KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange

Yanqi Gu[1], Stanislaw Jarecki[1], and Hugo Krawczyk[2]

[1] University of California, Irvine. Email: {`yanqig1@`,`stasio@ics.`}`uci.edu.`
[2] Algorand Foundation. Email: `hugokraw@gmail.com.`

Abstract. OPAQUE [Jarecki et al., Eurocrypt 2018] is an asymmetric password authenticated key exchange (aPAKE) protocol that is being developed as an Internet standard and for use within TLS 1.3. OPAQUE combines an Oblivious PRF (OPRF) with an authenticated key exchange to provide strong security properties, including security against pre-computation attacks (called saPAKE security). However, the security of OPAQUE relies crucially on the security of the OPRF. If the latter breaks (by cryptanalysis, quantum attacks or security compromise), the user's password is exposed to an offline dictionary attack. To address this weakness, we present KHAPE, a variant of OPAQUE that does not require the use of an OPRF to achieve aPAKE security, resulting in improved resilience and near-optimal computational performance. An OPRF can be *optionally* added to KHAPE, for enhanced saPAKE security, but without opening the password to an offline dictionary attack upon OPRF compromise.
In addition to resilience to OPRF compromise, a DH-based implementation of KHAPE (using HMQV) offers the best performance among aPAKE protocols in terms of exponentiations with less than the cost of an exponentiation on top of an UNauthenticated Diffie-Hellman exchange. KHAPE uses three messages if the server initiates the exchange or four when the client does (one more than OPAQUE in the latter case).
All results in the paper are proven within the UC framework in the ideal cipher model. Of independent interest is our treatment of *key-hiding AKE* which KHAPE uses as a main component as well as our UC proofs of AKE security for protocols 3DH (a basis of Signal), HMQV and SKEME, that we use as efficient instantiations of KHAPE.

1 Introduction

In the last few years the subject of password authenticated key exchange (PAKE) protocols, particularly in the client-server setting (called *asymmetric* PAKE, or aPAKE for short), has seen renewed interest due to the weaknesses of password protocols and the ongoing standardization effort at the Internet Engineering Task Force [51]. In particular, due to vulnerabilities in PKI systems and TLS deployment, the standard PKI-based encrypted password authentication (or "password-over-TLS") often leads to disclosure of passwords and increased exploitation of phishing techniques. Even when the password is decrypted at the correct server, its presence in plaintext form after decryption, constitutes a security vulnerability as evidenced by repeated incidents where plaintext passwords were accidentally stored in large quantities and for long periods of time even by security-conscious companies [1, 2].

In this paper we investigate the question of *how "minimal" an asymmetric PAKE can be.* In spite of the many subtleties surrounding the design and analysis of aPAKE protocols, there are several efficient and practical realizations which meet a universally composable (UC) notion of aPAKE [29]. For example, the overhead of the recently analyzed SPAKE2+ protocol [53] over the *unauthenticated* Diffie-Hellman (uDH) protocol is 1 or 2 exponentiations per party. Similar overhead costs are also imposed by the generic results which compile any PAKE to aPAKE [29, 35]. Known *strong* aPAKEs (see below), add similar or larger overhead costs [39, 15].

The comparison to uDH is significant not only from a practical point of view, but also because PAKE protocols imply unauthenticated key exchange in the sense of the Impagliazzo-Rudich results [36, 30]. Thus, we can see uDH as the lowest possible expected performance of PAKE protocols. But how close to the uDH cost can we get; can one improve on existing protocols?

In the symmetric PAKE case, where the two peers share the same password, there are almost optimal answers to this question. The Bellovin-Merrit's classical EKE protocol [10], shows that all you need is to apply a symmetric-key encryption on top of the uDH transcript. It requires a carefully chosen encryption scheme, e.g., one that is modeled after an ideal cipher, but it only involves symmetric key techniques [9, 4, 14, 48].[3]

Can this low overhead relative to uDH be achieved also in the more involved setting of asymmetric PAKEs, where security against offline attacks is to be provided even when the server is broken into? We show an aPAKE protocol, KHAPE, that only requires symmetric operations (in the ideal cipher model) over regular authenticated DH.

KHAPE (for K̲ey-H̲iding A̲symmetric P̲akE̲) can be seen as a variant of the OPAQUE protocol [39] that is being developed into an Internet standard [44] and intended for use within TLS 1.3 [54]. OPAQUE introduces the idea of password-encrypted credentials containing an encrypted private key for the user and an authenticated public key for the server. The user deposits the encrypted credentials at the server during password registration and it retrieves them for login sessions, thus allowing user and server to run a regular authenticated key exchange (AKE) protocol. However, encrypting and authenticating credentials with a password opens the protocol to trivial offline dictionary attacks. Therefore, OPAQUE first runs an Oblivious PRF (OPRF) on the user's password in order to derive a strong encryption key for the credential. This makes the protocol fully reliant on the strength of the OPRF.

[3] Several other symmetric PAKE protocols, e.g. SPAKE2 [5], SPEKE [37, 45, 31] and TBPEKE [50], attain universally composable security without relying on an ideal cipher but incur additional exponentiations over uDH costs [3].

If OPRF is ever broken (by cryptanalysis, quantum attacks or security compromise), the user's password is exposed to an offline dictionary attack.

Near-optimal aPAKE. KHAPE addresses this weakness by dispensing with the OPRF (hence also improving performance). It uses a "paradoxical" mechanism that allows to directly encrypt credentials with the password and still prevent dictionary attacks. Two key ideas are: (i) dispense with authentication of the credentials[4] and instead use a non-committing encryption where decryption of a given ciphertext under different keys cannot help identify which key from a candidate set was used to produce that ciphertext; and (ii) using a *key-hiding* AKE. The latter refers to AKE protocols that require that no adversary, not even active one, can identify the long-term keys used by the peers to an exchange even if provided with a list of candidate keys (a notion reminiscent of key anonymity for public key encryption [8]).

Fortunately, many established AKE protocols are key hiding, including implicitly authenticated protocols such as 3DH [46] and HMQV [43], and KEM-based protocols with key-hiding KEMs (e.g., SKEME [41]). The non-committing property of encryption models symmetric encryption as an ideal model (similarly to the case of EKE discussed above) and allows for implementations based on random oracles with hash-to-curve operations to encode group elements as strings. As a result, KHAPE with HMQV, uses only one fixed-base exponentiation, one variable-base (multi)exponentiation for each party, and one hash-to-curve operation for the client. In all, it achieves computational overhead relative to *unauthenticated* Diffie-Hellman of *less than the cost of one exponentiation,* thus providing a close-to-optimal answer to our motivating questions above. Such computational performance compares favorably to that of other efficient aPAKE protocols such as SPAKE2+ and OPAQUE that incur overhead of one and two (variable-base) exponentiations, respectively, for server and client. In terms of number of messages, KHAPE uses 4 (3 if server initiates), compared to 3 messages in SPAKE2+ and OPAQUE.

Refer to Section 6 for a detailed description and rationale of the generic KHAPE protocol (compiling any key-hiding AKE into an aPAKE) and to Section 7 for the instantiation using HMQV.

On Strong aPAKE and reliance on OPRF. In the comparisons above, it is important to stress that OPAQUE achieves a *stronger notion of aPAKE*, the so called Strong aPAKE (saPAKE) model from [39]. In this model, the attacker that compromises a server can only start running an offline dictionary attack after breaking into the server. In contrast, in regular aPAKE, an offline attack is still needed but a specialized dictionary can be prepared ahead of time and used to find the password almost instantaneously when breaking into the server. KHAPE, as discussed above, does not provide this stronger security. However, as shown in [39], one can add a run of an OPRF to any aPAKE protocol to achieve

[4] Dispensing with authentication of credentials in OPAQUE completely breaks the protocol, allowing for trivial offline dictionary attacks.

Strong aPAKE security. If one does that to KHAPE, one gets a Strong aPAKE protocol with performance similar to that of OPAQUE (using HMQV or 3DH).

However, there is a significant difference in the reliance on the security of OPRF. While the password security of OPAQUE breaks down with a compromise of the OPRF key (namely, it allows for an offline dictionary attack on the password), in KHAPE the effect of compromising the OPRF is only to fall back to the (non-strong) aPAKE setting. In particular, this distinction is relevant in the context of quantum-safe cryptography as there are currently no known efficient OPRFs considered to be quantum safe. This opens a path to quantum-safe aPAKEs based on KHAPE with key hiding quantum-safe KEMs.

Closer comparison with OPAQUE. As stated above, KHAPE has an advantage over OPAQUE in terms of security due to its weaker reliance on OPRF and its computational advantage when the OPRF is not used. Also, KHAPE seems more conducive to post-quantum security via post-quantum key-hiding KEMs.[5] On the other hand, KHAPE requires one more message and allows for a more restrictive family of AKEs relative to OPAQUE (e.g., it does not allow for signature-based protocols as those based on SIGMA [42] and used in TLS 1.3 and IKEv2). KHAPE also relies for its analysis on the ideal cipher model while OPAQUE uses the random oracle model. An interesting advantage of KHAPE over OPAQUE is that in OPAQUE, an online attacker testing a password learns whether the password was wrong before the server does (in KHAPE the server learns first). This leads to a more complex mechanism for counting password failures at a server running OPAQUE, especially in settings with unreliable communication. Finally, we point out an advantage of using an OPRF with KHAPE (in addition to providing Strong aPAKE security): It allows for multi-server security via a threshold OPRF [38] where an attacker needs to break into multiple servers before it can run an offline attack on a password.

UC model analysis of (key-hiding) AKE's. All our protocols are framed and analyzed in the Universally Composable (UC) model [17]. This includes a formalization of the key-hiding AKE functionality that underlies the design of KHAPE. In order to instantiate KHAPE with specific AKE protocols, we prove that protocols 3DH [46] and HMQV [43] realize the key-hiding AKE functionality (in the ROM and under the Gap CDH assumption). We prove a similar result for SKEME [41] with appropriate KEM functions. We see the security analysis of these AKE protocols in the UC model, with and without key confirmation, as a contribution of independent interest. Moreover, the study of key-hiding AKE has applicability in other settings, e.g., where a gateway or IP address hides behind it other identities; say, a corporate site hosting employee identities or a web server aggregating different websites.

Organization. In Section 2, we define the notion of UC key-hiding AKE. In Sections 3 and 4, we show, respectively, that 3DH and HMQV, are secure UC

[5] We are currently investigating the use of NIST's post-quantum KEM selections [49] in conjunction with KHAPE.

key-hiding AKE protocols under the Gap DH assumption in ROM. In Section 5, we study the security of the SKEME protocol as a key-hiding AKE. In Section 6, we show a compiler from key-hiding AKE to asymmetric PAKE. In Section 7 we describe a concrete example of aPAKE, $\mathsf{KHAPE\text{-}HMQV}$, that instantiates KHAPE with HMQV as the key-hiding AKE. Finally, in Section 8 we survey potential ideal cipher instantiations and curve encodings based on quasi-bijections.

Appendices: In Appendix A we model standard UC AKE with entity authentication and show that adding key confirmation to a UC key-hiding AKE protocol converts it into a secure UC AKE-EA protocol. In Appendix B we recall the definition of UC aPAKE, with an extension to explicit client-to-server entity authentication. Finally, appendices C and D include proofs deferred from the main body.

2 The Key-Hiding AKE UC Functionality

Protocol KHAPE results from the composition of an encrypted credentials scheme and a *key-hiding* AKE protocol. Fig. 1 defines the UC functionality $\mathcal{F}_{\mathsf{khAKE}}$ that captures the properties required from a key-hiding AKE protocol. The modeling choices target the following requirements: First, as shown in Section 6, the security and key-hiding properties of this key-hiding AKE model suffice for our main application, a generic construction of UC aPAKE from any protocol realizing $\mathcal{F}_{\mathsf{khAKE}}$. Second, as shown in Section A, adding a standard key confirmation to any protocol that realizes $\mathcal{F}_{\mathsf{khAKE}}$ results in a (standard) UC AKE with explicit entity authentication. Lastly, this functionality is realized by several well-known and efficient AKE protocols, including 3DH and HMQV, as shown in Sections 3 and 4, as well as by a KEM-based AKE such as SKEME, if instantiated with a key-hiding KEM, see Section 5. We provide more details and rationale for the $\mathcal{F}_{\mathsf{khAKE}}$ next.

High-level requirements for key-hiding AKE. The most salient property we require from AKE is *key hiding*. To illustrate this requirement consider an experiment where the attacker $\mathcal{A}$ is provided with a transcript of a session between a party P and its counterparty CP. Party P has two inputs in this AKE instance: a public key pk_{CP} for CP and its own private key sk_{P} which P uses to authenticate to CP who presumably knows P's public key pk_{P}. In addition, $\mathcal{A}$ is given a pair of private keys: P's private key sk_{P} and a second random independent private key. $\mathcal{A}$'s goal is to decide which of the two keys P used in that session.[6] We are interested in AKE protocols where the attacker has no better chance to answer correctly than guessing randomly *even for sessions in which $\mathcal{A}$ is allowed to choose the messages from* CP.

The key hiding property will come up in the analysis of KHAPE as follows. The attacker learns a ciphertext c that encrypts the user's private key under

[6] This is reminiscent of key anonymity for PK encryption [8] where the attacker needs to distinguish between public keys for a given ciphertext.

– PK is the list of all public keys created via Init, initially empty
– PK_{P} is the list of all public keys created by P, initially empty for all P
– CPK is the list of all compromised keys in PK, initially empty

Keys: Initialization and Attacks

On Init from P:

Send $(\mathsf{Init}, \mathsf{P})$ to $\mathcal{A}$, let $\mathcal{A}$ specify pk s.t. $pk \notin PK$, add pk to PK and to PK_{P}, and output (Init, pk) to P

On $(\mathsf{Compromise}, \mathsf{P}, pk)$ from $\mathcal{A}$:

If $pk \in PK_{\mathsf{P}}$ then add pk to CPK

Login Sessions: Initialization and Attacks

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk, pk_{\mathsf{CP}})$ from P:

If $pk \in PK_{\mathsf{P}}$ and there is no prior session record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot \rangle$ then:
– create session record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, \perp \rangle$ marked fresh
– initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$
– send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ to $\mathcal{A}$

On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{A}$:

If session $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \perp \rangle$ is marked fresh then change its mark to $\mathsf{interfered}$

Login Sessions: Key Establishment

On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ from $\mathcal{A}$:

If $\exists$ session record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \perp \rangle$ then:
– if rec is marked fresh: If $\exists$ record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk_{\mathsf{CP}}, pk_{\mathsf{P}}, k' \rangle$ marked fresh s.t. $k' \neq \perp$ then set $k \leftarrow k'$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
– if rec is marked $\mathsf{interfered}$ then set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$
– update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, k \rangle$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$ to P

Session-Key Query

On $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$ from $\mathcal{A}$:

If $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, ... \rangle$ and $pk' \in CPK$ or $pk' \notin PK$, send $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ to $\mathcal{A}$

Fig. 1. $\mathcal{F}_{\mathsf{khAKE}}$: Functionality for Key-Hiding AKE the user's password. By decrypting this ciphertext under all passwords in a dictionary, the attacker obtains a set of possible private keys for the user. The key hiding property ensures that the attacker cannot identify the correct key (or the password) in the set. Fortunately, as we prove here, a large class of AKE protocols satisfy the key-hiding property, including implicitly authenticated protocols such as HMQV and 3DH, and some KEM-based protocols.

