π^* without using its long-term key. Then, we use $\mathcal{G}_3^{\mathsf{lte}}$-$\mathcal{G}_4^{\mathsf{lte}}$ and the CRGapDH assumption to prove that the adversary cannot query KDF_f (or KDF_r) on the correct hash input of π^*, and thus

cannot distinguish whether π^*'s key is real or random.

GAME $\mathcal{G}_1^{\mathsf{lte}}$: We guess the responder session π_r^* involved in the test session π^* (i.e., either $\pi^* = \pi_r^*$ or $\pi^*.\mathsf{cid} = \pi_r^*.\mathsf{cid}$). If the guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote π_r^*'s ephemeral key pair as (y^*, Y^*).

Specifically, we guess π_r^* in the following way (the code will be provided in the reduction described later): We first initialize a counter $\mathsf{cnt} := 0$ and choose $\mathsf{cnt}^* \leftarrow\!\!\$\, [n_s]$. When $\mathcal{A}$ issues $\textsc{Send}(\mathtt{U}, i, m)$ queries with $m = (\mathsf{create}, *, *)$ (i.e., initializes a responder session), then cnt increments. If $\mathsf{cnt} = \mathsf{cnt}^*$, then we set $\pi_r^* := \pi_{\mathtt{U}}^i$.

In $\mathcal{G}_1^{\mathsf{lte}}$, there are at most n_s sesssions during $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$. Since cnt^* is independent of $\mathcal{A}$'s view, the probability that cnt^*-th responder session (regardless of the owner) is involved in π^* is $\frac{1}{n_s}$, and thus we have

$$\mathsf{Adv}^{\mathcal{G}_0^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) = n_s \cdot \mathsf{Adv}^{\mathcal{G}_1^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_2^{\mathsf{lte}}$: We guess the identity $\mathtt{A}^*$ of the initiator involved in the test session. If this guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote $\mathtt{A}^*$'s long-term key pair as (a^*, A^*). Since there are at most n_u users involved in this game and the guess is uniformly at random in $\mathcal{A}$'s view, we have

$$\mathsf{Adv}^{\mathcal{G}_1^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u \cdot \mathsf{Adv}^{\mathcal{G}_2^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_3^{\mathsf{lte}}$: In this game, we use the RO-simulatability of Sig to simulate $\mathtt{A}^*$'s signatures only using $\mathtt{A}^*$'s long-term public key A^*. Similar to $\mathcal{G}_2^{\mathsf{ltlt}}$ and $\mathcal{G}_3^{\mathsf{ltlt}}$ in the case of $\mathsf{clean}_{\mathsf{LTLT}}(\pi^*)$, we can use the δ-RO-simulatability of Sig to bound the probability difference.

$$\mathsf{Adv}^{\mathcal{G}_2^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_3^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \delta$$

GAME $\mathcal{G}_4^{\mathsf{lte}}$: We introduce an abort event $\mathtt{QueryDH}_{\mathsf{lte}}$ in $\mathcal{G}_4^{\mathsf{lte}}$: Specifically, if

- $\mathcal{A}$ queries KDF_f on (DH, A^*, B, S, Y^*, X) , where $d = h_0(A^*, X), e_1 = h_1(B, Y^*, S), e_2 = h_2(B, Y^*, S)$, $\textsc{Ddh}(X(A^*)^d, Y^* B^{e_1} S^{e_2}, DH) = 1$, and $(A^*, B, S, Y^*, X) = \pi^*.ctxt$,

then the game aborts (i.e., return 0 as $\mathcal{A}$'s bit guess). In other words, we abort the game if $\mathcal{A}$ queries KDF_f or KDF_r on any DH value that contains the information of $(A^*)^{y^*} (= (Y^*)^{a^*})$. We construct a $\mathsf{CRGapDH}$ adversary $\mathcal{B}_{\mathsf{lte}}$ such that if such an abort event happens, then $\mathcal{B}_{\mathsf{lte}}$ can output a solution to its $\mathsf{CRGapDH}$ challenge.

Reduction $\mathcal{B}_{\mathsf{lte}}^{\textsc{Ddh},\textsc{Ch}}(A^*, Y^*)$:

- Initialize $\mathcal{G}_4^{\mathsf{lte}}$:
  1. Choose a party $\mathtt{A}^*$ uniformly at random and set its long-term public key as A^*.

2. Initialize a counter $\mathsf{cnt} := 0$ and choose $\mathsf{cnt}^* \leftarrow\!\!\$\ [n_s]$. Initialize $\pi_r^* := \perp$ and set it up later.
3. Initialize lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$, and $\mathcal{L}_{\mathtt{sim}}$.
4. For party $\mathtt{A}^*$, we use its long-term public key, the algorithm $\mathsf{SIMsign}$, and $\mathcal{L}_{\mathtt{sim}}$ to simulate the signatures of its semi-static keys (as we did in Fig. 10). For all other parties, follow the KGenLT and KGenSS defined in XHMQV to generate $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$ faithfully.
5. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceed the game as in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$. If the initiator of π^* is not $\mathtt{A}^*$, π_r^* is not π^* itself, or π_r^* is not the contributive partner session of π^*, then overwrite $\mathcal{A}$'s bit guess with 0.

The random oracles and game oracles are simulated as described below.

– Simulation of oracles defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$:
  - All oracles except Send, CorruptLTKey, and RevealRand are simulated in the same way defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$.
  - If CorruptLTKey($\mathtt{A}^*$) is queried, then abort the game and overwrite $\mathcal{A}$'s bit guess with 0. By definition of $\mathsf{clean}_{\mathsf{LTE}}$, if the guess is correct, then $\mathcal{A}$ will not corrupt $\mathtt{A}^*$. CorruptLTKey queries on other parties are simulated as usual.
  - If RevealRand($\mathtt{U}, i$) is queried, where $\pi_i^{\mathtt{U}} = \pi_r^* \neq \perp$, then abort the game and overwrite $\mathcal{A}$'s bit with 0. By definition of $\mathsf{clean}_{\mathsf{LTE}}$, if the guess is correct, then $\mathcal{A}$ will not reveal the session randomness of π_r^*. RevealRand queries on other sessions are simulated as usual.
  - Send($\mathtt{U} \neq \mathtt{A}^*, i, m$):
    * Case $m = (\mathsf{create}, \mathsf{ssid}, \mathsf{type})$ (as the session responder): $\mathsf{cnt} := \mathsf{cnt} + 1$. If $\mathsf{cnt} = \mathsf{cnt}^*$ and $\mathsf{type} = \mathsf{full}$, then we use Y^* as the ephemeral public key of this session and set $\pi_r^* := \pi_i^{\mathtt{U}}$. Otherwise, simulate as usual.
    * For other cases, simulate as usual.
  - Send($\mathtt{A}^*, i, m$) (We only present how to set up the session keys):
    * Case $m = (\mathsf{create}, \mathsf{ssid}, \mathsf{type})$ (as the session responder): Simulate as usual. If $\mathsf{type} = \mathsf{full}$, then $\mathsf{cnt} := \mathsf{cnt} + 1$.
    * Case $m = (\mathtt{U}, X)$ (as the session responder): We apply the same approach as in case $\mathsf{clean}_{\mathsf{LTLT}}$ for $\mathcal{B}_{\mathsf{ltlt}}$. Namely, we use the lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ and oracle patching for KDF_f and KDF_r to simulate the session keys and the RO outputs of KDF_f and KDF_r consistently.
    * Case $m = (\mathtt{B}, S, Y, \sigma)$ (as the session initiator): Simulate analogously to the case $m = (\mathtt{U}, X)$ (no matter whether $Y = Y^*$).