Additionally, $\mathcal{F}_{\mathsf{khAKE}}$ strengthens the basic guarantees of AKE protocols in several ways. It requires resilience to KCI (key-compromise impersonation) attacks, namely, upon the compromise of the private key of party P, the attacker can impersonate P to others but it cannot impersonate others to P. In the aPAKE setting, this ensures that an attacker that compromises a server, cannot impersonate the client to the server without going through an offline dictionary attack. In the context of key hiding AKE, we also need KCI resilience to prevent the attacker from authenticating to the client when given a set of possible private keys for that client.

Second, $\mathcal{F}_{\mathsf{khAKE}}$ requires that keys exchanged by a honest P with a corrupted CP still maintain a good amount of randomness, namely, the attacker can cause them to deviate from uniform but not by much (a property sometimes referred to as "contributive" key exchange, and not required in standard UC treatment). In the setting of protocol KHAPE, adversarial choice of session keys (particularly the ability of the attacker to create equal keys in different sessions) could lead to protocols where the attacker can test more than one password in a single session.

Properties that we do *not* consider as part of the $\mathcal{F}_{\mathsf{khAKE}}$ functionality, but will be provided by our final aPAKE protocol, KHAPE, include key confirmation, explicit authentication and full forward secrecy ($\mathcal{F}_{\mathsf{khAKE}}$ itself implies forward secrecy only against passive attackers).

Identities and public keys. We consider a setting where each party P has multiple public keys in the form of arbitrary handles pk. In the security model we assume that the public keys are arbitrary bitstrings chosen without loss of generality by the attacker (ideal adversary) $\mathcal{A}$, with the limitation that honest parties are assigned non-repeating pk strings. Pairs (P, pk) act as regular UC identities from the environment's point of view, but the pk component is concealed from $\mathcal{A}$ during key exchange sessions, even for sessions which are actively attacked by $\mathcal{A}$. This model can capture practical settings where P represents a gateway or IP address behind which other identities reside, e.g., a corporate site hosting employee identities or a web server aggregating different websites, and where one is interested to hide which party behind the gateway is communicating in a given session. Our specific application setting when using key-hiding AKE in the aPAKE construction of Section 6, is more abstract: The party symbols P, CP represent parties like internet clients and servers, while the multiplicity of public keys comes from decryptions of encrypted credentials under multiple password.

$(\mathsf{Compromise}, \mathsf{P}, pk)$. This adversarial action hands the (long-term) private key of party (P, pk) to the attacker $\mathcal{A}$. Such private-key leakage does not provide $\mathcal{A}$ with control over party P, and it does not even imply that the sessions which party P runs using the (leaked) key pk are insecure. However, when combined with the ability to run active attacks, via the $\mathsf{Interfere}$ action below, $\mathcal{A}$ can fully impersonate (P, pk) in sessions of $\mathcal{A}$'s choice. The leakage of the private key sk corresponding to (P, pk) does not affect the security of a session executed by party P even if it uses the compromised key pk. This captures the KCI property, i.e. that leakage of the private key of party P does not allow to impersonate others to party P. Also, any party P' which runs AKE with a *counterparty* identity specified as (P, pk), will also be secure as long as $\mathcal{A}$ does not actively interfere in that protocol. This captures the requirement that passively-observed AKE instance are secure regardless of the compromise of the long-term secrets used by either party. Note that $\mathcal{A}$ cannot compromise a party P but rather an identity pair (P, pk) and such compromise does not affect other pairs (P, pk').

$\mathsf{NewSession}$. A session is initiated by a party P that specifies its own identity pair (P, pk) as well as the intended counterparty identity pair $(\mathsf{CP}, pk_{\mathsf{CP}})$. Session identifiers sid are assumed to be unique within an honest party. The role of the initialized session-specific random function $R^{\mathsf{sid}}_{\mathsf{P}}$ is described below. A record for a session is initialized as fresh and is represented by a tuple $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, \perp \rangle$ where the last position, set to $\perp$, is reserved for recording the session key. An *essential element* in $\mathsf{NewSession}$ is that $\mathcal{A}$ learns $(\mathsf{sid}, \mathsf{P}, \mathsf{CP})$ but *it does not learn* (pk, pk_{CP}). In the real world this translates into the inability of the attacker to identify public (or private) keys associated to a pair of parties $(\mathsf{P}, \mathsf{CP})$ engaging in the Key-Hiding AKE protocol.

The functionality enforces that an honest P can start a session only on key pk which P generated and for which it holds a private key. However, the functionality does not check anything about the intended counterparty's identity $(\mathsf{CP}, pk_{\mathsf{CP}})$, so the private key corresponding to pk_{CP} could be held by party CP, or it could be held by a different party, or it could be compromised by the adversary, or it could be that pk_{CP} was not even generated by the key generation interface of $\mathcal{F}_{\mathsf{khAKE}}$, and it is an *adversarial* public key, whose private key the environment gave to the adversary. Our model thus includes honest parties who are tricked to use a wrong public key for the counterparty (e.g., via a phishing attack) in which case the attacker may know the corresponding private key. Note that regardless of what key pk_{CP} the session runs on, it is not given to the adversary, so if it is a key created by the envriment (i.e. a higher-level application which uses the key-hiding AKE) it does not necessarily follow that this key will be known to the adversary, and only in the case it is known the adversary will be able to attack that session using interfaces $\mathsf{Interfere}$, NewKey, and $\mathsf{SessionKey}$ below.

Function $R^{\mathsf{sid}}_{\mathsf{P}}$. When command $\mathsf{NewSession}$ creates a session for $(\mathsf{sid}, \mathsf{P})$ the functionality initializes a *random* function $R^{\mathsf{sid}}_{\mathsf{P}}$ specific to this session. Function $R^{\mathsf{sid}}_{\mathsf{P}}$ is used to set the value of the session key for sessions in which $\mathcal{A}$ actively interferes. It also allows $\mathcal{A}$ to have limited control over the value of the key under strict circumstances, namely it must know the pulic keys pk, pk_{CP} used on that session, and it must compromise party $(\mathsf{CP}, pk_{\mathsf{CP}})$. Even then the only freedom $\mathcal{A}$ has is to evaluate function $R^{\mathsf{sid}}_{\mathsf{P}}$ on any point α via a $\mathsf{SessionKey}$ query, see below, and then choose one such point in the NewKey caommand.

This captures the "contributive" property discussed above: If an honest party runs the AKE protocol even with adversary as a counterparty, the adversary's influence over the session key is limited to pre-computing polynomially-many random key candidates and then choosing one of them as a key on that session. The exact mechanics and functionality of $R_{\mathsf{P}}^{\mathsf{sid}}$ are defined in the NewKey and $\mathsf{SessionKey}$ actions below.

$(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$. This action represents an active attack on session $(\mathsf{P}, \mathsf{sid})$ and makes the session change its status from fresh to $\mathsf{interfered}$. The adversary does not have to know either P's own key pk or the intended counterparty key pk_{CP} which P uses on that session.[7] Such active atack will prevent session $(\mathsf{P}, \mathsf{sid})$ from establishing a secure key with any other honest party session, e.g. $(\mathsf{CP}, \mathsf{sid})$. It will also allow $\mathcal{A}$ to learn and/or influence the value of the session key this session outputs (using function $R_{\mathsf{P}}^{\mathsf{sid}}$), but only if in addition to being active $\mathcal{A}$ compromises the counterparty key $(\mathsf{CP}, pk_{\mathsf{CP}})$ used on session $(\mathsf{P}, \mathsf{sid})$.

NewKey. This action finalizes an AKE instance and makes $(\mathsf{P}, \mathsf{sid})$ output a session key. If the session is fresh then it receives either a fresh random key or the same key that was previously received by a matching session. If the session is $\mathsf{interfered}$, the value of the session key is determined by the function $R_{\mathsf{P}}^{\mathsf{sid}}$ on input $(pk, pk_{\mathsf{CP}}, \alpha)$ where α is chosen arbitrarily by $\mathcal{A}$, allowing $\mathcal{A}$ to influence the value of the session key (but in a very limited way as explained above). In the real-world, α represents transcript elements generated by the attacker, e.g., value Y an adversarial P_2 sends to an honest party P_1 in 3DH or HMQV.

$\mathsf{SessionKey}$. This action allows $\mathcal{A}$ to query the function $R_{\mathsf{P}}^{\mathsf{sid}}$ associated to a session $(\mathsf{sid}, \mathsf{P})$, potentially allowing $\mathcal{A}$ to learn and/or influence the session key for $(\mathsf{sid}, \mathsf{P})$. Note that learning any values of function $R_{\mathsf{P}}^{\mathsf{sid}}$ is useless unless the adversary actively attacks session $(\mathsf{sid}, \mathsf{P})$, because otherwise $R_{\mathsf{P}}^{\mathsf{sid}}$ is not used to determine the key output by session $(\mathsf{sid}, \mathsf{P})$. Moreover, $\mathcal{A}$ needs to provide $(pk, pk_{\mathsf{CP}}, \alpha)$ as input to $\mathsf{SessionKey}$, and if those inputs do not match P's own key pk and the intended counterparty key pk_{CP} which P uses on session $(\mathsf{sid}, \mathsf{P})$, then this query reveals an irrelevant value, since $R_{\mathsf{P}}^{\mathsf{sid}}$ is a random fuction. Finally, $\mathcal{F}_{\mathsf{khAKE}}$ releases value $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk_{\mathsf{CP}}, \alpha)$ to $\mathcal{A}$ only if key pk_{CP} is either compromised or adversarial. Summing up, the ability to learn (and/or control via the NewKey interface) the session key output by session $(\mathsf{sid}, \mathsf{P})$ is restricted to the case where all of the following hold: $\mathcal{A}$ actively interfered on that session, $\mathcal{A}$ guesses keys pk, pk_{CP} which this session uses, and $\mathcal{A}$ compromises counterparty's key $(\mathsf{CP}, pk_{\mathsf{CP}})$.

How $\mathcal{F}_{\mathsf{khAKE}}$ ensures key hiding and session security. The description of $\mathcal{F}_{\mathsf{khAKE}}$ is now complete. We now explain how $\mathcal{F}_{\mathsf{khAKE}}$ ensures the key hiding

[7] Currently functionality $\mathcal{F}_{\mathsf{khAKE}}$ assumes the ideal-world adversary $\mathcal{A}$ knows, and indeed creates, all honest parties' public keys. A tighter model is possible, if $\mathcal{F}_{\mathsf{khAKE}}$ samples public keys on behalf of honest players using the prescribed key generation algorithm, instead of letting $\mathcal{A}$ pick them. This would allow modeling use cases where the public keys are not public and are not freely available to the adversary.

property by which $\mathcal{A}$ cannot learn the value pk for an identity pair (P, pk) even if $\mathcal{A}$ knows P, has a list of all possible values of (P, pk), and actively interacts with (P, pk) using a compromised party $(\mathsf{CP}, pk_{\mathsf{CP}})$. Let's assume these conditions hold. Note that the only actions in which $\mathcal{A}$ can learn pk values from $\mathcal{F}_{\mathsf{khAKE}}$ are upon key generation and via the $\mathsf{SessionKey}$ call. Key generation assumes that $\mathcal{A}$ has a list of all possible values (P, pk). As we explain above, the only argument on which the value of function $R_{\mathsf{P}}^{\mathsf{sid}}$ is useful is a tuple $(pk, pk_{\mathsf{CP}}, \alpha)$ which the functionality uses to derive a session key for an actively attacked session $(\mathsf{sid}, \mathsf{P})$.

Consequently, the only way $\mathcal{F}_{\mathsf{khAKE}}$ can leak the session key output by $(\mathsf{sid}, \mathsf{P})$ is if $\mathcal{A}$ satisfies the three conditions above, i.e. it interferes in that session, key pk_{CP} used on that session is either compromised or adversarial, and $\mathcal{A}$ queries $\mathsf{SessionKey}$ on the proper keys pk, pk_{CP}. This is also the only way $\mathcal{A}$ can learn anything about keys pk, pk_{CP} used by session $(\mathsf{sid}, \mathsf{P})$: It has to attack the session, compromise pk_{CP}, get a session key candidate k^* via query $\mathsf{SessionKey}$ on pk, pk_{CP}, and then compare this key candidate against any information it has about the key k output by session $(\mathsf{sid}, \mathsf{P})$. For example, if P's higher-level application uses key k to MAC or encrypt a message, the adversary can verify the result against a candidate key k^* and thus learn whether $k^* = k$, and hence whether keys pk, pk_{CP} which $\mathcal{A}$ used to compute k^* were the same keys that were used by session $(\mathsf{sid}, \mathsf{P})$.

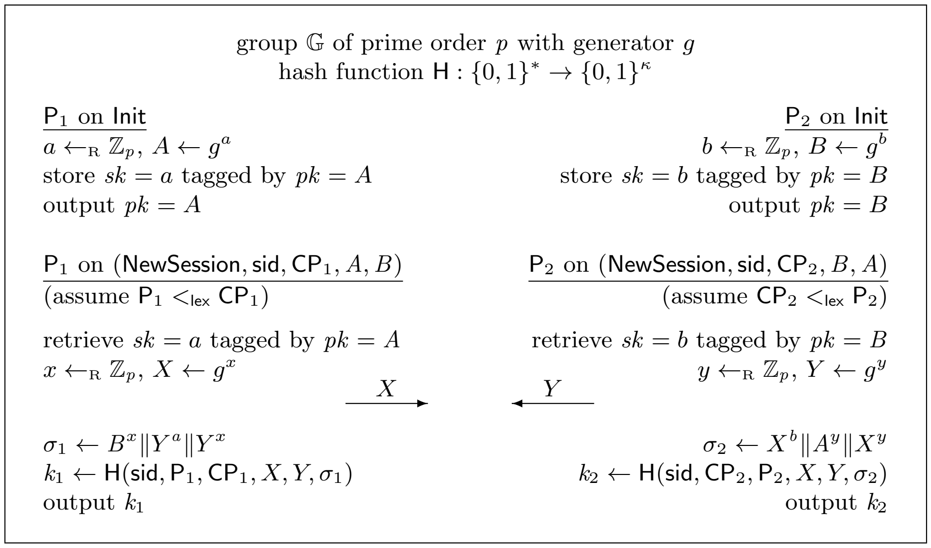
3 3DH as Key-Hiding AKE

We show that protocol 3DH, presented in Figure 2, realizes the UC notion of Key-Hiding AKE, as defined by functionality $\mathcal{F}_{\mathsf{khAKE}}$ in Section 2, under the Gap CDH assumption in ROM. As a consequence, 3DH can be used to instantiate protocol KHAPE in a simple and efficient way.

3DH [46] is a simple, implicitly authenticated key exchange used as the basis of the X3DH protocol [47] that underlies the Signal protocol. It consists of a plain Diffie-Hellman exchange authenticated via the session-key derivation that combines the ephemeral and long-term key of both peers. Specifically, if (a, A) and (b, B) are the long-term key pairs of two parties P_1 and P_2, and (x, X) and (y, Y) are their ephemeral DH values, then 3DH combines these key pairs to compute a (hash of) the *triple* of Diffie-Hellman values, $\sigma = g^{xb} \| g^{ay} \| g^{xy}$. Security of 3DH is intuitively easy to see: It follows from the fact that to compute σ the attacker must either (1) know (x, a) to attack party P_2 who uses A as a public key for its counterparty, or (2) know (y, b) to attack party P_1 who uses B as a public key for its counterparty. In other words, the attacker wins only if it is an active man-in-the-middle attacker *and* it compromises the key used as counterparty's public key by the attacked party. (Recall that "compromising a public key" stands for learning the corresponding private key.) The key-hiding property comes from the fact that the values X and Y exchanged in the protocol do not depend on long-term keys, and the fact that the only information about the long-term keys used by any party can be gleaned only from the session key they output and from H oracle queries on a σ value computed using these keys.