– Simulation of Random Oracles:
  - For fresh RO queries to $\mathsf{KDF}_r(DH, A, B, S, X)$ and $\mathsf{KDF}_f(DH, A, B, S, Y, X)$, where $(A, Y) \neq (A^*, Y^*)$, we simulate in the same way as in $\mathbf{G}_2$. Similarly, for fresh RO queries to h_0, h_1, h_2, and h_3 that do not involve $\mathtt{A}^*$ and Y^*, we simulate as usual.
  - $h_0(A^*, X)$: $h \leftarrow \mathrm{CH}(X)$, $h_0(A, X) := h$.
  - $h_1(B, Y^*, S)$: Simulate as usual.
  - $h_2(B, Y^*, S)$: $h \leftarrow \mathrm{CH}(S)$, $h_1(B, Y^*, S) := h^{-1}$
  - $\mathsf{KDF}_f(DH, A^*, B, S, Y^*, X)$:

1. If $\textsc{Ddh}(X(A^*)^d, Y^*B^{e_1}S^{e_2}, DH)$ outputs 1, then extract the value

$$DH' := \begin{cases} DH/(X(A^*)^d)^{be_1+se_2}, & \pi^* \text{ is a responder session} \\ \left(DH/\left(Y^*B^{e_1}S^{e_2}\right)^x \cdot (A^*)^{dbe_1}\right)^{d^{-1}{e_2}^{-1}}, & \pi^* \text{ is an initiator session} \end{cases}$$

   where $d := h_0(A^*, X), e_1 := h_1(B, Y, S)$, and $e_2 := h_2(B, Y, S)$. If π^* is an initiator session, then we have its ephemeral key x. If π^* is a responder session, then we have the long-term key b of the responder $\mathtt{B}$ and the semi-static key s. $\mathcal{B}_{\mathsf{lte}}$ aborts the simulation and return DH' as the $\mathsf{CRGapDH}$ solution.
2. Otherwise, simulate by lazy sampling.

We first prove that the simulation of KDF_f can successfully extract the $\mathsf{CRGapDH}$ solution. In KDF_f, if π^* is an initiator session, then the extracted DH value DH' equals to $(Y^*S^{e_2})^{a^*{e_2}^{-1}} = ((Y^*)^{{e_2}^{-1}}S)^{a^*}$. By the simulation of $h_2(B^*, Y, S)$, we have $e_2 = h_2(B, Y^*, S) = h^{-1}$ where $h \leftarrow \textsc{Ch}(S)$. Therefore, we have

$$DH' = ((Y^*)^{{e_2}^{-1}}S)^{a^*} = (S(Y^*)^h)^{a^*},$$

where $h = \textsc{Ch}(S)$. So, DH' is a solution to $\mathsf{CRGapDH}$. Similarly, if π^* is a responder session, then we have

$$DH' = (X(A^*)^d)^{y^*},$$

where $d = \textsc{Ch}(X)$. This is also a valid solution to $\mathsf{CRGapDH}$.

It is straight-forward to see that if $\mathtt{QueryDH}_{\mathsf{lte}}$ occurs, then $\mathcal{B}_{\mathsf{lte}}$ will extract a $\mathsf{CRGapDH}$ solution from KDF_f. So, we have

$$\begin{aligned} \mathsf{Adv}^{\mathcal{G}_3^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) &\leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \Pr\left[\mathtt{QueryDH}_{\mathsf{lte}}\right] \\ &\leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{B}_{\mathsf{lte}}) + Q_{\mathrm{RO}}/p, \end{aligned}$$

where the last term counts in the probability that there exists a non-invertible h value from the $\textsc{Ch}$ oracle. Finally, we use a similar argument as for bounding $\mathcal{G}_4^{\mathsf{ltlt}}$ to bound $\mathcal{A}$'s advantage in $\mathcal{G}_4^{\mathsf{lte}}$. That is, if $\mathcal{A}$ cannot trigger $\mathtt{QueryDH}_{\mathsf{lte}}$, then the session key of π^* is distributed uniformly at random in $\mathcal{A}$'s view. Therefore, we have

$$\mathsf{Adv}^{\mathcal{G}_4^{\mathsf{lte}}}_{\mathsf{XHMQV}}(\mathcal{A}) = 0.$$

We assume that the ϵ-$\mathsf{CRGapDH}$ assumption holds over $(\mathbb{G}, p, g)$. Combining all the probability bounds gives us the final bound of the case $\mathsf{clean}_{\mathsf{LTE}}(\pi^*)$:

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{LTE}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u n_s \cdot (\delta + \epsilon + Q_{\mathrm{RO}}/p)$$

A.4 Case $\mathsf{clean}_{\mathsf{ELT}}$

In this case, $\mathsf{clean}_{\mathsf{ELT}}$ ensures for π^* that (1) the initiator randomness involved in π^* is not revealed, (2) the long-term key of the responder of π^* is uncompromised, and (3) if π^* is a responder session, then there exists a partnered initiator session. Condition (3) ensures that the initiator ephemeral public key of π^* is always controlled by the game (so we can embed the $\mathsf{CRGapDH}$ challenge). Since $\mathsf{clean}_{\mathsf{ELT}}$ is similar to $\mathsf{clean}_{\mathsf{LTE}}$, we may reuse some arguments in proving $\mathsf{clean}_{\mathsf{LTE}}$ (cf. Appendix A.3). We begin with $\mathcal{G}_0^{\mathsf{elt}} := \mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{ELT}}$, so we have

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{ELT}}}_{\mathsf{XHMQV}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}_0^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_1^{\mathsf{elt}}$: We guess the initiator session π_i^* involved in the test session π^* (i.e., either $\pi^* = \pi_i^*$ or $\pi^*.\mathsf{sid} = \pi_i^*.\mathsf{sid}$). If the guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote π_i^*'s ephemeral key pair as (x^*, X^*).

We apply the same guessing approach used in proving $\mathcal{G}_1^{\mathsf{lte}}$ to this proof case. Roughly, we use a counter cnt which increments each time $\mathcal{A}$ queries SEND to initialize an initiator session. And we guess $\mathsf{cnt}^* \leftarrow_{\$} [n_s]$ and set $\pi_i^* := \pi_{\mathtt{U}}^i$ if SEND$(\mathtt{U}, i, m)$ is the cnt^*-th SEND query for initializing an initaitor session. We have

$$\mathsf{Adv}^{\mathcal{G}_0^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) = n_s \cdot \mathsf{Adv}^{\mathcal{G}_1^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_2^{\mathsf{elt}}$: We guess the identity $\mathtt{B}^*$ of the responder involved in the test session. If this guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote $\mathtt{B}^*$'s long-term key pair as (b^*, B^*). Since there are at most n_u users involved in this game and the guess is uniformly at random in $\mathcal{A}$'s view, we have

$$\mathsf{Adv}^{\mathcal{G}_1^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u \cdot \mathsf{Adv}^{\mathcal{G}_2^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_3^{\mathsf{elt}}$: In this game, we use the RO-simulatability of Sig to simulate $\mathtt{B}^*$'s signatures without using b^*. Similar to previous proof cases that involve uncompromised long-term secrets, we can use the δ-RO-simulatability of Sig to bound the probability difference.

$$\mathsf{Adv}^{\mathcal{G}_2^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_3^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \delta$$

GAME $\mathcal{G}_4^{\mathsf{elt}}$: We introduce an abort event $\mathtt{QueryDH}_{\mathsf{elt}}$ in $\mathcal{G}_4^{\mathsf{elt}}$: Specifically, if

- $\mathcal{A}$ queries KDF_f on (DH, A, B^*, S, Y, X^*) , where $d = h_0(A, X^*), e_1 = h_1(B, Y^*, S), e_2 = h_2(B^*, Y, S)$, DDH$(X^*A^d, Y(B^*)^{e_1}S^{e_2}, DH) = 1$, and $(A, B^*, S, Y, X^*) = \pi^*.ctxt$, or
- $\mathcal{A}$ queries KDF_r on (DH, A, B^*, S, X^*), where $d = h_0(A, X^*), e_1 = h_3(B^*, S)$, and DDH$(X^*(A)^d, S(B^*)^{e_1}, DH) = 1$ and $(A, B^*, S, X^*) = \pi^*.ctxt$,

then the game aborts (i.e., return 0 as $\mathcal{A}$'s bit guess). In other words, we abort the game if $\mathcal{A}$ queries KDF_f or KDF_r on any DH value that contains the information of $(B^*)^{x^*} (= (X^*)^{b^*})$. We construct a $\mathsf{CRGapDH}$ adversary $\mathcal{B}_{\mathsf{elt}}$ such that

if such an abort event happens, then $\mathcal{B}_{\mathsf{elt}}$ can output a solution to its $\mathsf{CRGapDH}$ challenge.