The formal proof of key-hiding in the UC model captures this argument, and we present it below.

We note that 3DH is not the most efficient key-hiding AKE. 3DH costs one fixed-base and three variable-base exponentiations per party, and in Section 4 we will show that HMQV, which preserves the bandwidth and round complexity of 3DH but folds the three variable-base exponentiations of 3DH into a single multi-exponentiation, realizes the key-hiding AKE functionality under the same Gap CDH assumption (although with worse exact security guarantees). However, HMQV can be seen as a modification of 3DH, and the security analysis of 3DH we show below will form a blueprint for the analysis of HMQV in Section 4.


group $\mathbb{G}$ of prime order p with generator g
hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$

$\underline{\mathsf{P}_1 \text{ on } \mathsf{Init}}$
$a \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $A \leftarrow g^a$
store $sk = a$ tagged by $pk = A$
output $pk = A$

$\underline{\mathsf{P}_2 \text{ on } \mathsf{Init}}$
$b \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $B \leftarrow g^b$
store $sk = b$ tagged by $pk = B$
output $pk = B$

$\underline{\mathsf{P}_1 \text{ on } (\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_1, A, B)}$
(assume $\mathsf{P}_1 <_{\mathsf{lex}} \mathsf{CP}_1$)

retrieve $sk = a$ tagged by $pk = A$
$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $X \leftarrow g^x$

$\underline{\mathsf{P}_2 \text{ on } (\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_2, B, A)}$
(assume $\mathsf{CP}_2 <_{\mathsf{lex}} \mathsf{P}_2$)

retrieve $sk = b$ tagged by $pk = B$
$y \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $Y \leftarrow g^y$

$\xrightarrow{X}$ $\xleftarrow{Y}$

$\sigma_1 \leftarrow B^x \| Y^a \| Y^x$
$k_1 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{P}_1, \mathsf{CP}_1, X, Y, \sigma_1)$
output k_1

$\sigma_2 \leftarrow X^b \| A^y \| X^y$
$k_2 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{CP}_2, \mathsf{P}_2, X, Y, \sigma_2)$
output k_2

Fig. 2. Protocol $3DH$: "Triple Diffie-Hellman" Key Exchange

Conventions.

(1) In Figure 2 we assume that each party runs 3DH using key pair (sk, pk) previously generated via procedure Init. In Figure 2 these are resp. (a, A) for P_1 and (b, B) for P_2. Note that no such requirement is posed on the counterparty public key each party uses, resp. public key B used by P_1 and A used by P_2.

(2) We implicitly assume that each party P_i uses its own identity as a protocol input, together with the identity CP_i of its assumed counterparty. These identities could be e.g. domain names, user names, or any other identifiers. They have no other semantics except that the two parties can establish the same session key only if they assume matching identifiers, i.e. $(\mathsf{P}_1, \mathsf{CP}_1) = (\mathsf{CP}_2, \mathsf{P}_2)$.

(3) Protocol 3DH is symmetric except for the ordering of group elements in tuple σ and the ordering of elements in the inputs to hash H. Each protocol party P

can locally determine this order based on whether string P is lexicographically smaller than string CP. (In Figure 2 we assume that $\mathsf{P}_1 <_{\mathsf{lex}} \mathsf{P}_2$.) An equivalent way to see it is that each party P computes a "role" bit $\mathsf{role} \in \{1,2\}$ and follows the protocol of party $\mathsf{P}_{\mathsf{role}}$ in Figure 2: Party P sets this bit as $\mathsf{role} = 1$, called the "client role", if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$, and $\mathsf{role} = 2$, called the "server role", otherwise.

(4) We assume that parties verify public keys and ephemeral DH values, resp. B, Y for P_1 and A, X for P_2, as group $\mathbb{G}$ elements. Optionally, instead of group membership testing one can use cofactor exponentiation to compute σ.

Cryptographic Setting: Gap CDH and RO Hash. Let g generate a cyclic group $\mathbb{G}$ of prime order p. The Computational Diffie-Hellman (CDH) assumption on $\mathbb{G}$ states that given $(X,Y) = (g^x, g^y)$ for $(x,y) \leftarrow_{\mathrm{R}} (\mathbb{Z}_p)^2$ it is hard to find $\mathsf{cdh}_g(X,Y) = g^{xy}$. The Gap CDH assumption states that CDH is hard even if the adversary has access to a Decisional Diffie-Hellman oracle ddh_g, which on input (A,B,C) returns 1 if $C = \mathsf{cdh}_g(A,B)$ and 0 otherwise.

Theorem 1. *Protocol 3DH shown in Figure 2 realizes $\mathcal{F}_{\mathsf{khAKE}}$ if the Gap CDH assumption holds and H is a random oracle.*

```
Initialization: Initialize an empty list $KL_{\mathsf{P}}$ for each $\mathsf{P}$

On $(\mathsf{Init}, \mathsf{P})$ from $\mathcal{F}$:
pick $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $pk \leftarrow g^{sk}$, add $(sk, pk)$ to $KL_{\mathsf{P}}$, and send $pk$ to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ send $sk$ to $\mathcal{A}$ and $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w \rangle$, send $W = g^w$ to $\mathcal{A}$

On $\mathcal{A}$'s message $Z$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, w \rangle$:
    if $\exists$ no record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, z \rangle$ s.t. $g^z = Z$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
    send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, Z)$ to $\mathcal{F}$

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ to random oracle $\mathsf{H}$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and:
if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x \rangle$ and $(a, A) \in KL_{\mathsf{C}}$ s.t. $(X, \sigma) = (g^x, (B^x \| Y^a \| Y^x))$ for
some $B$, send $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{C}, A, B, Y)$ to $\mathcal{F}$, if $\mathcal{F}$ returns $k^*$ reset $k \leftarrow k^*$
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, y \rangle$ and $(b, B) \in KL_{\mathsf{S}}$ s.t. $(Y, \sigma) = (g^y, (X^b \| A^y \| X^y))$ for
some $A$, send $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{S}, B, A, X)$ to $\mathcal{F}$, if $\mathcal{F}$ returns $k^*$ reset $k \leftarrow k^*$
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$
```

Fig. 3. Simulator SIM showing that 3DH realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$")

```
*Initialization:* Initialize an empty list $KL_{\mathsf{P}}$ for each $\mathsf{P}$

On message $\mathsf{Init}$ to $\mathsf{P}$:
pick $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $pk \leftarrow g^{sk}$, add $(sk, pk)$ to $KL_{\mathsf{P}}$, and output $(\mathsf{Init}, pk)$

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ then output $sk$

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathsf{P}$:
if $\exists\ (sk, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$, pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, sk, pk_{\mathsf{CP}}, w \rangle$, output $W = g^w$

On message $Z$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, sk_{\mathsf{P}}, pk_{\mathsf{CP}}, w \rangle$, set $\sigma \leftarrow ((pk_{\mathsf{CP}})^w \| Z^{sk_{\mathsf{P}}} \| Z^w)$,
$k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\mathrm{ord}})$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On $\mathsf{H}$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and:
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$
```

Fig. 4. 3DH: Environment's view of real-world interaction (Game 0)

Proof Overview. We show that that for any efficient environment algorithm $\mathcal{Z}$, its view of the *real-world* security game, i.e. an interaction between the real-world adversary and honest parties who follow protocol 3DH, is indistinguishable from its view of the *ideal-world* game, i.e. an interaction between the ideal-world adversary, whose role is played by the *simulator*, with the functionality $\mathcal{F}_{\mathsf{khAKE}}$. We show the simulator algorithm SIM in Figure 3. The real-world game, Game 0, is shown in Figure 4, and the ideal-world game defined by a composition of algorithm SIM and functionality $\mathcal{F}_{\mathsf{khAKE}}$, denoted Game 7, is shown in Figure 5.

As is standard, we assume that the real-world adversary $\mathcal{A}$ is a subroutine of the environment $\mathcal{Z}$, therefore the sole party that interacts with Games 0 or 7 is $\mathcal{Z}$, issuing commands Init and $\mathsf{NewSession}$ to honest parties P, adaptively compromising public keys, and using $\mathcal{A}$ to send protocol messages Z to honest party's sesssions and making hash function H queries. The proof follows a standard strategy of showing a sequence of games that bridge between Game 0 and Game 7, where at each transition we argue that the change is indistinguishable. We use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i, and the theorem follows if we show that $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G7}]|$ is negligible under the stated assumptions.

Notation. To make the real-world interaction in Figure 4 more concise, we adopt a notation which stresses the symmetric nature of 3DH protocol: We use variable $W = g^w$ to denote the message which party P sends out, and variable Z to denote the message it receives, e.g. $(W, Z) = (X, Y)$ if P plays the "client" role and $(W, Z) = (Y, X)$ if P plays the "server" role. If $\sigma = \sigma_1 \| \sigma_2 \| \sigma_3$ then let $\{\sigma\}_{\mathrm{flip}} = \sigma_2 \| \sigma_1 \| \sigma_3$. We will use $\{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\mathrm{ord}}$ to denote string $\mathsf{P}, \mathsf{CP}, W, Z, \sigma$ if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ or string $\mathsf{CP}, \mathsf{P}, Z, W, \{\sigma\}_{\mathrm{flip}}$ if $\mathsf{CP} <_{\mathsf{lex}} \mathsf{P}$. With this notation each party's 3DH protocol code can be restated in the symmetric way, as in Figure 4, because session key computation of party P can be denoted in a uniform way as $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\mathrm{ord}})$ for $\sigma = (pk_{\mathsf{CP}})^w \| Z^{sk_{\mathsf{P}}} \| Z^w$.

We use the same symmetric notation to describe simulator SIM in Figure 3 and the ideal-world game implied by SIM and $\mathcal{F}_{\mathsf{khAKE}}$ in Figure 5, except for the way SIM treats H oracle queries, which we separate into two cases based on the roles played by the two parties whose sessions are potentially involved in any H query. In H-handling code of SIM we denote the identifiers of the two parties involved in a query as C and S, for the parties playing respectively the client and server roles, and the code that follows uses role-specific notation to handle attacks on the sessions executed respectively by C and S.

Throughout the proof we use $\mathsf{P}^{\mathsf{sid}}$ to denote a session of party P with identifier sid. We use $v_{\mathsf{P}}^{\mathsf{sid}}$ to denote a local variable v pertaining to session $\mathsf{P}^{\mathsf{sid}}$ or a message v which this session receives, and whenever identifier sid is clear from the context we write v_{P} instead of $v_{\mathsf{P}}^{\mathsf{sid}}$. Note that session $\mathsf{CP}^{\mathsf{sid}}$ is uniquely defined for every session $\mathsf{P}^{\mathsf{sid}}$ by setting $\mathsf{CP} = \mathsf{CP}_{\mathsf{P}}^{\mathsf{sid}}$, and we will implicitly assume below that a counterparty's session is defined in this way.

For a fixed environment $\mathcal{Z}$, let q_{K} and q_{ses} be (the upper-bounds on) the number of resp. keys and sessions initialized by $\mathcal{Z}$, let q_{H} be the number of H oracle queries $\mathcal{Z}$ makes, and let $\epsilon_{\text{g-cdh}}^{\mathcal{Z}}$ be the maximum advantage in solving Gap CDH in $\mathbb{G}$ of an algorithm that makes q_{H} DDH oracle queries and uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{H}} + q_{\mathrm{ses}})$ exponentiations in $\mathbb{G}$.

Define the following two functions for every session $\mathsf{P}^{\mathsf{sid}}$:

$$\mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z) = \mathsf{cdh}_g(W, pk') \| \mathsf{cdh}_g(pk, Z) \| \mathsf{cdh}_g(W, Z) \text{ for } W = W_{\mathsf{P}}^{\mathsf{sid}} \quad (1)$$

$$R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z) = \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}_{\mathsf{P}}^{\mathsf{sid}}, W_{\mathsf{P}}^{\mathsf{sid}}, Z, \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)\}_{\mathrm{ord}}) \quad (2)$$

If session $\mathsf{P}^{\mathsf{sid}}$ runs on its own private key sk_{P}, counterparty's public key pk_{CP}, and receives message Z, then its output session key is $k = R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ for $pk_{\mathsf{P}} = g^{sk_{\mathsf{P}}}$. Note also that an adversary can locally compute function $R_{\mathsf{P}}^{\mathsf{sid}}$ for any pk_{P}, any key pk_{CP} which was either generated by the adversary or it was generated by an honest party but it has been compromised, and any Z which the adversary generates, because the adversary can then compute functions $\mathsf{cdh}_g(\cdot, pk_{\mathsf{CP}})$ and $\mathsf{cdh}_g(\cdot, Z)$ on any inputs.

Simulator. Simulator SIM, shown in Figure 3, picks all (sk, pk) pairs on behalf of honest players and surrenders the corresponding private key whenever an honestly-generated public key is compromised. To simulate honest party P behavior the simulator sends $W = g^w$ for random w. When $\mathsf{P}^{\mathsf{sid}}$ receives Z the simulator forks: If Z originated from honest session $\mathsf{CP}^{\mathsf{sid}}$ which runs on matching identifiers $(\mathsf{sid}, \mathsf{CP}, \mathsf{P})$, SIM treats this as a case of honest-but-curious attack that connects two potentially matching sessions and sends NewKey to $\mathcal{F}_{\mathsf{khAKE}}$. ($Z$ included in this call is ignored by $\mathcal{F}_{\mathsf{khAKE}}$.) Otherwise SIM treats it as an active attack on $\mathsf{P}^{\mathsf{sid}}$ and sends $\mathsf{Interfere}$ followed by $(\mathsf{NewKey}, ..., Z)$. Note that in response $\mathcal{F}_{\mathsf{khAKE}}$ will treat $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{interfered}$ and set its output key as $k \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the (own,counterparty) pair of public keys which $\mathsf{P}^{\mathsf{sid}}$ uses, and which is unknown to SIM (except if pk_{CP} was generated by the adversary, in which case it was leaked to SIM at $\mathsf{NewSession}$). Finally, SIM services H oracle queries $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ by identifying those that pertain to viable session-key computations by either session $\mathsf{C}^{\mathsf{sid}}$ or $\mathsf{S}^{\mathsf{sid}}$. We describe it here only for $\mathsf{C}^{\mathsf{sid}}$-side H queries since $\mathsf{S}^{\mathsf{sid}}$-side queries are handled symmetrically. If H query involves $\sigma = \mathsf{3DH}^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$ for some A, B s.t. (1) A is one of the public keys generated by C, and (2) B is either some compromised honestly generated public key or it is an adversarial key which $\mathsf{C}^{\mathsf{sid}}$ uses for the counterparty (recall that if $\mathsf{C}^{\mathsf{sid}}$ runs on an adversary-generated counterparty key pk_{CP} then functionality $\mathcal{F}_{\mathsf{khAKE}}$ leaks it to the adversary), then SIM treats that query as a potential computation of a session key output by $\mathsf{C}^{\mathsf{sid}}$, queries $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{C}, A, B, Y)$ to $\mathcal{F}_{\mathsf{khAKE}}$. If B is compromised or adverarial then $\mathcal{F}_{\mathsf{khAKE}}$ responds with $k^* \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$ and SIM embeds k^* into H output. Note that if (A, B) matches the (own,counterparty) keys used by $\mathsf{C}^{\mathsf{sid}}$, and $\mathsf{C}^{\mathsf{sid}}$ receives $Z = Y$ in the protocol, then k^* will match the session key output by $\mathsf{C}^{\mathsf{sid}}$. For all other triples (A, B, Y) the outputs of $R^{\mathsf{sid}}_{\mathsf{C}}$ are irrelevant except that (1) if the adversary learns the real session key output by $\mathsf{C}^{\mathsf{sid}}$ then these H outputs inform the adversary that pair (A, B) is *not* the (own,counterparty) key pair used by $\mathsf{C}^{\mathsf{sid}}$, and (2) if the adversary bets on some (A, B) pair used by $\mathsf{C}^{\mathsf{sid}}$ then it can use H queries to find an "optimal" protocol response Y to $\mathsf{C}^{\mathsf{sid}}$ for which the resulting (randomly sampled) session key has some properties the adversary likes, e.g. its last 20 bits are all zeroes, etc.