Reduction $\mathcal{B}_{\mathsf{elt}}^{\text{DDH,CH}}(X^*, B^*)$:

- Initialize $\mathcal{G}_4^{\mathsf{elt}}$:
  1. Choose a party $\mathtt{B}^*$ uniformly at random and set the long-term public key of $\mathtt{B}^*$ as B^*.
  2. Initialize a counter $\mathsf{cnt} := 0$ and choose $\mathsf{cnt}^* \leftarrow\!\!\$\ [n_s]$. Initialize $\pi_i^* := \bot$ and set up it later.
  3. Initialize lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$, and $\mathcal{L}_{\mathtt{sim}}$.
  4. For party $\mathtt{B}^*$, we use B^*, the algorithm $\mathsf{SIMsign}$, and $\mathcal{L}_{\mathtt{sim}}$ to simulate the signatures of its semi-static keys (as we did in Fig. 10). For all other parties, follow the KGenLT and KGenSS defined in XHMQV to generate $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$ faithfully.
  5. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceeds the game as in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$. If the responder of π^* is not $\mathtt{B}^*$, π_i^* is not π^*, or π_i^* is not the matching session of π^*, then overwrite $\mathcal{A}$'s bit with 0.

  The random oracles and game oracles are simulated as described below.
- Simulation of oracles defined in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$:
  - All oracles except Send, CorruptLTKey, and RevealRand are simulated in the same way defined in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$.
  - If CorruptLTKey($\mathtt{B}^*$) is queried, then abort the game and overwrite $\mathcal{A}$'s bit with 0. By definition of $\mathsf{clean}_{\mathsf{ELT}}$, if the guess is correct, then $\mathcal{A}$ will not corrupt $\mathtt{B}^*$. CorruptLTKey queries on other parties are simulated as usual.
  - If RevealRand($\mathtt{U}, j$) is queried, where $\pi_j^{\mathtt{U}} = \pi_i^* \neq \bot$, then abort the game and overwrite $\mathcal{A}$'s bit with 0. By definition of $\mathsf{clean}_{\mathsf{ELT}}$, if the guess is correct, then $\mathcal{A}$ will not reveal the session randomness of π_i^*. RevealRand queries on other sessions are simulated as usual.
  - Send($\mathtt{U} \neq \mathtt{B}^*, j, m$):
    - Case $m = (\mathtt{B}^*, sspk, epk)$ (initiator session with intended partner $\mathtt{B}^*$): $\mathsf{cnt} := \mathsf{cnt} + 1$. If $\mathsf{cnt} = \mathsf{cnt}^*$, then we use X^* as the ephemeral public key of this session and set $\pi_i^* := \pi_j^{\mathtt{U}}$. Otherwise, simulate as usual.
    - For other cases, simulate as usual.
  - Send($\mathtt{B}^*, j, m$): We apply the same approach used in $\mathcal{B}_{\mathsf{ltlt}}$ to this case. Namely, we use the lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ and oralce patching of KDF_f and KDF_r to simualte the session keys and the RO outputs of KDF_f and KDF_r consistently.
- Simulation of Random Oracles:
  - For fresh RO queries to $\mathsf{KDF}_r(DH, A, B, S, X)$ and $\mathsf{KDF}_f(DH, A, B, S, Y, X)$, where $(B, X) \neq (B^*, X^*)$, we simulate in the same way as in $\mathbf{G}_2$. Similarly, for fresh RO queries to h_0, h_1, h_2, and h_3 that do not involve $\mathtt{B}^*$ and X^*, we simulate as usual.
  - $h_1(B^*, Y, S)$: Compute $e_2 := h_2(B^*, Y, S), R := YS^{e_2}$, and queries $\text{CH}(R)$ and gets h. Set $h_1(B^*, Y, S) := h$.

- $h_3(B^*, S)$: $h \leftarrow \textsc{Ch}(S), h_3(B^*, S) := h$.
- $h_0(A, X^*)$ and $h_2(B, Y^*, S)$: Simulate as usual.
- $\mathsf{KDF}_f(DH, A, B^*, S, Y, X^*)$:
  1. If $(A, B^*, S, Y, X^*) = \pi^*.ctxt$ and $\textsc{Ddh}(X^*A^d, Y(B^*)^{e_1}S^{e_2}, DH)$ outputs 1, then extract the value

$$DH' := \begin{cases} \left(DH/(X^*A^d)^{y+se_2} \cdot (B^*)^{e_1 ad}\right)^{{e_1}^{-1}}, & \pi^* \text{ is a responder session} \\ DH/(Y(B^*)^{e_1}S^{e_2})^{ad}, & \pi^* \text{ is an initiator session} \end{cases}$$

     where $d := h_0(A, X^*), e_1 := h_1(B^*, Y, S)$, and $e_2 := h_2(B^*, Y, S)$. We have a since we have long-term keys of all parties except $\mathtt{B}^*$. If π^* is an initiator session, then $\mathcal{B}_{\mathsf{lte}}$ aborts the simulation and return DH' as the $\mathsf{CRGapDH}$ solution. If π^* is a responder session (so we have the ephemeral key y and the semi-static key s), then we have that $DH' = (X^*)^{b^*}$. Similar to $\mathcal{B}_{\mathsf{ltss}}$, in this case, $\mathcal{B}_{\mathsf{elt}}$ first pick a $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute $R := g^r$, and queries $\textsc{Ch}(R)$ and gets h. Then $\mathcal{B}_{\mathsf{ltss}}$ computes $DH'' := (S^*)^r \cdot (DH')^h$ and outputs DH' as the $\mathsf{CRGapDH}$ solution.
  2. Otherwise, simulate by lazy sampling.
- $\mathsf{KDF}_r(DH, A, B^*, S, X^*)$:
  1. If $(A, B^*, S, X^*) = \pi^*.ctxt$ and $\textsc{Ddh}(X^*A^d, S(B^*)^{e_1}, DH)$ outputs 1, then extract the value

$$DH' := \begin{cases} \left(DH/(X^*A^d)^{s} \cdot (B^*)^{e_1 ad}\right)^{{e_1}^{-1}}, & \pi^* \text{ is a responder session} \\ DH/(S(B^*)^{e_1})^{ad}, & \pi^* \text{ is an initiator session} \end{cases}$$

     where $d := h_0(A, X^*)$ and $h_1(B^*, S)$. If π^* is an initiator session, then $\mathcal{B}_{\mathsf{lte}}$ aborts the simulation and return DH' as the $\mathsf{CRGapDH}$ solution. If π^* is a responder session, then we have $DH' = (X^*)^{b^*}$, and we can use the same approach in KDF_f to construct a valid $\mathsf{CRGapDH}$ solution.
  2. Otherwise, simulate by lazy sampling.

Here we only prove that the simulation of KDF_f and KDF_r can extract the $\mathsf{CRGapDH}$ solution. In KDF_f, if π^* is an initiator session with full mode, then we have $DH' = (Y(B^*)^{e_1}S^{e_2})^{x^*} = ((YS^{e_2}) \cdot (B^*)^{e_1})^{x^*}$, where $e_1 = h_1(B^*, Y, S)$ is from $\textsc{Ch}(YS^{e_2})$, so DH' is a valid solution in this case. A similar argument applies to KDF_r. Therefore, we have

$$\mathsf{Adv}^{\mathcal{G}_3^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{B}_{\mathsf{ltss}}) + Q_{\mathrm{RO}}/p$$

The remain argument to bound $\Pr[\mathtt{QueryDH}_{\mathsf{ltss}}]$ is the same as in previous cases. We have $\mathsf{Adv}^{\mathcal{G}_4^{\mathsf{elt}}}_{\mathsf{XHMQV}}(\mathcal{A}) = 0$. We assume that the ϵ-$\mathsf{CRGapDH}$ assumption holds over $(\mathbb{G}, p, g)$. Combining all the probability bounds gives us the final bound of the case $\mathsf{clean}_{\mathsf{LTSS}}$:

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{ELT}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u n_s \cdot (\delta + \epsilon + Q_{\mathrm{RO}}/p)$$

A.5 Case $\mathsf{clean}_{\mathsf{ESS}}$

In this case, $\mathsf{clean}_{\mathsf{ESS}}$ ensures for π^* that:

- The initiator randomness involved in π^* is not revealed.
- The responder's semi-static key is uncompromised.
- If π^* is an initiator session, then the responder's long-term key was uncompromised upon acceptance or $\pi^*.sspk \in \boldsymbol{sspk}_{\pi^*.\mathsf{pid}}$.
  We can use the argument in Appendix A.2 to compress this condition into simply $\pi^*.sspk \in \boldsymbol{sspk}_{\pi^*.\mathsf{pid}}$ (without considering $\pi^*.\mathsf{pcorr}$). Roughly, $\mathbf{G}_2$ ensures that $\mathcal{A}$ cannot forge a signature on an uncompromised party, so we can argue that $\mathsf{clean}_{\mathsf{ESS}}(\pi^*) \implies \pi^*.sspk \in \boldsymbol{sspk}_{\pi^*.\mathsf{pid}}$. Therefore, the third condition described above and the modification in $\mathbf{G}_2$ ensure that the semi-static key used in π^* has been generated by the game. This enables us to embed a $\mathsf{CRGapDH}$ challenge into the semi-static key.
- If π^* is a responder session, then there exists a partner initiator session. This condition ensures that the initiator ephemeral key of π^* is controlled by the game so that we can embed $\mathsf{CRGapDH}$ challenge when doing the reduction.