Game Sequence from Game 0 to Game 7.

GAME 0 *(real world)*: This is the interaction of environment $\mathcal{Z}$ (and its subroutine, the real-world adversary) with protocol 3DH, as shown in Fig. 4.

GAME 1 *(past* H *queries are irrelevent to new sessions)*: Game 1 adds an abort if $\mathsf{NewSession}$ initializes session $\mathsf{P}^{\mathsf{sid}}$ with $W = g^w$ s.t. H has been queried on any tuple of the form $(\mathsf{sid}, \{\mathsf{P}, \cdot, W, \cdot, \cdot\}_{\text{ord}})$. Since each H query can pertain to at most two sessions, $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$, there at most q_{H} such queries, and $w \leftarrow_{\text{R}} \mathbb{Z}_p$, we have:

$$|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (2q_{\mathsf{H}})/p$$

GAME 2 *(programming* $R^{\mathsf{sid}}_{\mathsf{P}}$ *values into* H *outputs)*: Define sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ to be *matching* if $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{C}} = \mathsf{S}$ and $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{S}} = \mathsf{C}$. Note that for any matching sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ and any public keys A, B correctness of 3DH implies that $R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y_{\mathsf{S}}) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X_{\mathsf{C}})$. While in equation (2) we defined function $R^{\mathsf{sid}}_{\mathsf{P}}$ in terms of hash H, in Game 2 we set H outputs using appropriately chosen functions $R^{\mathsf{sid}}_{\mathsf{P}}$. For every pair of matching sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ consider a pair of random functions $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}} : (\mathbb{G})^3 \to \{0,1\}^\kappa$ s.t.

$$R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y^{\mathsf{sid}}_{\mathsf{S}}) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X^{\mathsf{sid}}_{\mathsf{C}}) \quad \text{for all } A, B \in \mathbb{G} \tag{3}$$

More precisely, for any session $\mathsf{P}^{\mathsf{sid}}$ with no matching session $R^{\mathsf{sid}}_{\mathsf{P}}$ is set as a random function, and for $\mathsf{P}^{\mathsf{sid}}$ for which a prior matching session exists $R^{\mathsf{sid}}_{\mathsf{P}}$ is set as a random function subject to constraint (3). Let PK be the list of all

```
*Initialization:* Initialize empty lists: $PK$, $CPK$, and $KL_{\mathsf{P}}$ for all $\mathsf{P}$

On message $\mathsf{Init}$ to $\mathsf{P}$:
set $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $pk \leftarrow g^{sk}$, send $(\mathsf{Init}, pk)$ to $\mathsf{P}$, add $pk$ to $PK$ and $(sk, pk)$ to $KL_{\mathsf{P}}$

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ add $pk$ to $CPK$ and output $sk$

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathsf{P}$:
if $\exists\ (sk, pk_{\mathsf{P}}) \in KL_P$ then:
    initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \rightarrow \{0,1\}^\kappa$
    if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
    pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$ as $\mathsf{fresh}$, output $W = g^w$

On message $Z$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$:
    if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \mathsf{role}', z, k' \rangle$ s.t. $g^z = Z$
        then if $\mathsf{rec}'$ is $\mathsf{fresh}$, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$, and $k' \neq \perp$:
            then $k \leftarrow k'$
            else $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
        else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ and re-label $\mathsf{rec}$ as $\mathsf{interfered}$
    update $\mathsf{rec}$ to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, k \rangle$, send $(\mathsf{NewKey}, \mathsf{sid}, k)$ to $\mathsf{P}$

On $\mathsf{H}$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$ and:
  1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, \cdot \rangle$ s.t. $(X, \sigma) = (g^x, (B^x \| Y^a \| Y^x))$ for some
     $(a, A) \in KL_{\mathsf{C}}$ and $B$ s.t. $B \in CPK$ or $B \notin PK$ then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$
  2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, y, \cdot \rangle$ s.t. $(Y, \sigma) = (g^y, (X^b \| A^y \| X^y))$ for some
     $(b, B) \in KL_{\mathsf{S}}$ and $A$ s.t. $A \in CPK$ or $A \notin PK$ then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$
```

Fig. 5. 3DH: Environment's view of ideal-world interaction (Game 7) public keys generated so far, and PK_{P} be the set of keys generated for P. Let $PK^{+}(\mathsf{P}^{\mathsf{sid}})$ stand for $PK \cup \{pk_{\mathsf{CP}}\}$ where pk_{CP} is the counterparty public key used by $\mathsf{P}^{\mathsf{sid}}$. (If $pk_{\mathsf{CP}} \in PK$ then $PK^{+}(\mathsf{P}^{\mathsf{sid}}) = PK$.) Consider an oracle H which responds to each new query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ for $\mathsf{C} <_{\mathsf{lex}} \mathsf{S}$ as follows:

1. If $\exists\, \mathsf{C}^{\mathsf{sid}}$ s.t. $(\mathsf{S}, X) = (\mathsf{CP}_{\mathsf{C}}^{\mathsf{sid}}, X_{\mathsf{C}}^{\mathsf{sid}})$, and $\exists\, A, B$ s.t. $A \in PK_{\mathsf{C}}$, $B \in PK^{+}(\mathsf{C}^{\mathsf{sid}})$, and $\mathsf{3DH}_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y) = \sigma$, then set $k \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y)$
2. If $\exists\, \mathsf{S}^{\mathsf{sid}}$ s.t. $(\mathsf{C}, Y) = (\mathsf{CP}_{\mathsf{S}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}})$, and $\exists\, B, A$ s.t. $B \in PK_{\mathsf{S}}$, $A \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$, and $\mathsf{3DH}_{\mathsf{S}}^{\mathsf{sid}}(B, A, X) = \{\sigma\}_{\text{flip}}$, then set $k \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B, A, X)$
3. In any other case sample $k \leftarrow_{\text{R}} \{0,1\}^{\kappa}$

Since the game knows each key pair $(sk_{\mathsf{P}}, pk_{\mathsf{P}})$ generated for each P, and the ephemeral state w of each session $\mathsf{P}^{\mathsf{sid}}$, it can decide for any Z, pk' if $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk', Z) = (pk')^{w} \| Z^{sk_{\mathsf{P}}} \| Z^{w}$. Note that each value of $R_{\mathsf{P}}^{\mathsf{sid}}$ is used to program H on at most one query. Also, if H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ satisfies both conditions then $(X, Y) = (X_{\mathsf{C}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}}) = (g^{x}, g^{y})$ and $\exists\, A', B', a, b$ s.t.

$$\mathsf{3DH}_{\mathsf{C}}^{\mathsf{sid}}(g^{a}, B', Y) = (B')^{x} \| Y^{a} \| Y^{x} = X^{b} \| (A')^{y} \| X^{y} = \{\mathsf{3DH}_{\mathsf{S}}^{\mathsf{sid}}(g^{b}, A', X)\}_{\text{flip}}$$

Since these equations imply that $(A', B') = (g^{a}, g^{b})$, and by equation (3), $R_{\mathsf{C}}^{\mathsf{sid}}(A', B', Y_{\mathsf{S}}^{\mathsf{sid}}) = R_{\mathsf{S}}^{\mathsf{sid}}(B', A', X_{\mathsf{C}}^{\mathsf{sid}})$, it follows that if both conditions are satisfied then both will program H output to the same value. Thus we conclude:

$$\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]$$

Game 3 *(direct programming of session keys using random functions $R_{\mathsf{P}}^{\mathsf{sid}}$)*: In Game 3 we make the following changes: We mark each initialized session $\mathsf{P}^{\mathsf{sid}}$ as fresh, and when $\mathcal{A}$ sends Z to $\mathsf{P}^{\mathsf{sid}}$ then we re-label $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{interfered}$ if Z does not equal to the message sent by the matching session $\mathsf{CP}^{\mathsf{sid}}$, i.e. if $Z_{\mathsf{P}}^{\mathsf{sid}} \neq W_{\mathsf{CP}}^{\mathsf{sid}}$. Secondly, if session $\mathsf{P}^{\mathsf{sid}}$ runs on its own key pair $(sk_{\mathsf{P}}, pk_{\mathsf{P}})$ and intended counterparty public key pk_{CP}, we say that it runs "under keys $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$". Using this book-keeping, Game 3 modifies session-key computation for session $\mathsf{P}^{\mathsf{sid}}$ which runs under keys $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ as follows:

1. If $k_{\mathsf{CP}}^{\mathsf{sid}} \neq \perp$, sessions $\mathsf{P}^{\mathsf{sid}}, \mathsf{CP}^{\mathsf{sid}}$ are fresh and *matching*, and $\mathsf{CP}^{\mathsf{sid}}$ runs under keys $(pk_{\mathsf{CP}}, pk_{\mathsf{P}})$, then $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow k_{\mathsf{CP}}^{\mathsf{sid}}$
2. In any other case set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$.

We argue that this change makes no difference to the environment. In Game 2 the session key $k_{\mathsf{P}}^{\mathsf{sid}}$ is computed as $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\text{ord}})$ for $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$. However, H on such input is programmed in Game 2 to output $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ if $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ for any $pk_{\mathsf{CP}} \in PK^{+}(\mathsf{P}^{\mathsf{sid}})$. Since pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ must be in set $PK^{+}(\mathsf{P}^{\mathsf{sid}})$, setting $k_{\mathsf{P}}^{\mathsf{sid}}$ directly as $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ only short-circuits this process. Moreover, since $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ are correlated by equation (3), setting $k_{\mathsf{C}}^{\mathsf{sid}}$ as $k_{\mathsf{S}}^{\mathsf{sid}}$ or vice versa, in the case both are fresh, i.e. $Z_{\mathsf{C}}^{\mathsf{sid}} = Y_{\mathsf{S}}^{\mathsf{sid}}$ and $Z_{\mathsf{S}}^{\mathsf{sid}} = X_{\mathsf{C}}^{\mathsf{sid}}$, and sessions

$\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ run under matching keys, resp. $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (A, B)$ and $(pk_{\mathsf{CP}}, pk_{\mathsf{P}}) = (B, A)$, also does not change the game. Thus we conclude:

$$\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$$

Game 4 *(abort on session-key derivation* H *query for passive sessions)*: We add an abort if oracle H triggers evaluation of $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ for any pk, pk' and $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$ where $\mathsf{CP}^{\mathsf{sid}}$ is a matching session of $\mathsf{P}^{\mathsf{sid}}$. Note that if $\mathsf{P}^{\mathsf{sid}}$ is passively observed, i.e. it remains fresh then value $W_{\mathsf{CP}}^{\mathsf{sid}}$ either has been delivered to $\mathsf{P}^{\mathsf{sid}}$, i.e. $Z_{\mathsf{P}}^{\mathsf{sid}} = W_{\mathsf{CP}}^{\mathsf{sid}}$, or $\mathsf{P}^{\mathsf{sid}}$ is still waiting for message Z. By the code of oracle H in Game 2 the call to $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', W_{\mathsf{CP}}^{\mathsf{sid}})$ is triggered only if H query $(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\mathrm{ord}})$ satisfies the following for $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$ and $W = W_{\mathsf{P}}^{\mathsf{sid}}$:

$$\sigma \;=\; \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z) \;=\; \mathsf{cdh}_g(W, pk') \,\|\, \mathsf{cdh}_g(pk, Z) \,\|\, \mathsf{cdh}_g(W, Z)$$

Hardness of computing such tuple relies on the hardness of computing its last element, i.e. $\mathsf{cdh}_g(W, Z)$, because (W, Z) are Diffie-Hellman KE messages sent by honest sessions $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$. We show that solving Gap CDH can be reduced to causing event Bad, defined as the event that adversary makes such H query. Reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{Y})$ and embeds it in the messages of simulated parties: If $\mathsf{role} = 1$ then $\mathcal{R}$ sends $X = \bar{X}^s$ for $s \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ as the message from $\mathsf{C}^{\mathsf{sid}}$, and if $\mathsf{role} = 2$ then $\mathcal{R}$ sends $Y = \bar{Y}^t$ for $t \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ as the message from $\mathsf{S}^{\mathsf{sid}}$. Finally, $\mathcal{R}$ responds to Init by generating keys $(sk_{\mathsf{P}}, pk_{\mathsf{P}})$ as in Game 0.

$\mathcal{R}$ does not know $x = s \cdot \bar{x}$ and $y = t \cdot \bar{y}$ corresponding to messages X, Y, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{y} = \mathsf{dlog}_g(\bar{Y})$, but it can use the DDH oracle to emulate the way Game 3 services H queries: To test if H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ for $X = \bar{X}^s$ satisfies $\sigma = (L \| M \| N) = (pk^x \| Y^a \| Y^x)$ for $x = s \cdot \bar{x}$ and any a, pk, reduction $\mathcal{R}$ checks if $L = \mathsf{cdh}_g(\bar{X}, pk^s)$, $M = Y^a$, and $N = \mathsf{cdh}_g(\bar{X}, Y^s)$. Symmetrically, $\mathcal{R}$ tests if (X, Y, σ) for $Y = \bar{Y}^t$ satisfies $\sigma = (M \| L \| N) = (X^b \| pk^y \| X^y)$ by checking if $L = \mathsf{cdh}_g(\bar{Y}, pk^t)$, $M = X^b$, and $N = \mathsf{cdh}_g(\bar{Y}, X^t)$.

Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3. If it does then $\mathcal{R}$ detects it by checking if the last element N in σ satisfies $N = \mathsf{cdh}_g(W, Z)$ for $W = W_{\mathsf{P}}^{\mathsf{sid}}$ and $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$. If $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ are matching then one of them plays the client role and the other the server role, i.e. either (W, Z) or (Z, W) is equal to $(\bar{X}^s, \bar{Y}^t)$ for some s, t known by $\mathcal{R}$. In either case $\mathcal{R}$ can output $N^{1/(st)}$ as the answer $\mathsf{cdh}_g(\bar{X}, \bar{Y})$ to its CDH challenge. It follows that $\Pr[\mathsf{Bad}] \leq \epsilon_{\text{g-cdh}}^{\mathcal{Z}}$, hence:

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon_{\text{g-cdh}}^{\mathcal{Z}}$$

Game 5 *(random keys on passively observed sessions)*: We modify the game so that if session $\mathsf{P}^{\mathsf{sid}}$ remains fresh when $\mathcal{A}$ sends Z to $\mathsf{P}^{\mathsf{sid}}$ then instead of setting $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ as in Game 3, we now set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$. Since session $\mathsf{P}^{\mathsf{sid}}$ can remain fresh only if Z it receives was sent by its matching session, i.e. $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$, and by Game 4 oracle H never queries $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ for such

Z, it follows by randomness of $R_{\mathsf{P}}^{\mathsf{sid}}$ that the modified game remains externally identical, hence:

$$\Pr[\mathsf{G5}] = \Pr[\mathsf{G4}]$$

GAME 6 *(decorrelating function pairs $R_{\mathsf{C}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}}$)*: Let Game 6 be as Game 5, except that functions $R_{\mathsf{S}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}}$ are chosen without the constraint imposed by equation (3). Since by Game 5 neither function is queried on the points which create the correlation imposed by equation (3), it follows that:

$$\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$$

GAME 7 *(hash computation consistent only for compromised keys)*: Recall that in Game 6, as in Game 2, $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W_{\mathsf{P}}^{\mathsf{sid}}, Z, \sigma\}_{\text{ord}})$ is defined as $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ if $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ for some $pk \in PK_{\mathsf{P}}$, and $pk' \in PK^{+}(\mathsf{P}^{\mathsf{sid}})$. In Game 7 we add a condition that this programming of H can occur only if either (1) pk' is an honestly generated key of some party, but it has been $\mathsf{compromised}$ or (2) pk' is the counterparty key which session $\mathsf{P}^{\mathsf{sid}}$ runs under, and it is an *adverarial* key, i.e. it has not been generated by Init. Note that these are the two cases in which the adversary can know the secret key corresponding to pk', and we will show that this knowledge is indeed necessary for adversary to compute σ s.t. $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$.

Let CPK be the list of generated public keys who were compromised so far, and let $CPK^{+}(\mathsf{P}^{\mathsf{sid}})$ stand for CPK if the counterparty public key pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ is an honestly generated key, and for $CPK \cup \{pk_{\mathsf{CP}}\}$ if pk_{CP} is adversarially-generated. The modification of Game 7 is that H output is programmed to $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ for pk' s.t. $\sigma = \mathsf{3DH}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ only if $pk' \in CPK^{+}(\mathsf{P}^{\mathsf{sid}})$, while in Game 6, as in Game 2, this programming was done whenever $pk' \in PK^{+}(\mathsf{P}^{\mathsf{sid}})$. Therefore the two games diverge in the case of event Bad defined as H query as above for $pk' \in PK \setminus CPK$, i.e. honestly generated and *not* compromised key. Let Bad_n be Bad where $\mathsf{P}^{\mathsf{sid}}$ plays role $= n$. We show a reduction $\mathcal{R}$ that solves Gap CDH if Bad_1 occurs. The argument for event Bad_2 is symmetrical.

Note that Bad_1 corresponds to H query on string $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$ for $\sigma = (B^x \| Y^a \| Y^x)$ where $x = x_{\mathsf{C}}^{\mathsf{sid}}$, a is some private key of C, and B is a non-compromised public key in PK (not necessarily owned by S). On input a CDH challenge $(\bar{X}, \bar{B})$, $\mathcal{R}$ sets each $X_{\mathsf{C}}^{\mathsf{sid}}$ as $\bar{X}^s$ for random s, just like the reduction in Game 4, but it sets each $Y_{\mathsf{S}}^{\mathsf{sid}}$ as g^y for random y. $\mathcal{R}$ also picks all keys $(sk_{\mathsf{P}}, pk_{\mathsf{P}})$ as in Game 0, except for a chosen index $i \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ sets the key generated in the i-th call to Init (by any party P) as $pk[i] \leftarrow \bar{B}$. Let $\mathsf{Bad}_1[i]$ denote event Bad_1 occuring for B which is this i-th key, i.e. $B = \bar{B}$.

As long as key $pk[i]$ is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to a compromise of all other keys, and it can service H queries as follows: To test server-side σ's, i.e. if $\sigma = (M \| L \| N) = (X^{sk} \| pk^y \| X^y)$, reduction $\mathcal{R}$ tests it as Game 6 does except for sk that corresponds to the public key $\bar{B}$, in which case it tests if $M = \mathsf{cdh}_g(\bar{B}, X)$, $L = pk^y$, and $N = X^y$. To test client-side σ's, i.e. if $\sigma = (L \| M \| N) = (pk^x \| Y^{sk} \| Y^x)$ for $x = s \cdot \bar{x}$ where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and any pk, including $pk = \bar{B}$, reduction $\mathcal{R}$ tests if $L = \mathsf{cdh}_g(\bar{X}, pk^s)$, $M = Y^{sk}$, and

$N = \mathsf{cdh}_g(\bar{X}, Y^s)$, except for the case that sk is the private key corresponding to the public key $\bar{B}$, in which case $\mathcal{R}$ replaces test $M = Y^{sk}$ with $M = \mathsf{cdh}_g(\bar{B}, Y)$.

Note that $\mathsf{Bad}_1[i]$ can happen only before key $pk[i]$ is compromised, so event $\mathsf{Bad}_1[i]$ occurs in the reduction with the same probability as in Game 6. (If $\mathcal{A}$ asks to compromise of $pk[i]$ then $\mathcal{R}$ aborts.) $\mathcal{R}$ can detect event $\mathsf{Bad}_1[i]$ because it occurs if H query involves the public key $pk[i] = \bar{B}$ and σ satisfies the client-side equation for this key, in which case $\mathcal{R}$ can output $L^{1/s} = \mathsf{cdh}_g(\bar{X}, \bar{B})$. If $\mathcal{R}$ picks index i at random it follows that $\Pr[\mathsf{Bad}_1] \leq q_{\mathrm{K}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$. Since a symmetric argument holds also for $\Pr[\mathsf{Bad}_2]$, we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq (2q_{\mathrm{K}}) \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Observe that Game 7 is identical to the ideal-world game shown in Figure 4: By Game 6 all functions $R^{\mathsf{sid}}_{\mathsf{P}}$ are random, by Game 5 the game responds to Z messages to $\mathsf{P}^{\mathsf{sid}}$ as the game in Figure 4, and after the modification in oracle H done in Game 7 this oracle also acts as in Figure 4. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof of Theorem 1.

4 HMQV as Key-Hiding AKE

We show that protocol HMQV [43], presented in Figure 6, realizes the UC notion of Key-Hiding AKE, as defined by functionality $\mathcal{F}_{\mathsf{khAKE}}$ in Section 2, under the Gap CDH assumption in ROM. It allows us to use HMQV with KHAPE, resulting in its most efficient instantiation, and, to the best of our knowledge the most efficient aPAKE protocol proposed. HMQV has been analyzed in [43] under the game-based AKE model of Canetti and Krawczyk [19], but the analysis we present is the first, to the best of our knowledge, to be done in the UC model.[8]

The logic of why HMQV is key hiding is similar to the case of 3DH. Namely, the only way to attack the privacy of party P which runs HMQV on inputs $(sk, pk) = (a, B)$, is to compromise the private key b corresponding to the public key B. (And symmetrically for the party that runs on $(sk, pk) = (b, A)$.) The HMQV equation, just like the 3DH key equation, involves both the ephemeral sessions secrets (x, y) and the long-term keys (a, b), combining them in a DH-like formula $\sigma = g^{(x+da)\cdot(y+eb)}$ where d, e are hashes of (session state identifiers and) resp. $X = g^x$ and $Y = g^y$. Following essentially the same arithmetics as in the proof due to [43] shows that the only way to compute σ is to know either *both* x, a or *both* y, b, which means that the attacker must be (1) active, to chose the ephemeral session state variable resp. x or y, and (2) it must know the counterparty private key, resp. a or b.

Theorem 2. *Protocol HMQV shown in Figure 6 realizes* $\mathcal{F}_{\mathsf{khAKE}}$ *if the Gap CDH assumption holds and* H, H' *are random oracles.*

The proof of theorem 2 follows the template of the proof for the corresponding theorem on 3DH security, i.e. theorem 1, and we defer it to Appendix C.

[8] However, we do not include adaptive session state compromise considered in [19, 43].

group $\mathbb{G}$ of prime order p with generator g

hash functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$, $\mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_p$

$\underline{\mathsf{P}_1 \text{ on } \mathsf{Init}}$		$\underline{\mathsf{P}_2 \text{ on } \mathsf{Init}}$
$a \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $A \leftarrow g^a$		$b \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $B \leftarrow g^b$
store $sk = a$ tagged by $pk = A$		store $sk = b$ tagged by $pk = B$
output $pk = A$		output $pk = B$
$\underline{\mathsf{P}_1 \text{ on } (\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_1, A, B)}$		$\underline{\mathsf{P}_2 \text{ on } (\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_2, B, A)}$
(assume $\mathsf{P}_1 <_{\mathsf{lex}} \mathsf{CP}_1$)		(assume $\mathsf{CP}_2 <_{\mathsf{lex}} \mathsf{P}_2$)
retrieve $sk = a$ tagged by $pk = A$		retrieve $sk = b$ tagged by $pk = B$
$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $X \leftarrow g^x$	$\xrightarrow{X}$ $\xleftarrow{Y}$	$y \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $Y \leftarrow g^y$
$d_1 \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{P}_1, \mathsf{CP}_1, 1, X)$		$d_2 \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{CP}_2, \mathsf{P}_2, 1, X)$
$e_1 \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{P}_1, \mathsf{CP}_1, 2, Y)$		$e_2 \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{CP}_2, \mathsf{P}_2, 2, Y)$
$\sigma_1 \leftarrow (Y \cdot B^{e_1})^{x + d_1 \cdot a}$		$\sigma_2 \leftarrow (X \cdot A^{d_2})^{y + e_2 \cdot b}$
$k_1 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{P}_1, \mathsf{CP}_1, X, Y, \sigma_1)$		$k_2 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{CP}_2, \mathsf{P}_2, X, Y, \sigma_2)$
output k_1		output k_2

Fig. 6. Protocol HMQV [43]

5 SKEME as Key-Hiding AKE

We present a variant of the SKEME protocol [41] in Figure 7, where we implement two modifications. First, we rely on OW-PCA 4 secure and perfect key-private 3 KEM. Recall that KEM is a special case of public key encryption customized to encrypting random keys, hence it follows from OW-PCA secure and key-hiding PKE [8]. Secondly, for compliance with the UC notion of AKE modeled by functionality $\mathcal{F}_{\mathsf{khAKE}}$, we derive the session key via a hash involving several additional elements, including a session identifier sid, party identities C and S, public keys A and B, and the transcript X, c, Y, d. We will also use $\{\mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma\}_{\mathrm{ord}}$ to denote $(\mathsf{P}, \mathsf{CP}, A, B, g^w, c, Z, d, (K, L, Z^w))$ if P plays role $= 1$, and string $(\mathsf{CP}, \mathsf{P}, A, B, Z, c, g^w, d, (K, L, Z^w))$ if role $= 2$. Using this notation each party P can derive its session key as $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma\}_{\mathrm{ord}})$.

Theorem 3. *Protocol SKEME shown in Figure 7 realizes $\mathcal{F}_{\mathsf{khAKE}}$ if the Gap CDH assumption holds,* KEM *is a OW-PCA secure and perfect key-private KEM, and* H *is a random oracle.*

Because of inherent similarities of SKEME and 3DH, the proof of the above theorem follows a similar pattern as the proof of Theorem 1, and we defer it to Appendix D.

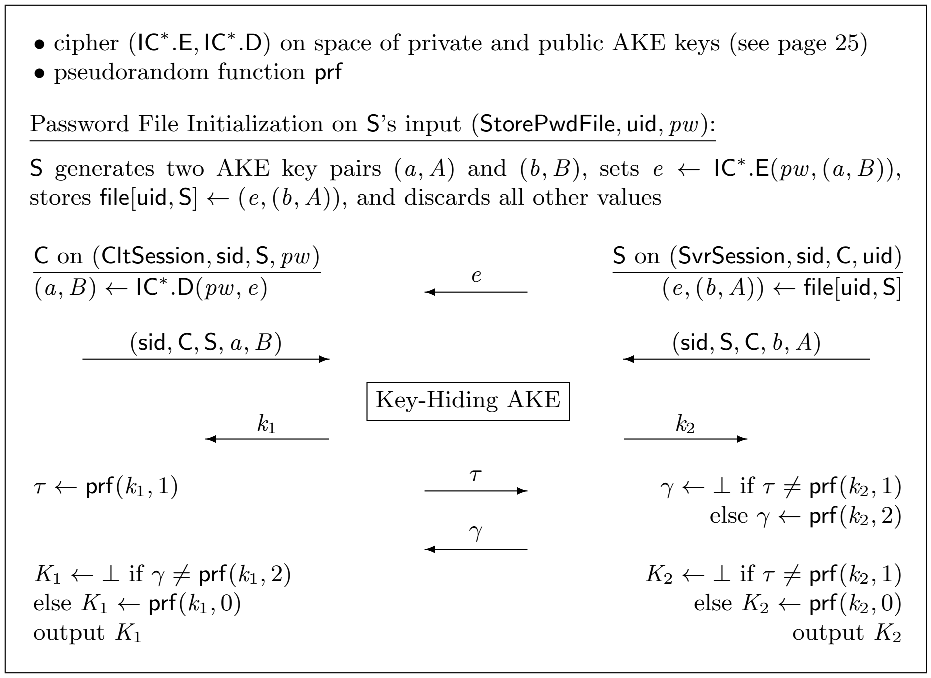
group $\mathbb{G}$ of prime order p with generator g
hash function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$
KEM scheme $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

$\underline{\mathsf{P}_1 \text{ on } \mathsf{Init}}$		$\underline{\mathsf{P}_2 \text{ on } \mathsf{Init}}$
$(a, A) \leftarrow \mathsf{KEM.Gen}$		$(b, B) \leftarrow \mathsf{KEM.Gen}$
store $sk = a$ tagged by $pk = A$		store $sk = b$ tagged by $pk = B$
output $pk = A$		output $pk = B$
$\underline{\mathsf{P}_1 \text{ on } (\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_1, A, B)}$		$\underline{\mathsf{P}_2 \text{ on } (\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_2, B, A)}$
(assume $\mathsf{P}_1 <_{\mathsf{lex}} \mathsf{CP}_1$)		(assume $\mathsf{CP}_2 <_{\mathsf{lex}} \mathsf{P}_2$)
retrieve $sk = a$ tagged by $pk = A$		retrieve $sk = b$ tagged by $pk = B$
$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $X \leftarrow g^x$		$y \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $Y \leftarrow g^y$
$c, K \leftarrow \mathsf{KEM.Enc}(B)$		$d, L \leftarrow \mathsf{KEM.Enc}(A)$
	$\xrightarrow{X, c}$ $\xleftarrow{Y, d}$	
$L \leftarrow \mathsf{KEM.Dec}(a, d)$		$K \leftarrow \mathsf{KEM.Dec}(b, c)$
$\sigma \leftarrow (K, L, Y^x)$		$\sigma \leftarrow (K, L, X^y)$
$k_1 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{P}_1, \mathsf{CP}_1, A, B, X, c, Y, d, \sigma)$		$k_2 \leftarrow \mathsf{H}(\mathsf{st}, \mathsf{CP}_2, \mathsf{P}_2, A, B, X, c, Y, d, \sigma)$
output k_1		output k_2

Fig. 7. Protocol SKEME: KEM-authenticated Key Exchange

6 Compiler from key-hiding AKE to asymmetric PAKE

We show that any UC Key-Hiding AKE protocol can be converted to a UC asymmetric PAKE (aPAKE) with a very small computational overhead. We call this AKE-to-aPAKE compiler construction KHAPE, which stands for Key-Hiding Asymmetric PakE, shown in Figure 8. The compiler views each party's AKE inputs, namely its own private key and its counterparty public key, as a single object, an AKE "credential". The two parties participating in aPAKE, the server and the user, a.k.a. the client, each will have such a credential: The server's credential contains the server's private key and the client's public key, and the client's credential contains the client's private key and the server's public key. Running AKE on such matching pair of inputs would establish a secure shared key, but while the server can store its credential, the client's only input is her password and it is not clear how one can derive an AKE credential from a password. Protocol KHAPE enables precisely this derivation: In addition to server's credential, the server will also store a ciphertext which encrypts, via an ideal cipher, the client's credential under the user's password, and the aPAKE protocol consists of server sending that ciphertext to the client, the client decrypting it using the user's password to obtain its certificate, and using that certificate to run an AKE instance with the server.