In this proof case, we reuse some arguments from proving $\mathsf{clean}_{\mathsf{ELT}}$ and $\mathsf{clean}_{\mathsf{LTSS}}$ (cf. Appendices A.2 and A.4). The main difference is that the reductions always know the long-term secret keys of all parties, so we do not need to use the simulatability property of Sig. We begin with $\mathcal{G}_0^{\mathsf{ess}} := \mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{ESS}}$, so we have

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{ESS}}}_{\mathsf{XHMQV}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}_0^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

Game $\mathcal{G}_1^{\mathsf{ess}}$: We guess the initiator session π_i^* involved in the test session π^* (i.e., either $\pi^* = \pi_i^*$ or $\pi^*.\mathsf{sid} = \pi_i^*.\mathsf{sid}$). If the guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote π_i^*'s ephemeral key pair as (x^*, X^*). We apply the same guessing approach used in proving $\mathcal{G}_1^{\mathsf{lte}}$ and $\mathcal{G}_1^{\mathsf{elt}}$ (cf. Appendices A.3 and A.4) to this proof case. We have

$$\mathsf{Adv}^{\mathcal{G}_0^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A}) = n_s \cdot \mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

Game $\mathcal{G}_2^{\mathsf{ess}}$: We guess the identity $\mathtt{B}^*$ of the responder involved in π^* and the identifier $\mathsf{ssid}^* \in [n_{ss}]$ of the repsonder $\mathtt{B}^*$'s uncompromised semi-static key. We denote the semi-static key pair as (s^*, S^*). We have

$$\mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_u n_{ss} \cdot \mathsf{Adv}^{\mathcal{G}_2^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

Game $\mathcal{G}_3^{\mathsf{ess}}$: We introduce an abort event $\mathtt{QueryDH}_{\mathsf{ess}}$ in $\mathcal{G}_3^{\mathsf{ess}}$: Specifically, if

- $\mathcal{A}$ queries KDF_f on $(DH, A, B^*, S^*, Y, X^*)$, where $d = h_0(A, X^*), e_1 = h_1(B^*, Y, S^*), e_2 = h_2(B^*, Y, S^*)$, $\textsc{Ddh}(X^*A^d, YB^{e_1}(S^*)^{e_2}, DH) = 1$, and $(A, B^*, S^*, Y, X^*) = \pi^*.ctxt$, or
- $\mathcal{A}$ queries KDF_r on (DH, A, B^*, S^*, X^*), where $d = h_0(A, X^*)$, $e_1 = h_3(B^*, S^*)$, and $\textsc{Ddh}(X^*(A)^d, S^*B^{e_1}, DH) = 1$ and $(A, B^*, S^*, X^*) = \pi^*.ctxt$,

then the game aborts (i.e., return 0 as $\mathcal{A}$'s bit guess). In other words, we abort the game if $\mathcal{A}$ queries KDF_f or KDF_r on DH value that contains the information of $(S^*)^{x^*} (= (X^*)^{s^*})$. We construct a CRGapDH adversary $\mathcal{B}_{\mathsf{ess}}$ such that if such an abort event happens, then $\mathcal{B}_{\mathsf{ess}}$ can output a solution to its CRGapDH challenge.

Reduction $\mathcal{B}_{\mathsf{ess}}^{\text{DDH,CH}}(X^*, S^*)$:

- Initialize $\mathcal{G}_3^{\mathsf{ess}}$:
  1. Choose a party $\mathtt{B}^*$ uniformly at random and pick $\mathsf{ssid}^* \leftarrow\!\!\$\ [n_{ss}]$.
  2. Initialize a counter $\mathsf{cnt} := 0$ and choose $\mathsf{cnt}^* \leftarrow\!\!\$\ [n_s]$. Initialize $\pi_i^* := \perp$ and set up it later.
  3. Initialize lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$.
  4. For all parties, follow the KGenLT and KGenSS defined in XHMQV to generate $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$. After this step, $\mathcal{B}_{\mathsf{ess}}$ has $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$. $\mathcal{B}_{\mathsf{ess}}$ further replaces $\boldsymbol{sspk}_{\mathtt{B}^*}[\mathsf{ssid}^*]$ with S^*.
  5. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceeds the game as in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$. If the responder of π^* is not $\mathtt{B}^*$, π_i^* is not π^*, or π_i^* is not the matching session of π^*, then overwrite $\mathcal{A}$'s bit with 0.

  The random oracles and game oracles are simulated as described below.
- Simulation of oracles defined in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$:
  - All oracles except SEND, CORRUPTSSKEY, and REVEALRAND are simulated in the same way defined in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$.
  - If CORRUPTSSKEY$(\mathtt{B}^*, \mathsf{ssid}^*)$ is queried, then abort the game and overwrite $\mathcal{A}$'s bit with 0. CORRUPTSSKEY queries on other parties are simulated as usual.
  - If REVEALRAND$(\mathtt{U}, j)$ is queried, where $\pi_j^{\mathtt{U}} = \pi_i^* \neq \perp$, then abort the game and overwrite $\mathcal{A}$'s bit with 0. REVEALRAND queries on other sessions are simulated as usual.
  - SEND$(\mathtt{U} \neq \mathtt{B}^*, j, m)$:
    - Case $m = (\mathtt{B}^*, sspk, epk)$ (initiator session with intended partner $\mathtt{B}^*$): $\mathsf{cnt} := \mathsf{cnt} + 1$. If $\mathsf{cnt} = \mathsf{cnt}^*$, then we use X^* as the ephemeral pk of this session and set $\pi_i^* := \pi_j^{\mathtt{U}}$. Otherwise, simulate as usual.
    - For other cases, simulate as usual.
  - For all other SEND queries, We apply the same approach used in $\mathcal{B}_{\mathsf{ltlt}}$ to simulate the session key. Namely, we use the lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ and oralce patching of KDF_f and KDF_r (using DDH) to simualte the session keys and the RO outputs of KDF_f and KDF_r consistently.
- Simulation of Random Oracles:
  - For fresh RO queries to $\mathsf{KDF}_r(DH, A, B, S, X)$ and $\mathsf{KDF}_f(DH, A, B, S, Y, X)$ that do not involve (B^*, S^*, X^*), we simulate in the same way as in $\mathbf{G}_2$. Similarly, for fresh RO queries to h_0, h_1, h_2, and h_3 that do not involve (B^*, S^*, X^*), we simulate as usual.
  - $h_2(B^*, Y, S^*)$: $h \leftarrow \text{CH}(Y)$, $h_2(B^*, Y, S^*) := h$
  - h_0, h_1, and h_3: Simulate as usual.
  - $\mathsf{KDF}_f(DH, A, B^*, S^*, Y, X^*)$:

1. If $(A, B^*, S^*, Y, X^*) = \pi^*.ctxt$ and $\textsc{Ddh}(X^*A^d, YB^{e_1}(S^*)^{e_2}, DH)$ outputs 1, then extract the value

$$DH' := \begin{cases} \left(DH/(YB^{e_1}(S^*)^{e_2})^{ad} \cdot (X^*)^{y+e_1 b}\right)^{{e_2}^{-1}}, & \pi^* \text{ is a responder session} \\ DH/(YB^{e_1}(S^*)^{e_2})^{ad} \cdot (X^*)^{e_1 b}, & \pi^* \text{ is an initiator session} \end{cases}$$

   where $d := h_0(A, X^*), e_1 := h_1(B^*, Y, S^*)$, and $e_2 := h_2(B^*, Y, S^*)$. We have a since we have long-term keys of all parties. If π^* is an initiator session, then $\mathcal{B}_{\mathsf{ess}}$ aborts the simulation and return DH' as the $\mathsf{CRGapDH}$ solution. If π^* is a responder session (so we have the ephemeral key y), then we have that $DH' = (X^*)^{s^*}$, and $\mathcal{B}_{\mathsf{ess}}$ can use a similar approach in $\mathcal{B}_{\mathsf{ess}}$ to construct a valid $\mathsf{CRGapDH}$ solution.
2. Otherwise, simulate by lazy sampling.