- cipher $(\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D})$ on space of private and public AKE keys (see page 25)
- pseudorandom function prf

Password File Initialization on S's input $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$:

S generates two AKE key pairs (a, A) and (b, B), sets $e \leftarrow \mathsf{IC}^*.\mathsf{E}(pw, (a, B))$, stores $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e, (b, A))$, and discards all other values

C on $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$

$(a, B) \leftarrow \mathsf{IC}^*.\mathsf{D}(pw, e)$ $\xleftarrow{e}$ $(e, (b, A)) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$

S on $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$

$\xrightarrow{(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)}$ $\boxed{\text{Key-Hiding AKE}}$ $\xleftarrow{(\mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A)}$

$\xleftarrow{k_1}$ $\xrightarrow{k_2}$

$\tau \leftarrow \mathsf{prf}(k_1, 1)$ $\xrightarrow{\tau}$ $\gamma \leftarrow \perp$ if $\tau \neq \mathsf{prf}(k_2, 1)$ else $\gamma \leftarrow \mathsf{prf}(k_2, 2)$

$\xleftarrow{\gamma}$

$K_1 \leftarrow \perp$ if $\gamma \neq \mathsf{prf}(k_1, 2)$
else $K_1 \leftarrow \mathsf{prf}(k_1, 0)$
output K_1

$K_2 \leftarrow \perp$ if $\tau \neq \mathsf{prf}(k_2, 1)$
else $K_2 \leftarrow \mathsf{prf}(k_2, 0)$
output K_2

Fig. 8. Protocol KHAPE: Compiler from Key-Hiding AKE to aPAKE

Reduced-bandwidth variant. In the aPAKE construction in Figure 8, ciphertext e password-encrypts a pair of the client's secret key sk_{C} and the server's public key pk_{S}. Without loss of generality every AKE key pair (sk, pk) is generated by the key generation algorithm from uniformly sampled randomness r. The aPAKE construction can be modified so that envelope e password-encrypts only the server's public key pk_{S}, while the client derives its private key sk_{C} using the key generation algorithm on randomness $r \leftarrow \mathsf{H}(pw)$ via RO hash H. Note that if key-hiding AKE is either 3DH or HMQV then this amounts to the client setting it's secret exponent $a \leftarrow \mathsf{H}(pw)$ where H maps onto range $\mathbb{Z}_q$.[9] This change does not simplify the construction of the ideal cipher by much because typically the public key is a group element and the private key is a random modular residue, but it reduces the size of ciphertext e. We believe that the security proof for the aPAKE protocol in Figure 8 can be adjusted to show security of this reduced-bandwidth implementation.

Why we need key-hiding AKE. Note that anyone who observes the credential-encrypting ciphertext e can decrypt it under any password. Each password guess will decrypt e into some credential $cred = (sk_{\mathsf{C}}, pk_{\mathsf{S}})$, where sk_{C} is a client's private key and pk_{S} is a server's public key. Let $cred(pw)$

[9] If AKE is implemented as SKEME of Section 5 then the client must also derive the public key pk_{C}, since it is used in the key-derivation hash, see Figure 7.

denote the credential obtained by decrypting e using password pw. For any password guess pw^* the attacker can use credential $cred(pw^*)$ as input to an AKE protocol with the server, but that is equivalent to an on-line password authentication attempt using pw^* as a password guess (see below). Note that the attacker can also either watch or interfere with AKE instances executed by the honest user on credential $cred(pw)$ that corresponds to the correct password pw. Moreover, the attacker w.l.o.g. holds a list of credential candidates $cred(pw_1), ..., cred(pw_n)$ corresponding to offline password guesses. However, the key-hiding property of AKE implies that even if $cred(pw)$ is on the attacker's list, interfering or watching client's AKE instances cannot help the attacker decide which credential is the one that the client uses. The only way to learn anything from client AKE instances on input $cred(pw)$ would be to engage them using a matching credential, i.e. $(sk_{\mathsf{S}}, pk_{\mathsf{C}})$. This is possible if the adversary compromises the server who holds exactly these keys, but otherwise doing so is equivalent to breaking AKE security.

Why we need mutual key confirmation. To handle the server-side attack we needed the key-hiding property of AKE to imply that the only way to decide which keys $(sk_{\mathsf{S}}, pk_{\mathsf{C}})$ the server uses is to engage in an AKE instance using the matching counterparty keys (sk_{C}, pk_S). The key-hiding property provided by 3DH and HMQV, as modeled by functionality $\mathcal{F}_{\mathsf{khAKE}}$, actually does *not* suffice for this by itself. Let the attacker hold a list of n possible decrypted client credentials $cred_i = cred(pw_i) = (a_i, B_i)$ for $i = 1, ..., n$, and let S hold credential $cred_{\mathsf{S}} = (b, A)$ which matches $cred_i$, i.e. $A = g^{a_i}$ and $B_i = g^b$, which is the case if password guess pw_i matches the correct password pw. If an active attacker chooses x and sends $X = g^x$ to S then it can locally complete the 3DH or HMQV equation using *any* key pair (a_i, B_i) it holds, thus computing n candidate session keys k_i. By 3DH or HMQV correctness, since the i-th client credential matches the server's credential, key k_i equals to the session key k computed by S. Therefore, if S used key k straight away then the attacker could observe that $k_i = k$ and hence that $pw_i = pw$.

However, the fix is simple: To make the server's session key output safe to use, the client must first send a key confirmation message to the server, implemented in Figure 8 by client's final message τ. This stops the attack because the attacker sending τ uniquely determines one of the keys k_i on its candidate list, and since this succeeds only if $k_i = k$, this attack reduces to an on-line test of a single password guess pw_i, which is unavoidable in a (a)PAKE protocol. A natural question is if there is no equivalent attack on the client-side, which would be abetted by the client sending a key confirmation message τ. This is not the case because of the following asymmetry: Off-line password guesses give the attacker a list of possible *client-side* credentials, which by AKE rules can be tested against server sessions. However, by the the key-hiding property of AKE such credentials are useless in deciding which of them, if any, is used by the honest user. Moreover, since the ciphertext e encrypts only the client-side keys, by the KCI property of the AKE the knowledge of client-side keys is not helpful in breaking the security of AKE instances executed by the honest client on such keys.

Server-to-client key confirmation is needed too, in this case to ensure forward secrecy. Without it, an attacker could choose $Y = g^y$ (in the HMQV or 3DH instantiations) and later, after the session is complete, compromise the server to learn the private key b with which it can compute the session key. The client-to-server key confirmation addresses this issue on the client side.

In addition to ensuring security, key confirmation serves as (explicit) *entity authentication* in this aPAKE construction.

Why we need credential encryption to be an ideal cipher. Note that the attacker can attack the client too, by sending an arbitrary ciphertext to the client, but the ideal cipher property is that the ciphertext commits the attacker to only one choice of key for which the attacker can decide a plaintext: for all other keys the decrypted plaintext will be random.

For the above to work the encryption used to password-encrypt the client credential needs to be an ideal cipher over the space of (private,public) key pairs used in AKE. In all key-hiding AKE protocols examples we discuss in this paper, i.e. 3DH, HMQV, as well as SKEME instantiated with Diffie-Hellman KEM, this message space is $\mathbb{Z}_p \times \mathbb{G}$ where $\mathbb{G}$ is a group of order p. We refer to Section 8 for several methods of instantiate an ideal cipher on this space. Here we will assume the implementation of the following form, which is realized by the Elligator2 or Elligator-squared encodings (see Section 8). Let X be the Cartesian product of the space of private keys and the space of public keys in AKE, let $\mathsf{IC.E}, \mathsf{IC.D}$ be an ideal cipher on n-bit strings, and let map be a (randomized) invertible quasi-bijective map from X to $X' = \{0,1\}^n$. A randomized 1-1 function $\mathsf{map} : X \to X'$ is quasi-bijective if there is a negligible statistical difference between a uniform distribution over X' and $x' \leftarrow_{\mathrm{R}} \mathsf{map}(x)$ for random x in X. Instead of a direct ideal cipher on message space X protocol KHAPE in Fig. 8 uses a randomized cipher $(\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D})$ on X' where $\mathsf{IC}^*.\mathsf{E}(x)$ outputs $\mathsf{IC.E}(x')$ where $x' \leftarrow \mathsf{map}(x; r)$ for random r used by map, and $\mathsf{IC}^*.\mathsf{D}(y)$ outputs $x = \mathsf{map}^{-1}(x')$ where $x' = \mathsf{IC.D}(y)$.

Comparison with Encrypted Key Exchange of Bellovin-Merritt. It is instructive to compare the KHAPE design to that of the "Encrypted Key Exchange" (EKE) construction of Bellovin-Meritt [10]. The EKE compiler starts from unauthenticated KE, uses an Ideal Cipher to encrypt each KE protocol message under the password, and this results in UC PAKE in the IC model (see e.g. [48]). By contrast, our compiler starts from *Authenticated* KE, and uses IC to password-encrypt only the client's inputs to the AKE protocol, while the protocol messages themselves are exchanged without any change. Just like EKE, our compiler adds only symmetric-key overhead to the underlying KE, but it results in an aPAKE instead of just PAKE. However, just like EKE, it imposes additional requirements on the underlying key exchange protocol: Whereas EKE needs the key exchange to have a "random transcript" property, i.e. KE protocol messages must be random in some message space, in the case of KHAPE the underlying AKE needs to have the key-hiding property we define in Section 2. Either condition also relies on an Ideal Cipher (IC) modeling for a non-standard plaintext space: For EKE the

IC plaintext space is the space of KE protocol *messages*, while for KHAPE the IC plaintext space is the Cartesian product of the space of private keys and the space of public keys which form AKE protocol *inputs*.

UC aPAKE security model. We refer to Appendix B for the functionality $\mathcal{F}_{\mathsf{aPAKE}}$ we use to model UC aPAKE. This model is very similar to the one defined by Gentry et al. [29], but it introduces some modifications, which we discuss in Appendix B. The main notational change is that we use a *user account identifier* uid, instead of generic *session identifier* sid, to index password files held by a given server. Functionality $\mathcal{F}_{\mathsf{aPAKE}}$ also includes uni-directional (client-to-server) entity authentication as part of the security definition. We refer to Appendix B also for a discusson of several subtle issues involved in UC modeling of tight bounds on adversary's local computation during an offline dictionary attack.

Theorem 4. *Protocol* KHAPE *realizes the UC aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$ *if the AKE protocol realizes the Key-Hiding AKE functionality* $\mathcal{F}_{\mathsf{khAKE}}$*, assuming that* prf *is a secure PRF and* $(\mathsf{Enc}, \mathsf{Dec})$ *is an ideal cipher over message space of private,public key pairs in AKE.*

We show that the environment's view of the *real-world* security game, denoted Game 0, i.e. an interaction between the real-world adversary and honest parties who follow protocol KHAPE, is indistinguishable from the environment's view of the *ideal-world* game, denoted Game 7, i.e. an interaction between simulator SIM of Figures 9 and 10 and functionality $\mathcal{F}_{\mathsf{aPAKE}}$. As before, we use Gi to denote the event that $\mathcal{Z}$ outputs 1 while interacting with Game i, and the theorem follows if $|\Pr[\mathsf{G0}] - \Pr[\mathsf{G7}]|$ is negligible. For a fixed environment $\mathcal{Z}$, let q_{pw}, q_{IC}, and q_{ses} be the upper-bounds on the number of resp. password files, IC queries, and online S or C aPAKE sessions. Let $\epsilon^{\mathcal{Z}}_{\mathrm{kdf}}(\mathsf{SIM}_{\mathsf{AKE}})$ and $\epsilon^{\mathcal{Z}}_{\mathrm{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$ be the advantages of an environment who uses the resources of $\mathcal{Z}$ plus $O(q_{\mathsf{IC}} + q_{\mathrm{ses}} + q_{\mathsf{pw}})$ exponentiations in $\mathbb{G}$ in resp. breaking the PRF security of prf, and in distinguishing between the real-world AKE protocol and its ideal-world emulation of $\mathsf{SIM}_{\mathsf{AKE}}$ interacting with $\mathcal{F}_{\mathsf{khAKE}}$. Let $X' = Y = \{0,1\}^n$ be the domain and range of the ideal cipher IC used within IC^*, let X be the domain of (private,public) keys in AKE (e.g. for both 3DH and HMQV we have $X = \mathbb{Z}_p \times \mathbb{G}$ where $\mathbb{G}$ is a group of order p), and let $\mathsf{map} : X \to \{0,1\}^n$ be ϵ_{map}-quasi-bijective.

Simulator construction. We split the description of simulator SIM into two phases: Figure 9 shows how SIM deals with creation and compromise of a password file and with adversary's ideal cipher queries, while Figure 10 shows how SIM deals with on-line sessions, i.e. how it executes AKE sessions and translates adversary's responses into on-line attacks on the aPAKE.