- $\mathsf{KDF}_r(DH, A, B^*, S^*, X^*)$:
  1. If $(A, B^*, S^*, X^*) = \pi^*.ctxt$ and $\textsc{Ddh}(X^*A^d, (S^*)B^{e_1}, DH)$ outputs 1, then extract the value

$$DH' := DH/(S^*B^{e_1})^{ad} \cdot (X^*)^{e_1 b}$$

     where $d := h_0(A, X^*), e_1 := h_1(B^*, Y, S^*)$, and $e_2 := h_2(B^*, Y, S^*)$. We have a and b since we have long-term keys of all parties. Since we have $DH' = (X^*)^{s^*}$ in this case, $\mathcal{B}_{\mathsf{ess}}$ can use a similar approach in $\mathcal{B}_{\mathsf{ess}}$ to construct a valid $\mathsf{CRGapDH}$ solution.
  2. Otherwise, simulate by lazy sampling.

Here we only prove that the simulation of KDF_f can extract the $\mathsf{CRGapDH}$ solution. If π^* is an initiator session with full mode, then we have $DH' = (Y(S*)^{e_2})^{x^*}$, where $e_2 = h_2(B^*, Y, S^*)$ is from $\textsc{Ch}(Y)$, so DH' is a valid solution in this case. Therefore, we have

$$\mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}_4^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{B}_{\mathsf{ess}}) + Q_{\mathrm{RO}}/p$$

The remain argument to bound $\Pr[\mathtt{QueryDH}_{\mathsf{ess}}]$ is the same as in previous cases. We have $\mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ess}}}_{\mathsf{XHMQV}}(\mathcal{A}) = 0$ and ϵ-$\mathsf{CRGapDH}$ assumption holds over $(\mathbb{G}, p, g)$. Combining all the probability bounds gives us the final bound of the case $\mathsf{clean}_{\mathsf{ESS}}$:

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{ESS}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_s n_u n_{ss} \cdot (\epsilon + Q_{\mathrm{RO}}/p)$$

A.6 Case $\mathsf{clean}_{\mathsf{EE}}$

In this case, $\mathsf{clean}_{\mathsf{EE}}$ ensures for π^* that:

– The randomness of π^* is not revealed, and
– π^* has a partnered session whose randomness is also not revealed.

In this proof case, we reuse some arguments from proving $\mathsf{clean}_{\mathsf{ELT}}$ and $\mathsf{clean}_{\mathsf{LTE}}$ (cf. Appendices A.3 and A.4). Similar to the case of $\mathsf{clean}_{\mathsf{ESS}}(\pi^*)$, the reductions always know the long-term secret keys of all parties, so we do not need to use the simulatability property of Sig. We begin with $\mathcal{G}_0^{\mathsf{ee}} := \mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{EE}}$, so we have

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{EE}}}_{\mathsf{XHMQV}}(\mathcal{A}) = \mathsf{Adv}^{\mathcal{G}_0^{\mathsf{ee}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_1^{\mathsf{ee}}$: We guess the initiator session π_i^* involved in the test session π^* (i.e., either $\pi^* = \pi_i^*$ or $\pi^*.\mathsf{sid} = \pi_i^*.\mathsf{sid}$). If the guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote π_i^*'s ephemeral key pair as (x^*, X^*). We use the same guessing approach as for proving $\mathcal{G}_1^{\mathsf{ee}}$ and $\mathcal{G}_1^{\mathsf{ee}}$ (cf. Appendices A.3 and A.4). We have

$$\mathsf{Adv}^{\mathcal{G}_0^{\mathsf{ee}}}_{\mathsf{XHMQV}}(\mathcal{A}) = n_s \cdot \mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ee}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

GAME $\mathcal{G}_2^{\mathsf{ee}}$: We guess the responder session π_r^* involved in the test session π^* (i.e., either $\pi^* = \pi_i^*$ or $\pi^*.\mathsf{cid} = \pi_i^*.\mathsf{cid}$). If the guess is incorrect, then we overwrite $\mathcal{A}$'s bit guess with 0. We denote π_r^*'s ephemeral key pair as (y^*, Y^*). We use the same guessing approach in proving $\mathcal{G}_1^{\mathsf{lte}}$ and $\mathcal{G}_2^{\mathsf{elt}}$ (cf. Appendices A.3 and A.4). We have

$$\mathsf{Adv}^{\mathcal{G}_1^{\mathsf{ee}}}_{\mathsf{XHMQV}}(\mathcal{A}) = n_s \cdot \mathsf{Adv}^{\mathcal{G}_2^{\mathsf{ee}}}_{\mathsf{XHMQV}}(\mathcal{A})$$

Now if the guess of π_i^* and π_r^* is correct, then either $\pi^* = \pi_i^*$ or $\pi^* = \pi_r^*$.

GAME $\mathcal{G}_3^{\mathsf{ee}}$: We introduce an abort event $\mathtt{QueryDH}_{\mathsf{ee}}$ in $\mathcal{G}_3^{\mathsf{ee}}$: Specifically, if $(A, B, S, Y^*, X^*) = \pi^*.ctxt$, and

- $\mathcal{A}$ queries $\mathsf{KDF}_f(DH, A, B, S, Y^*, X^*)$, where $d = h_0(A, X^*), e_1 = h_1(B, Y^*, , S), e_2 = h_2(B, Y^*, S)$, $\textsc{Ddh}(X^* A^d, Y^* B^{e_1} S^{e_2}, DH) = 1$,

then the game aborts (i.e., return 0 as $\mathcal{A}$'s bit guess). In other words, we abort the game if $\mathcal{A}$ queries KDF_f on DH value that contains the information of $(Y^*)^{x^*} (= (X^*)^{y^*})$. We construct a $\mathsf{CRGapDH}$ adversary $\mathcal{B}_{\mathsf{ee}}$ such that if such an abort event happens, then $\mathcal{B}_{\mathsf{ee}}$ can output a solution to its $\mathsf{CRGapDH}$ challenge.

Reduction $\mathcal{B}_{\mathsf{ee}}^{\textsc{Ddh},\textsc{Ch}}(X^*, S^*)$:

- Initialize $\mathcal{G}_3^{\mathsf{ee}}$:
  1. Choose a $\mathtt{B}^*$ uniformly at random. Pick $\mathsf{ssid}^* \leftarrow\!\!\$\ n_{ss}$.
  2. Initialize counters $\mathsf{cnt}_1 := 0$ and $\mathsf{cnt}_2 := 0$ and choose $\mathsf{cnt}_1^*, \mathsf{cnt}_2^* \leftarrow\!\!\$\ [n_s]$. Initialize $\pi_i^* := \perp$ and Initialize $\pi_j^* := \perp$ and set up them later.
  3. Initialize lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$.
  4. For all parties, follow the KGenLT and KGenSS defined in XHMQV to generate $\boldsymbol{ltpk}$, $\boldsymbol{sspk}$, and $\boldsymbol{ltsk}$.
  5. Run $b_{\mathsf{guess}} \leftarrow\!\!\$\ \mathcal{A}(\boldsymbol{ltpk}, \boldsymbol{sspk})$ and proceeds the game as in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$. If π^* and the partner session of π^* are not π_i^* and π_j^*, then overwrite b_{guess} with 0. The random oracles and game oracles are simulated as described below.
- Simulation of oracles defined in $\mathcal{G}_{\mathsf{KE}}^{\mathsf{KI}}$:

- All oracles except SEND and REVEALRAND are simulated in the same way defined in $\mathcal{G}^{\mathsf{KI}}_{\mathsf{KE}}$.
- If REVEALRAND$(\mathtt{U}, j)$ is queried, where $\pi^{\mathtt{U}}_j = \pi^*_i \neq \bot$ or $\pi^{\mathtt{U}}_j = \pi^*_r \neq \bot$, then abort the game and overwrite $\mathcal{A}$'s bit with 0. REVEALRAND queries on other sessions are simulated as usual.
- SEND$(\mathtt{U}, j, m)$:
  - Case $m = (\mathtt{V}, sspk, epk)$: $\mathsf{cnt}_1 := \mathsf{cnt}_1 + 1$. If $\mathsf{cnt}_1 = \mathsf{cnt}^*_1$, then $\mathcal{B}_{\mathsf{ee}}$ uses X^* as the ephemeral pk of this session and set $\pi^*_i := \pi^{\mathtt{U}}_j$. Otherwise, simulate as usual.
  - Case $m = (\mathsf{create}, \mathsf{ssid}, \mathsf{type})$: $\mathsf{cnt}_2 := \mathsf{cnt}_2 + 1$. If $\mathsf{cnt}_2 = \mathsf{cnt}^*_2$, then $\mathcal{B}_{\mathsf{ee}}$ uses Y^* as the ephemeral pk of this session and set $\pi^*_r := \pi^{\mathtt{U}}_j$. $\mathcal{B}_{\mathsf{ee}}$ further sets $h_2(U, Y^*, S) := h^{-1}$ where U is the long-term public key of $\mathtt{U}$, $h \leftarrow$ CH(S), S is the semi-static pk of $\pi^{\mathtt{U}}_j$. Otherwise, simulate as usual.
  - For other cases, simulate as usual.
- For all other SEND queries, we apply the same approach used in $\mathcal{B}_{\mathsf{ltlt}}$ to simulate the session key. Namely, we use the lists $\mathcal{L}_{\mathsf{patch\text{-}f}}$, $\mathcal{L}_{\mathsf{patch\text{-}r}}$, $\mathcal{L}_{\mathsf{skey\text{-}f}}$, and $\mathcal{L}_{\mathsf{skey\text{-}r}}$ and oralce patching of KDF_f and KDF_r (using DDH) to simualte the session keys and the RO outputs of KDF_f and KDF_r consistently.

– Simulation of Random Oracles:
- For fresh RO queries to $\mathsf{KDF}_f(DH, A, B, S, Y, X)$ that do not involve (X^*, Y^*), we simulate in the same way as in $\mathbf{G}_2$.
- h_0, h_1, h_2, h_3, and KDF_r: Simulate as usual.
- $\mathsf{KDF}_f(DH, A, B, S, Y^*, X^*)$:
  1. If $(A, B, S, Y^*, X^*) = \pi^*.ctxt$ and DDH$(X^*A^d, Y^*B^{e_1}S^{e_2}, DH)$ outputs 1, then extract the value

$$DH' := \begin{cases} DH/(Y^*B^{e_1}S^{e_2})^{ad} \cdot (X^*)^{e_1 b + e_2 s}, & \pi^* \text{ is a responder session} \\ \left(DH/(Y^*B^{e_1}S^{e_2})^{ad} \cdot (X^*)^{e_1 b}\right)^{{e_2}^{-1}}, & \pi^* \text{ is an initiator session} \end{cases}$$

     where $d := h_0(A, X^*), e_1 := h_1(B, Y^*, S)$, and $e_2 := h_2(B, Y^*, S)$. We have a and b since we have long-term keys of all parties. If π^* is an initiator session, then $\mathcal{B}_{\mathsf{ee}}$ aborts the simulation and return DH' as the CRGapDH solution. If π^* is a responder session (and we have the semi-static key s), then we have that $DH' = (X^*)^{y^*}$, and $\mathcal{B}_{\mathsf{ee}}$ can use a similar approach in $\mathcal{B}_{\mathsf{ee}}$ to construct a valid CRGapDH solution.
  2. Otherwise, simulate by lazy sampling.

If π^* is an initiator session with full mode, then we have $DH' = (Y^*(S)^{e_2})^{{x^*e_2}^{-1}}$, where $e_2 = h_2(B, Y^*, S) = h^{-1}$ and h is from CH(S). So, we also have

$$DH' = (Y^*(S)^{e_2})^{{x^*e_2}^{-1}} = (Y^*(S)^{h^{-1}})^{x^*h} = (S(Y^*)^h)^{x^*}$$

Therefore, DH' is a valid solution in this case, and we have

$$\mathsf{Adv}^{\mathcal{G}^{\mathsf{ee}}_3}_{\mathsf{XHMQV}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathcal{G}^{\mathsf{ee}}_4}_{\mathsf{XHMQV}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{CRGapDH}}_{(\mathbb{G},p,g)}(\mathcal{B}_{\mathsf{ess}}) + Q_{\mathrm{RO}}/p$$

The remain argument to bound $\Pr[\texttt{QueryDH}_{\mathsf{ee}}]$ is the same as in previous cases. We have $\mathsf{Adv}^{\mathcal{G}_3^{\mathsf{ee}}}_{\mathsf{XHMQV}}(\mathcal{A}) = 0$. By assuming ϵ-CRGapDH holds over $(\mathbb{G}, p, g)$ and combining all the probability bounds, we have the final bound of the case $\mathsf{clean}_{\mathsf{ESS}}$:

$$\mathsf{Adv}^{\mathbf{G}_2 \wedge \mathsf{clean}_{\mathsf{EE}}}_{\mathsf{XHMQV}}(\mathcal{A}) \leq n_s^2 \cdot (\epsilon + Q_{\mathrm{RO}}/p)$$

B Deferred Material on Deniability

We recall the formal definition for deniability of key exchange from Fiedler and Janson [24]. The model includes user messages, i.e., "Hey Bob, what's up?", as μ. We leave the user messages μ empty for clarity and since the subsequent AEAD encryption of user messages is not affected by the changes proposed in our protocol XHMQV. We refer the reader to [24, Section 3] for a full explanation of the model.

Definition 5 (Deniability for Key Exchange [24]). *We say that a key exchange protocol* $\mathsf{KE} = (\mathsf{KGenLT}, \mathsf{KGenSS}, \mathsf{Run})$ *is* $(n_p, n_{ss}, q_{\mathrm{O_A}}, q_{\mathrm{O_F}}, q_{\mathrm{O_D}}, t_{\mathcal{A}}, t_{\mathsf{Dist}}, \epsilon_{\mathsf{Dist}})$*-deniable wrt.* $(\mathrm{O}_A, \mathrm{O}_F, \mathrm{O}_D)$*-oracles, where* $\mathrm{O}_A \subseteq \{\textsc{Reg}, \textsc{Init}, \textsc{ChallInit}, \textsc{ChallResp}, \textsc{RegHon}, \textsc{ChallHonInit}, \textsc{ChallHonResp}\}$, $\mathrm{O}_F \subseteq \{\mathrm{SK}, \textsc{User}_{x,y}\}$, $\mathrm{O}_D \subseteq \{\mathrm{SKs}, \textsc{aux}\}$*, and auxiliary inputs sampled with* AuxPrep*, if for any adversary* $\mathcal{A}$ *running in time* $t_{\mathcal{A}}$*, making queries to* O_A *limited by* $q_{\mathrm{O_A}}$*, there exists an algorithm* Fake *making queries to* O_F *limited by* $q_{\mathrm{O_F}}$ *such that for any distinguisher* Dist *making queries to* O_D *limited by* $q_{\mathrm{O_D}}$ *and running in time* t_{Dist}*, it holds that*

$$\Pr[\mathcal{G}^{\mathrm{O}_A,\mathrm{O}_F,\mathrm{O}_D\text{-}den}_{\mathsf{KE},\mathsf{AuxPrep},n_p,n_{ss}}(\mathcal{A}, \mathsf{Dist}) = 1] \leq \frac{1}{2} + \epsilon_{\mathsf{Dist}},$$

where $\mathcal{G}^{\mathrm{O}_A,\mathrm{O}_F,\mathrm{O}_D\text{-}den}_{\mathsf{KE},\mathsf{AuxPrep},n_p,n_{ss}}(\mathcal{A}, \mathsf{Dist})$ *is defined in Figures 11 and 12.*

B.1 Deferred proofs of deniability for XHMQV

Here, we give the deferred proofs for deniability, beginning with initiator (Bob) deniability. Note that the model of [24] considers the party who sends the first message as initiator, in contrast to the setting of Signal, where we usually consider Alice as initiator.