Simulator SIM uses as a sub-procedure the AKE-protocol simulator $\mathsf{SIM}_{\mathsf{AKE}}$, which exists by the assumption that the AKE protocol realizes functionality $\mathcal{F}_{\mathsf{khAKE}}$. Namely, SIM hands over to $\mathsf{SIM}_{\mathsf{AKE}}$ the simulation of all C-side and S-side AKE instances where parties run on honestly generated AKE keys. SIM employs $\mathsf{SIM}_{\mathsf{AKE}}$ to generate such keys, in password file initialization and in IC decryption queries, see Figure 9, and then it hands off to $\mathsf{SIM}_{\mathsf{AKE}}$ the handling

```
Initialization
Initialize simulator $\mathsf{SIM_{AKE}}$, an empty table $\mathsf{T_{IC}}$, empty lists $CPK, PK_\mathsf{C}, PK_\mathsf{S}$
Notation: $\mathsf{T}^{pw}_\mathsf{IC}.X' = \{x' \| \exists y\ (pw, x', y) \in \mathsf{T_{IC}}\}$, $\mathsf{T}^{pw}_\mathsf{IC}.Y = \{y \mid \exists x'\ (pw, x', y) \in \mathsf{T_{IC}}\}$
Convention: First call to SvrSession or StealPwdFile for $(\mathsf{S}, \mathsf{uid})$ sets $e^\mathsf{uid}_\mathsf{S} \leftarrow_\mathrm{R} Y$

Ideal Cipher IC queries
  – On query $(pw, x')$ to IC.E, send back $y$ if $(pw, x', y) \in \mathsf{T_{IC}}$, otherwise pick
    $y \leftarrow_\mathrm{R} Y \setminus \mathsf{T}^{pw}_\mathsf{IC}.Y$, add $(pw, x', y)$ to $\mathsf{T_{IC}}$, and send back $y$
  – On query $(pw, y)$ to IC.D, send back $x'$ if $(pw, x', y) \in \mathsf{T_{IC}}$, otherwise do:
      1. If $y \neq e^\mathsf{uid}_\mathsf{S}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \leftarrow_\mathrm{R} X' \setminus \mathsf{T}^{pw}_\mathsf{IC}.X'$
      2. If $y = e^\mathsf{uid}_\mathsf{S}$ for some $(\mathsf{S}, \mathsf{uid})$ send $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_\mathsf{aPAKE}$ and:
         (a) If $\mathcal{F}_\mathsf{aPAKE}$ sends "correct guess" then set $(A, B) \leftarrow (A^\mathsf{uid}_\mathsf{S}, B^\mathsf{uid}_\mathsf{S})$
         (b) Otherwise initialize keys $A$ and $B$ via two Init calls to $\mathsf{SIM_{AKE}}$, add $A$
             to $PK_\mathsf{C}$ and $B$ to $PK_\mathsf{S}$
         Set $pk^\mathsf{uid}_\mathsf{S}(pw) \leftarrow (A, B)$, send query $(\mathsf{Compromise}, A)$ to $\mathsf{SIM_{AKE}}$, define $a$
         as $\mathsf{SIM_{AKE}}$'s response, add $A$ to $CPK$, set $x' \leftarrow_\mathrm{R} \mathsf{map}(a, B)$
    In either case add $(pw, x', y)$ to $\mathsf{T_{IC}}$ and send back $x'$

Stealing Password Data
On $\mathcal{Z}$'s permission to do so send $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_\mathsf{aPAKE}$. If $\mathcal{F}_\mathsf{aPAKE}$ sends
"no password file," pass it to $\mathcal{A}$, otherwise declare $(\mathsf{S}, \mathsf{uid})$ compromised and:
  1. If $\mathcal{F}_\mathsf{aPAKE}$ returns no value then initialize keys $A$ and $B$ via two Init calls to
     $\mathsf{SIM_{AKE}}$, add $A$ to $PK_\mathsf{C}$ and $B$ to $PK_\mathsf{S}$
  2. If $\mathcal{F}_\mathsf{aPAKE}$ returns $pw$ then set $(A, B) \leftarrow pk^\mathsf{uid}_\mathsf{S}(pw)$
Send $(\mathsf{Compromise}, B)$ to $\mathsf{SIM_{AKE}}$, define $b$ as $\mathsf{SIM_{AKE}}$'s response, add $B$ to $CPK$,
set $(A^\mathsf{uid}_\mathsf{S}, B^\mathsf{uid}_\mathsf{S}) \leftarrow (A, B)$, return $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e^\mathsf{uid}_\mathsf{S}, b, A)$ to $\mathcal{A}$.
```

Fig. 9. Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_\mathsf{aPAKE}$: Part 1

of all AKE instances that run on such keys, see Figure 10. SIM cannot handle all AKE executions via $\mathsf{SIM_{AKE}}$, because the adversary can guess client C's password pw and form an envelope e' as IC encryption of arbitrary keys (a^*, B^*) under pw, in which case C executes AKE on adversarial keys (a^*, B^*). The ideal model of key-hiding AKE, i.e. functionality $\mathcal{F}_\mathsf{khAKE}$ of Figure 1, allows the environment to invoke AKE sessions on adversarially chosen counterparty public key, i.e. B^*, but it assumes that an honest party can use only its own previously generated key as its private key a. Since functionality $\mathcal{F}_\mathsf{khAKE}$ makes no claims for parties who run on inputs that violate this assumption, simulator SIM in this case simply executes the AKE protocol on behalf of C on such adversarially-chosen inputs (a^*, B^*). However, since this case implies a succesful on-line password guessing attack against client C, the simulation can give up on security on such sessions, hence w.l.o.g. this AKE execution could reveal inputs a^*, B^* to the adversary.

Game 0 *(real world)*: This is the interaction, shown in Figure 11, of environment $\mathcal{Z}$ with the real-world protocol KHAPE, except that the symmetric encryption scheme is idealized as an ideal cipher oracle.

Starting AKE sessions

On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ from $\mathcal{F}_{\mathsf{aPAKE}}$, initialize random function $R^{\mathsf{sid}}_{\mathsf{S}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, send $e^{\mathsf{uid}}_{\mathsf{S}}$ to $\mathcal{A}$ as a message from $\mathsf{S}^{\mathsf{sid}}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$.

On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathsf{aPAKE}}$ and message e' sent by $\mathcal{A}$ to $\mathsf{C}^{\mathsf{sid}}$, initialize random function $R^{\mathsf{sid}}_{\mathsf{C}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$, and:

1. If $e' = e^{\mathsf{uid}}_{\mathsf{S}}$ set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
2. If $e' \neq e^{\mathsf{uid}}_{\mathsf{S}}$ check if e' was output by $\mathsf{IC.E}$ on some (pw, x'), and:
   (a) If there is no such $\mathsf{IC.E}$ query then send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$, and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
   (b) Otherwise define (pw, x') as the first query to $\mathsf{IC.E}$ which outputted e', send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, and:
      i. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "wrong guess" then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
      ii. If $\mathcal{F}_{\mathsf{aPAKE}}$ returns "correct guess" then set $(a, B) \leftarrow \mathsf{map}^{-1}(x')$ and run the AKE protocol on behalf of $\mathsf{C}^{\mathsf{sid}}$ on inputs $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$; When $\mathsf{C}^{\mathsf{sid}}$ terminates with key k then send $\tau \leftarrow \mathsf{prf}(k, 1)$ to $\mathcal{A}$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{prf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$

Responding to AKE messages

SIM forwards AKE protocol messages between $\mathcal{A}$ and $\mathsf{SIM}_{\mathsf{AKE}}$, and reacts as follows to $\mathsf{SIM}_{\mathsf{AKE}}$'s queries to $\mathcal{F}_{\mathsf{khAKE}}$, whose role is played by SIM. (SIM ignores $\mathsf{SIM}_{\mathsf{AKE}}$'s queries pertaining to any $\mathsf{P}^{\mathsf{sid}}$ that was not started by a $\mathsf{NewSession}$ message.)

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$ set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$ and $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$ then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:

1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}$ then send $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ to $\mathcal{F}_{\mathsf{aPAKE}}$;
   If $\mathcal{F}_{\mathsf{aPAKE}}$ sends "correct guess" output $\tau \leftarrow \mathsf{prf}(k, 1)$ for $k = R^{\mathsf{sid}}_{\mathsf{C}}(A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}, \alpha)$
2. In any other case (including "wrong guess" above), output $\tau \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and $\mathcal{A}$ sends τ' to $\mathsf{S}^{\mathsf{sid}}$:

1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' was generated by SIM for $\mathsf{C}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$, then send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$ and output $\gamma \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\tau' = \mathsf{prf}(k,1)$ for $k = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, \alpha)$ and $(A, B) = pk^{\mathsf{uid}}_{\mathsf{S}}(pw)$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, pw), (\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \mathsf{prf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$, output $\gamma \leftarrow \mathsf{prf}(k, 2)$
3. Else $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{S}, \perp), (\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, output $\gamma \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$

If $\mathsf{SIM}_{\mathsf{AKE}}$ sends γ' to $\mathsf{C}^{\mathsf{sid}}$:

1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}^{\mathsf{uid}}_{\mathsf{S}}$ and γ' was generated by SIM for $\mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$
2. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}^{\mathsf{uid}}_{\mathsf{S}}$, $\mathcal{F}_{\mathsf{aPAKE}}$ sent "correct guess" for $\mathsf{C}^{\mathsf{sid}}$, and $\gamma' = \mathsf{prf}(k, 2)$ for k computed for $\mathsf{C}^{\mathsf{sid}}$ above, send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \mathsf{prf}(k, 0))$ to $\mathcal{F}_{\mathsf{aPAKE}}$
3. Else send $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{C}, \perp), (\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\mathsf{aPAKE}}$

If $\mathsf{SIM}_{\mathsf{AKE}}$ outputs $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$:
    If $pk \in PK_{\mathsf{P}}$ and $pk' \in CPK$ send $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ to $\mathcal{A}$

Fig. 10. Simulator SIM showing that protocol KHAPE realizes $\mathcal{F}_{\mathsf{aPAKE}}$: Part 2

(Technically, this is a hybrid world where each party has access to the ideal cipher functionality IC.)

Initialize empty table $\mathsf{T_{IC}}$; (Notation $\mathsf{T}^{pw}_{\mathsf{IC}}.X'$ and $\mathsf{T}^{pw}_{\mathsf{IC}}.Y$ as in Fig. 9)

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw^{\mathsf{uid}}_{\mathsf{S}})$ to S: Generate keys (a, A), (b, B), set $e^{\mathsf{uid}}_{\mathsf{S}} \leftarrow \mathsf{IC.E}(pw^{\mathsf{uid}}_{\mathsf{S}}, \mathsf{map}(a, B))$, and $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e^{\mathsf{uid}}_{\mathsf{S}}, b, A)$
- On new (pw, x') to $\mathsf{IC.E}$: Output $y \leftarrow_{\mathrm{R}} Y \setminus \mathsf{T}^{pw}_{\mathsf{IC}}.Y$, add (pw, x', y) to $\mathsf{T_{IC}}$
- On new (pw, y) to $\mathsf{IC.D}$: Output $x' \leftarrow_{\mathrm{R}} X' \setminus \mathsf{T}^{pw}_{\mathsf{IC}}.X'$, add (pw, x', y) to $\mathsf{T_{IC}}$
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Set $(e^{\mathsf{uid}}_{\mathsf{S}}, (b, A)) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$, send $e^{\mathsf{uid}}_{\mathsf{S}}$ and start AKE session $\mathsf{S}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{S}, \mathsf{C}, b, A)$, set k_2 to $\mathsf{S}^{\mathsf{sid}}$ output;
  If $\mathcal{Z}$ sends $\tau' = \mathsf{prf}(k_2, 1)$ to $\mathsf{S}^{\mathsf{sid}}$, set K_2, γ as $\mathsf{prf}(k_2, 0), \mathsf{prf}(k_2, 2)$, else as $\perp, \perp$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and message e' to C: Set $(a, B) \leftarrow \mathsf{map}^{-1}(\mathsf{IC.D}(pw, e'))$, and start AKE session $\mathsf{C}^{\mathsf{sid}}$ on $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, a, B)$, set k_1 to $\mathsf{C}^{\mathsf{sid}}$ output, send $\tau = \mathsf{prf}(k_1, 1)$ to $\mathcal{Z}$;
  If $\mathcal{Z}$ sends $\gamma' = \mathsf{prf}(k_1, 2)$ to $\mathsf{C}^{\mathsf{sid}}$, set $K_1 = \mathsf{prf}(k_1, 0)$ else $K_1 = \perp$

Fig. 11. Game 0: $\mathcal{Z}$'s interaction with real-world protocol KHAPE

Game 1 *(embedding random keys in* $\mathsf{IC.D}$ *outputs)*: We modify processing of $\mathcal{Z}$'s query (pw, y) to $\mathsf{IC.D}$ for any $y \notin \mathsf{T}^{pw}_{\mathsf{IC}}.Y$, i.e. y for which $\mathsf{IC.D}(pw, y)$ has not been yet defined. On such query Game 1 generates fresh key pairs (a, A) and (b, B), sets $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$, and if $x' \notin \mathsf{T}^{pw}_{\mathsf{IC}}.X'$ then it sets $\mathsf{IC.D}(pw, y) \leftarrow x'$. If $x' \in \mathsf{T}^{pw}_{\mathsf{IC}}.X'$, i.e. x' is already mapped by $\mathsf{IC.E}(pw, \cdot)$ to some value, Game 1 aborts. If $y = e^{\mathsf{uid}}_{\mathsf{S}}$ for some $(\mathsf{S}, \mathsf{uid})$ then the game also sets $pk^{\mathsf{uid}}_{\mathsf{S}}(pw) \leftarrow (A, B)$.

The divergence this game introduces is due to the probability $(q_{\mathsf{IC}})^2/2^n$ of ever encountering an abort, and the statistical distance $q_{\mathsf{IC}}\epsilon_{\mathsf{map}}$ between random IC domain elements and images of map on random X elements, which leads to $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq q_{\mathsf{IC}}\epsilon_{\mathsf{map}} + (q_{\mathsf{IC}})^2/2^n$.

Game 2 *(random* $e^{\mathsf{uid}}_{\mathsf{S}}$ *in the password file)*: We change $\mathsf{StorePwdFile}$ processing by picking ciphertext $e^{\mathsf{uid}}_{\mathsf{S}}$ as a random element in $\{0,1\}^n$ instead of via query to $\mathsf{IC.E}$, then we pick two key pairs (a, A), (b, B), define $(A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}) \leftarrow (A, B)$, and sample $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$. If $e^{\mathsf{uid}}_{\mathsf{S}} \in \mathsf{T}^{pw}_{\mathsf{IC}}.Y$ for *any* pw, not necessarily $pw^{\mathsf{uid}}_{\mathsf{S}}$, the game aborts. The game also aborts if $x' \in \mathsf{T}^{pw}_{\mathsf{IC}}.X'$ for $pw = pw^{\mathsf{uid}}_{\mathsf{S}}$. Otherwise the game sets $\mathsf{IC.D}(pw^{\mathsf{uid}}_{\mathsf{S}}, e^{\mathsf{uid}}_{\mathsf{S}}) \leftarrow x'$ and $pk^{\mathsf{uid}}_{\mathsf{S}}(pw^{\mathsf{uid}}_{\mathsf{S}}) \leftarrow (A, B)$. The divergence this game introduces is due to the probability of abort occuring in either case, which leads to $|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq q_{\mathsf{pw}}\epsilon_{\mathsf{map}} + 2q_{\mathsf{pw}}q_{\mathsf{IC}}/2^n$.

Game 3 *(abort on ambiguous ciphertexts)*: In the ideal-world game simulator SIM identifies ciphertext e' which was output by the ideal cipher for some query (pw, x') to $\mathsf{IC.E}$, as an encryption of the *first* (pw, x') pair which satisfies this. To eliminate the possibility of ambiguous ciphertexts we introduce an abort if $\mathsf{IC.E}$

oracle picks the same ciphertext for any two queries (pw_1, x'_1) and (pw_2, x'_2). Since $\mathsf{IC.E}$ samples random outputs in Y we get $|\Pr[\mathsf{G3}] - \Pr[\mathsf{G2}]| \leq (q_{\mathsf{IC}})^2/2^n$.

Taking stock of the game. Let us review how Game 3 operates: The initialization of password file $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ on password $pw_{\mathsf{S}}^{\mathsf{uid}}$ picks fresh keys (a, A), (b, B), picks $e_{\mathsf{S}}^{\mathsf{uid}}$ as a random string, keeps the client and server public keys as $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}}) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) = (A, B)$, and programs $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ to $\mathsf{map}(a, B)$. Oracle $\mathsf{IC.D}$ on inputs (pw', y) for which decryption is undefined, picks fresh key pairs (a', A') and (b', B') and programs $\mathsf{IC.D}(pw', y)$ to $\mathsf{map}(a', B')$. In addition, if $y = e_{\mathsf{S}}^{\mathsf{uid}}$ then it assigns $pk_{\mathsf{S}}^{\mathsf{uid}}(pw') \leftarrow (a', B')$. Finally, encryption is now unambiguous, i.e. every ciphertext e can be output by $\mathsf{IC.E}$ on only one pair (pw, x').