Proof (of Theorem 2). We give the Fake algorithm in Figure 13.

The Fake algorithm simulates Bob's pre-key bundle by sampling a fresh ephemeral key (if needed) and learns a signature on the semi-static key from the auxiliary info *aux*. Furthermore, to process Alice's message, the Fake algorithm

$\mathcal{G}^{\mathrm{O}_A,\mathrm{O}_F,\mathrm{O}_D\text{-}den}_{\mathsf{KE},\mathsf{AuxPrep},n_p,n_{ss}}(\mathcal{A},\mathsf{Dist})$:

$Q[\cdot] := []; \quad K[\cdot] := \bot; \quad C := \emptyset$
$(\mathbf{pk},\mathbf{sk}) \leftarrow\!\$\ \mathsf{KeyPrep}(n_p, n_{ss})$
$aux \leftarrow\!\$\ \mathsf{AuxPrep}(\mathbf{pk},\mathbf{sk})$
$b \leftarrow\!\$\ \{0,1\}$
$r \leftarrow\!\$\ \mathcal{R}_{\mathcal{A}}$ // randomness for the adversary
$\mathcal{A}^{\mathrm{O}_A}(\mathbf{pk}, aux; r)$
$b' \leftarrow\!\$\ \mathsf{Dist}^{\mathrm{O}_D}(\mathbf{pk}, r, Q, K)$
return $[\![b' = b]\!]$

$\mathsf{KeyPrep}(n_p, n_{ss})$:

for $U \in [n_p]$ // number of keys to prepare
  $(ltpk_U, ltsk_U) \leftarrow\!\$\ \mathsf{KGenLT}()$
  for $\mathsf{ssid} \in [n_{ss}]$
    $(sspk_U^{\mathsf{ssid}}, sssk_U^{\mathsf{ssid}}) \leftarrow\!\$\ \mathsf{KGenSS}()$
$\boldsymbol{ltpk} := \{ltpk_U\}_{U\in[n_p]}$
$\boldsymbol{sspk} := \{sspk_U^{\mathsf{ssid}}\}_{U\in[n_p]}^{\mathsf{ssid}\in[n_{ss}]}$
$\boldsymbol{ltsk} := \{ltsk_U\}_{U\in[n_p]}$
$\boldsymbol{sssk} := \{sssk_U^{\mathsf{ssid}}\}_{U\in[n_p]}^{\mathsf{ssid}\in[n_{ss}]}$
return $((\boldsymbol{ltpk}, \boldsymbol{sspk}), (\boldsymbol{ltsk}, \boldsymbol{sssk}))$

$\textsc{Init}(U, s, \mathsf{role}, V)$:

if $\pi_U^s = \bot \wedge U \notin C$
  $\pi_U^s.\mathsf{role} := \mathsf{role}$
  $\pi_U^s.\mathsf{oid} := U$ // set owner identity
  $\pi_U^s.\mathsf{pid} := V$
// set partner identity ($\star$ if post-specified peers)
  return success
return $\bot$

$\textsc{User}_{x,y}(\pi, u, m, \mu)$:

$U := \pi.\mathsf{oid}$
$H := H_\pi^u$
// easier notation for u^{th} honest dummy for session π
if $\pi_U^H = \bot$
  $(\mathsf{pk}_H, \mathsf{sk}_H) \leftarrow\!\$\ \mathsf{KeyPrep}(1,1)$
  $\mathbf{pk} := \mathbf{pk} \cup \{\mathsf{pk}_H\}$
  if $\pi.\mathsf{role} = \mathsf{initiator}$
    $\pi_U^H.\mathsf{role} := x$
  else
    $\pi_U^H.\mathsf{role} := y$
  $\pi_U^H.\mathsf{oid} := U$
  $\pi_U^H.\mathsf{pid} := H$
$(\pi_U^H, m', \mu') \leftarrow\!\$\ \mathsf{Run}(\mathsf{sk}_U, \mathbf{pk}, \pi_U^H, m, \mu)$
return m'

$\textsc{RegHon}(U)$:

if $U \in [n_p] \cup C$
// abort if another key is already known for U
  return $\bot$
$(\mathbf{pk}_U, \mathbf{sk}_U) \leftarrow\!\$\ \mathsf{KeyPrep}(1, n_{ss})$
$\mathbf{pk} \xleftarrow{+} \mathbf{pk}_U; \quad \mathbf{sk} \xleftarrow{+} \mathbf{sk}_U$
$C := C \cup \{U\}$
return $(\mathbf{pk}_U, \mathbf{sk}_U)$
// simulates $\mathcal{A}$ honestly generating keys

$\textsc{Reg}(U, \mathbf{pk}_U)$:

if $U \in [n_p] \cup C$
// abort if another key is already known for U
  return $\bot$
$C := C \cup \{U\}$
$\mathbf{pk} \xleftarrow{+} \mathbf{pk}_U$ // includes semi-static keys
return success

$\textsc{SK}(\pi)$:

return $\mathsf{sk}_{\pi.\mathsf{pid}}$

$\textsc{SKs}()$:

return $\mathbf{sk}$

$\textsc{aux}()$:

return aux

Fig. 11. The deniability game for key exchange as defined in Definition 5. The challenge oracles are in Figure 12. The oracles are split into the adversary's capabilities $\mathrm{O}_A \subseteq \{\textsc{Reg}, \textsc{Init}, \textsc{ChallInit}, \textsc{ChallResp}, \textsc{RegHon}, \textsc{ChallHonInit}, \textsc{ChallHonResp}\}$, the capabilities of the Fake algorithm $\mathrm{O}_F \subseteq \{\textsc{SK}, \textsc{User}_{x,y}\}$, and the capabilities of the distinguisher Dist $\mathrm{O}_D \subseteq \{\textsc{SKs}, \textsc{aux}\}$.

ChallInit(U, s, m, μ) [ChallResp(U, s, m, μ)]:

if $\pi_U^s = \perp$ // initialized session only
  $\vee\ \pi_U^s.\mathsf{role} = \mathsf{responder}$ [$\mathsf{initiator}$]
  return $\perp$
if $b = 0$
  $(\pi_U^s, m') \leftarrow\!\!\$\ \mathsf{Run}(\mathsf{sk}_U, \mathbf{pk}, \pi_U^s, m, \mu)$
else
  $(\pi_U^s, m') \leftarrow\!\!\$\ \mathsf{Fake}^{\mathrm{O}_F}(\mathbf{pk}, \pi_U^s, m, \mu, aux, r)$
$Q[\pi_U^s] \xleftarrow{+} (m, m')$
$K[\pi_U^s] \leftarrow \pi_U^s.\mathsf{K}$
return m'

ChallHonInit$(U, V, \mathsf{info}_{\mathsf{create}}, \boldsymbol{\mu}, r_C)$:

if $U \in C$ // do not allow challenging a party under $\mathcal{A}$'s control
  return $\perp$
$\pi_U.\mathsf{oid} \leftarrow U$; $\pi_U.\mathsf{pid} \leftarrow V$
$\pi_V.\mathsf{oid} \leftarrow V$; $\pi_V.\mathsf{pid} \leftarrow U$
$\pi_U.\mathsf{role} \leftarrow \mathsf{initiator}$; $\pi_V.\mathsf{role} \leftarrow \mathsf{responder}$
$m_1 \leftarrow (\mathsf{create}, \mathsf{info}_{\mathsf{create}})$
$(\mu_1, \ldots, \mu_{n_m}) \leftarrow \boldsymbol{\mu}$
for $i \in [1, 3, \ldots, n_m - 1]$ // until n_m if n_m is odd
  if $b = 0$
    $(\pi_U, m_{i+1}) \leftarrow\!\!\$\ \mathsf{Run}(\mathsf{sk}_U, \mathbf{pk}, \pi_U, m_i, \mu_i)$
  else
    $(\pi_U, m_{i+1}) \leftarrow\!\!\$\ \mathsf{Fake}^{\mathrm{O}_F}(\mathbf{pk}, \pi_U, m_i, \mu_i, aux, r_C)$
  $(\pi_V, m_{i+2}) \leftarrow \mathsf{Run}(\mathsf{sk}_V, \mathbf{pk}, \pi_V, m_{i+1}, \mu_{i+1}; r_C)$
$Q[\pi_U] \leftarrow (m_i)_{i=1}^{n_m}$; $K[\pi_U] \leftarrow \pi_U.K$
return $(Q[\pi_U], K[\pi_U])$

ChallHonResp$(U, V, \mathsf{info}_{\mathsf{create}}, \boldsymbol{\mu}, r_C)$:

if $V \in C$ // do not allow challenging a party under $\mathcal{A}$'s control
  return $\perp$
$\pi_U.\mathsf{oid} \leftarrow U$; $\pi_U.\mathsf{pid} \leftarrow V$
$\pi_V.\mathsf{oid} \leftarrow V$; $\pi_V.\mathsf{pid} \leftarrow U$
$\pi_U.\mathsf{role} \leftarrow \mathsf{initiator}$; $\pi_V.\mathsf{role} \leftarrow \mathsf{responder}$
$m_1 \leftarrow (\mathsf{create}, \mathsf{info}_{\mathsf{create}})$
$(\mu_1, \ldots, \mu_{n_m}) \leftarrow \boldsymbol{\mu}$
for $i \in [1, 3, \ldots, n_m - 1]$ // until n_m if n_m is odd
  $(\pi_U, m_{i+1}) \leftarrow \mathsf{Run}(\mathsf{sk}_U, \mathbf{pk}, \pi_U, m_i, \mu_i; r_C)$
  if $b = 0$
    $(\pi_V, m_{i+2}) \leftarrow\!\!\$\ \mathsf{Run}(\mathsf{sk}_V, \mathbf{pk}, \pi_V, m_{i+1}, \mu_{i+1})$
  else
    $(\pi_V, m_{i+2}) \leftarrow\!\!\$\ \mathsf{Fake}^{\mathrm{O}_F}(\mathbf{pk}, \pi_V, m_{i+1}, \mu_{i+1}, aux, r_C)$
$Q[\pi_V] \leftarrow (m_i)_{i=1}^{n_m}$; $K[\pi_V] \leftarrow \pi_V.K$
return $(Q[\pi_V], K[\pi_V])$

Fig. 12. The challenge oracles for the deniability game in Definition 5, separate from Figure 11 for space reasons. The suffix of the oracle name indicates as which party the oracle acts. The infix Hon indicates that this oracle enforces semi-honest behavior of the adversary.

$\underline{\mathsf{Fake}(\mathbf{pk} = (\boldsymbol{ltpk}, \boldsymbol{sspk}), \pi, m, \mu, aux, r_C):}$

1 $(\mathsf{create}, (\mathsf{ssid}, \mathsf{type})) := m$
2 $\mathsf{B} := \pi.\mathsf{oid}$
3 $\pi.(\mathsf{pid}, \mathsf{sspk}, \mathsf{type}) := (\star, sspk, \mathsf{type})$
4 $sspk := \boldsymbol{sspk}_{\mathtt{B}}[\mathsf{ssid}]$
5 $\sigma :=$ from aux for $sspk$
6 **if** $\mathsf{type} = \mathsf{full}$
7 $\quad y \leftarrow\!\!\$\ \mathbb{Z}_p,\ epk_{\mathtt{B}} := g^y$
8 **else** $epk_{\mathtt{B}} := \bot$
9 $m := (\mathsf{B}, sspk, epk_{\mathtt{B}}, \boxed{\sigma})$
10 **return** (π, m, ε)

$\underline{\mathsf{Fake}(\mathbf{pk}, \pi, m = (\mathtt{A}, X), \mu, aux, r_C):}$

11 $\pi.\mathsf{pid} := \mathtt{A}, \quad A := \boldsymbol{ltpk}[\mathtt{A}]$
12 $\left(a, \boxed{\mathsf{sk}_{\mathtt{A}}}\right) :=$ extract from $\mathrm{SK}(\pi)$
13 $x :=$ from r_C // Alice's ephemeral secret key
14 $X := g^x$
15 $(B, \boxed{\mathsf{vk}_{\mathtt{B}}}) := \boldsymbol{ltpk}[\pi.\mathsf{oid}], \quad S := \pi.\mathsf{sspk}$
16 $Y := \pi.\mathsf{cid}[4]$ // Bob's ephemeral public key
17 $d := h_0(A, X)$
18 **if** $Y \neq \bot$ // full handshake
19 $\quad e_1 := h_1(B, Y, S)$
20 $\quad e_2 := h_2(B, Y, S)$
21 $\quad DH := (Y B^{e_1} S^{e_2})^{x+da}$
22 $\quad ctxt := (A, B, S, Y, X)$
23 $\quad \pi.\mathsf{K} := \mathsf{KDF}_f(ctxt, DH)$
24 **else** // reduced handshake
25 $\quad e_1 := h_3(B, S)$
26 $\quad DH := (B^{e_1} S)^{x+da}$
27 $\quad ctxt := (A, B, S, X)$
28 $\quad \pi.\mathsf{K} := \mathsf{KDF}_r(ctxt, DH)$
29 **return** $(\pi, \varepsilon, \varepsilon)$

Fig. 13. The Fake algorithms simulating Bob's messages (initiator deniability), used in Theorem 2.

learns Alice's long-term secret key from the SK oracle and Alice's ephemeral secret key X from the randomness that was previously used to create Alice's message. Finally, the Fake algorithm computes the session key with Alice's ephemeral and long-term secret keys.

Since Fake produces a transcript and session key in the same way as Run, the distinguisher cannot have an advantage in winning its game, even if the distinguisher has access to all long-term and semi-static secret keys.

Next is responder (Alice) deniability.

Proof (of Theorem 3). We give the Fake algorithm in Figure 14.

The Fake algorithm verifies Bob's signature on the semi-static key. Next, it learns Bob's long-term and semi-static secret keys from the SK oracle and Bob's ephemeral secret key Y from the randomness that was previously used to create Bob's message. It samples a fresh ephemeral key X for Alice. Finally, the Fake algorithm computes the session key with Bob's ephemeral, semi-static, and long-term secret keys.

Since Fake produces a transcript and session key in the same way as Run, the distinguisher cannot have an advantage in winning its game, even if the distinguisher has access to all long-term and semi-static secret keys.

$\mathsf{Fake}(\mathbf{pk} = (\boldsymbol{ltpk}, \boldsymbol{sspk}), \pi, m, \mu, aux, r_C)$:

1 $(\mathtt{B}, sspk, epk_{\mathtt{B}}) := m, \quad \pi.\mathsf{pid} := \mathtt{B},$
2 $(B, \boxed{\mathsf{vk}_{\mathtt{B}}}) := \boldsymbol{ltpk}[\mathtt{B}] \quad S := sspk,$ // prepare Bob's keys
3 $\left(b, \boxed{\mathsf{sk}_{\mathtt{B}}}, s\right) :=$ extract from $\mathrm{SK}(\pi)$
4 $y \leftarrow$ from $r_C, \quad Y := epk_{\mathtt{B}}$
5 **if** $\mathsf{Vf}(B, \sigma, sspk) \neq 1$: **return** $(\pi, \varepsilon, \varepsilon)$ // verify signature
6 **if** $\mathsf{Vf}(\mathsf{vk}_{\mathtt{B}}, \sigma, sspk) \neq 1$: **return** $(\pi, \varepsilon, \varepsilon)$
7 $A := \boldsymbol{ltpk}[\mathtt{A}]$ // prepare Alice's keys
8 $x \leftarrow\!\!\$\ \mathbb{Z}_p, \quad X := g^x$
9 $d := h_0(A, X)$
10 **if** $Y \neq \perp$ // full handshake
11 $\quad e_1 := h_1(B, Y, S)$
12 $\quad e_2 := h_2(B, Y, S)$
13 $\quad DH := (XA^d)^{y + e_1 b + e_2 s}$
14 $\quad ctxt := (A, B, S, Y, X)$
15 $\quad \pi.\mathsf{K} := \mathsf{KDF}_f(ctxt, DH)$
16 **else** // reduced handshake
17 $\quad e_1 := h_3(\mathtt{B}, S)$
18 $\quad DH := (XA^d)^{e_1 b + s}$
19 $\quad ctxt := (A, B, S, X)$
20 $\quad \pi.\mathsf{K} := \mathsf{KDF}_r(ctxt, DH)$
21 **return** $(\pi, \varepsilon, \varepsilon)$

Fig. 14. The Fake algorithms simulating Alice's message (responder deniability), used in Theorem 3.