This is already very close to how simulator SIM operates as well. The crucial difference between the ideal-world interaction and Game 3, is that in Game 3 keys $A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}$ are generated at the time of password file initialization, and $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set to $\mathsf{map}(a_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ at the same time. In the ideal-world game these keys are undefined until password compromise, and $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is set only after offline dictionary attack succeeds in finding $pw_{\mathsf{S}}^{\mathsf{uid}}$. This delayed generation of the keys in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ is possible because AKE sessions which S and C run on these keys can be simulated without knowledge of these keys, an key-hiding AKE functionality allows precisely for such simulation, as we show next.

Game 4 *(Using $\mathsf{SIM}_{\mathsf{AKE}}$ for AKE's on honestly-generated keys)*: In Game 4 we modify Game 3 by replacing all honest parties that run AKE instances on keys A, B generated either in password file initialization or by oracle $\mathsf{IC.D}$, with a simulation of these AKE instances via simulator $\mathsf{SIM}_{\mathsf{AKE}}$. Game 4 is shown in Figure 12. For notational brevity in Figure 12 we say that query (pw, x') to $\mathsf{IC.E}$ or (pw, y) to $\mathsf{IC.D}$ are new$^{(!)}$ as a shortcut for saying that table T_{IC} includes no prior tuple corresponding to these inputs, resp. $(pw, x', \cdot)$ and $(pw, \cdot, y)$. If such tuple exists then $\mathsf{IC.E}$ and $\mathsf{IC.D}$ oracles use the retrieved (key,input,output) tuple to answer the according query. We also omit the possibilities of the game aborts, because such aborts happen only with negligible probability. These aborts occur in three places, all marked $^{(*)}$: (1) When $e_{\mathsf{S}}^{\mathsf{uid}}$ is chosen in $\mathsf{StorePwdFile}$ the game aborts if $e_{\mathsf{S}}^{\mathsf{uid}} \in \mathsf{T}_{\mathsf{IC}}^{pw}.Y$ for any pw (not necessarily $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$); (2) When x' is then sampled as $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$, the game aborts if $x' \in \mathsf{T}_{\mathsf{IC}}^{pw}.X'$ for $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$; (3) When $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$ is sampled in $\mathsf{IC.D}$ query (pw, y) the game aborts also if $x' \in \mathsf{T}_{\mathsf{IC}}^{pw}.X'$.

Game 4 operates like Game 3, except that it outsources AKE key generation in $\mathsf{StorePwdFile}$ and $\mathsf{IC.D}$ to $\mathsf{SIM}_{\mathsf{AKE}}$, and whenever $\mathsf{S}^{\mathsf{sid}}$ or $\mathsf{C}^{\mathsf{sid}}$ runs AKE on such keys these executions are outsourced to $\mathsf{SIM}_{\mathsf{AKE}}$, while the game emulates what $\mathcal{F}_{\mathsf{khAKE}}$ would do in response to $\mathsf{SIM}_{\mathsf{AKE}}$'s actions. In particular, Game 4 initializes random function $R_{\mathsf{P}}^{\mathsf{sid}}$ for every AKE session $\mathsf{P}^{\mathsf{sid}}$ invoked by emulated $\mathcal{F}_{\mathsf{khAKE}}$. Whenever C and S run an AKE instance under keys generated by AKE key generation the game, playing $\mathcal{F}_{\mathsf{khAKE}}$, triggers $\mathsf{SIM}_{\mathsf{AKE}}$ with messages resp. $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$. When $\mathsf{SIM}_{\mathsf{AKE}}$ translates the real-world adversary's behavior into $\mathsf{Interfere}$

Initialize simulator $\mathsf{SIM}_{\mathsf{AKE}}$, empty table T_{IC}, and lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Initialize A and B via two Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, send $(\mathsf{Compromise}, A)$ and $(\mathsf{Compromise}, B)$ to $\mathsf{SIM}_{\mathsf{AKE}}$, define a and b as $\mathsf{SIM}_{\mathsf{AKE}}$'s responses, add A to PK_{C}, B to PK_{S}, and both to CPK, pick$^{(*)}$ $e_{\mathsf{S}}^{\mathsf{uid}} \leftarrow_{\mathrm{R}} Y$, set$^{(*)}$ $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$, add $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ to T_{IC}, set $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$ and $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, B)$
- On new$^{(!)}$ (pw, x') to $\mathsf{IC.E}$: Output $y \leftarrow_{\mathrm{R}} Y \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.Y$, add (pw, x', y) to T_{IC}
- On new$^{(!)}$ (pw, y) to $\mathsf{IC.D}$: Initialize A and B via two Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, send $(\mathsf{Compromise}, A)$ to $\mathsf{SIM}_{\mathsf{AKE}}$, define a as $\mathsf{SIM}_{\mathsf{AKE}}$'s response, add A to CPK and PK_{C}, add B to PK_{S}, set$^{(*)}$ $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$ add (pw, x', y) to T_{IC}, output x'
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$: Output $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Initialize function $R_{\mathsf{S}}^{\mathsf{sid}}$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, output $e_{\mathsf{S}}^{\mathsf{uid}}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and e' to C: Initialize function $R_{\mathsf{C}}^{\mathsf{sid}}$ and:
  1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$, set $x' \leftarrow \mathsf{IC.D}(pw, e_{\mathsf{S}}^{\mathsf{uid}})$, $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}(g^a, B)$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
  2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$, check if e' was output by $\mathsf{IC.E}$ on (pw, x') for some x' and:
     (a) If not, set $x' \leftarrow \mathsf{IC.D}(pw, e')$, $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}(g^a, B)$, send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
     (b) If so, set $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, run $\mathsf{C}^{\mathsf{sid}}$ of AKE on $(\mathsf{sid}, \mathsf{S}, a, B)$; If $\mathsf{C}^{\mathsf{sid}}$ terminates with k, output $\tau \leftarrow \mathsf{prf}(k, 1)$ and $K_1 \leftarrow \mathsf{prf}(k, 0)$

<u>Responding to AKE messages:</u>

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}(A, B)$ then change it to $\mathsf{act}(A, B)$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}(A, B)$ set $k_1 \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, \alpha)$
  2. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}(A, B)$: If $(A, B) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ and $\mathsf{S}^{\mathsf{sid}}$ outputted key k_2 then copy this k_2 to k_1, otherwise pick $k_1 \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$

  Output $\tau \leftarrow \mathsf{prf}(k_1, 1)$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$, set $k_2 \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B_{\mathsf{S}}^{\mathsf{uid}}, A_{\mathsf{S}}^{\mathsf{uid}}, \alpha)$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$: If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ and $\mathsf{C}^{\mathsf{sid}}$ outputted key k_1 then copy this k_1 to k_2, otherwise pick $k_2 \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$

  If $\tau' = \mathsf{prf}(k_2, 1)$ output $(K_2, \gamma) \leftarrow (\mathsf{prf}(k_2, 0), \mathsf{prf}(k_2, 2))$, else $(K_2, \gamma) \leftarrow (\perp, \perp)$
- On γ' to $\mathsf{C}^{\mathsf{sid}}$: If $\gamma' = \mathsf{prf}(k_1, 2)$ output $K_1 \leftarrow \mathsf{prf}(k_1, 0)$ else $K_1 \leftarrow \perp$
- On $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: send $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ if $pk \in PK_{\mathsf{P}}$, $pk' \in CPK$

Fig. 12. Proof of KHAPE security: Game 4

actions on these sessions, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by marking these sessions as actively attacked. If $\mathsf{SIM}_{\mathsf{AKE}}$ sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, \alpha)$ on activey attacked session, its output key k is set to $R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \alpha)$ where $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ are the keys this session runs under, which are $(B^{\mathsf{uid}}_{\mathsf{S}}, A^{\mathsf{uid}}_{\mathsf{S}})$ for S, and keys (A, B) defined by $\mathsf{IC.D}(pw, e')$ for C. The game must also emulate $\mathsf{SessionKey}$ interface of $\mathcal{F}_{\mathsf{khAKE}}$ and let $\mathsf{SIM}_{\mathsf{AKE}}$ evaluate $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', \alpha)$ for any $pk \in PK_{\mathsf{P}}$ and any $pk' \in CPK$. (Note that all sessions emulated by $\mathsf{SIM}_{\mathsf{AKE}}$ run on public keys pk' which are created by the Init interface.) Set PK_{S} contains only one key, $B^{\mathsf{uid}}_{\mathsf{S}}$, while set PK_{C} contains $A^{\mathsf{uid}}_{\mathsf{S}}$ and all keys A' created by $\mathsf{IC.D}$ queries. Set CPK consists of $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$, because these were compromised in $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ initialization, which used the corresponding private keys, and all client-side keys A' generated in $\mathsf{IC.D}$ queries, because each $\mathsf{IC.D}$ query creates and immediately compromises key A', since it needs to embed the corresponding private key a' into $\mathsf{IC.D}$ output. Finally, if $\mathsf{SIM}_{\mathsf{AKE}}$ sends NewKey on non-attacked session, the game emulates $\mathcal{F}_{\mathsf{khAKE}}$ by issuing random keys to such sessions except if $\mathsf{C}^{\mathsf{sid}}$ runs under key pair $(A', B') = (A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}})$, which matches the key pair used by $\mathsf{S}^{\mathsf{sid}}$, in which case the game copies the key output by the session which terminates first into the key output by the session which terminates second. The rest of the code is as in Game 3: C uses its key k_1 to compute authenticator $\tau = \mathsf{prf}(k_1, 1)$ and its local output $K_1 = \mathsf{prf}(k_1, 0)$, while S uses its key k_2 to verify the incoming authenticator τ' and outputs $K_2 = \mathsf{prf}(k_2, 0)$ if $\tau' = \mathsf{prf}(k_2, 1)$ and $K_2 = \bot$ otherwise.

The one case where a party might not run AKE on keys generated via a call to $\mathsf{SIM}_{\mathsf{AKE}}$ is client session C which receives e' which was output by $\mathsf{IC.E}(pw, x')$ for some x' and pw matching the password input to $\mathsf{C}^{\mathsf{sid}}$. In this case $\mathsf{C}^{\mathsf{sid}}$ runs AKE on $(a, B) = \mathsf{map}^{-1}(x')$, and since wlog these keys are chosen by the adversary and not by $\mathsf{SIM}_{\mathsf{AKE}}$, we cannot outsource that execution to $\mathsf{SIM}_{\mathsf{AKE}}$. As we said above, functionality $\mathcal{F}_{\mathsf{khAKE}}$ does not admit honest parties running AKE on arbitrary private keys a, hence $\mathsf{SIM}_{\mathsf{AKE}}$ does not have an interface to simulate such executions. In Game 4 such AKE instances are executed as in Game 3: This is the case in step (2b) in Figure 12.

Since Game 4 and Game 3 are identical except for replacing real-world AKE executions with the game emulating functionality $\mathcal{F}_{\mathsf{khAKE}}$ interacting with $\mathsf{SIM}_{\mathsf{AKE}}$, it follows that $|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\mathrm{ake}}(\mathsf{SIM}_{\mathsf{AKE}})$

GAME 5 *(delay $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$ generation until password compromise)*: In Game 4 keys $A^{\mathsf{uid}}_{\mathsf{S}}, B^{\mathsf{uid}}_{\mathsf{S}}$ are initialized and compromised in $\mathsf{StorePwdFile}$, in Game 5 we postpone these steps until password compromise. This change can be done in several steps.
For the first step denoted as Game 5(a), we remove compromising $B^{\mathsf{uid}}_{\mathsf{S}}$ and setting $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$ in $\mathsf{StorePwdFile}$, and delay them to $\mathsf{StealPwdFile}$. $\mathcal{Z}$ cannot notice this change because in Game 4, only $\mathsf{StealPwdFile}$ will use $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$, and compromising $B^{\mathsf{uid}}_{\mathsf{S}}$ to get $b^{\mathsf{uid}}_{\mathsf{S}}$ is not needed except when generating $\mathsf{file}[\mathsf{uid}, \mathsf{S}]$.
In Game 5(b) we make a change in $\mathsf{IC.D}$, that if $y \neq e^{\mathsf{uid}}_{\mathsf{S}}$ then $x' \leftarrow_{\mathrm{R}} X' \setminus \mathsf{T}^{pw}_{\mathsf{IC}}.X'$, while in Game 4, $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$ for randomly initialized (a, B), with restriction that this x' hasn't been mapped before. The divergence

Initialize simulator $\mathsf{SIM}_{\mathsf{AKE}}$, empty table T_{IC}, and lists $CPK, PK_{\mathsf{C}}, PK_{\mathsf{S}}$.

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}})$ to S: Pick $e_{\mathsf{S}}^{\mathsf{uid}} \leftarrow_{\mathrm{R}} Y$, mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ as fresh
- On new$^{(!)}$ (pw, x') to $\mathsf{IC.E}$: Output $y \leftarrow_{\mathrm{R}} Y \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.Y$, add (pw, x', y) to T_{IC}
- On new$^{(!)}$ (pw, y) to $\mathsf{IC.D}$:
  1. If $y \neq e_{\mathsf{S}}^{\mathsf{uid}}$ for any $(\mathsf{S}, \mathsf{uid})$ then pick $x' \leftarrow_{\mathrm{R}} X' \setminus \mathsf{T}_{\mathsf{IC}}^{pw}.X'$
  2. If $y = e_{\mathsf{S}}^{\mathsf{uid}}$ for some $(\mathsf{S}, \mathsf{uid})$ then:
     (a) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is fresh or $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$ then record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw\rangle$, initialize A and B via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S}
     (b) If $pw_{\mathsf{S}}^{\mathsf{uid}}$ is $\mathsf{compromised}$ and $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$ set $(A, B) \leftarrow (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$
     In both cases (a) and (b), set $pk_{\mathsf{S}}^{\mathsf{uid}}(pw) \leftarrow (A, B)$, define a as $\mathsf{SIM}_{\mathsf{AKE}}$'s response to $(\mathsf{Compromise}, A)$, add A to CPK, and set $x' \leftarrow_{\mathrm{R}} \mathsf{map}(a, B)$

  Add (pw, x', y) to T_{IC} and send back x'
- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$:Y: mark $pw_{\mathsf{S}}^{\mathsf{uid}}$compromised and:
  If $\exists$ record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}}\rangle$ then set $(A, B) \leftarrow pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$;
  Else initialize A and B via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add A to PK_{C} and B to PK_{S};
  In either case, set $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}) \leftarrow (A, B)$, define b as $\mathsf{SIM}_{\mathsf{AKE}}$'s response to $(\mathsf{Compromise}, B)$, add B to CPK, output $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e_{\mathsf{S}}^{\mathsf{uid}}, b, A)$
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ to S: Initialize function $R_{\mathsf{S}}^{\mathsf{sid}}$, set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}$, output $e_{\mathsf{S}}^{\mathsf{uid}}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ and e' to C: Initialize function $R_{\mathsf{C}}^{\mathsf{sid}}$ and:
  1. If $e' = e_{\mathsf{S}}^{\mathsf{uid}}$ then: (1) set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ if $pw = pw_{\mathsf{S}}^{\mathsf{uid}}$, otherwise set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$; (2) send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
  2. If $e' \neq e_{\mathsf{S}}^{\mathsf{uid}}$ then:
     (a) If e' was not output by $\mathsf{IC.E}$ or it was output on (pw', x') for $pw' \neq pw$, then set $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$ and send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \perp)$ to $\mathsf{SIM}_{\mathsf{AKE}}$
     (b) If e' was output by $\mathsf{IC.E}$ on (pw, x') then set $(a, B) \leftarrow \mathsf{map}^{-1}(x')$, run $\mathsf{C}^{\mathsf{sid}}$ of AKE on $(\mathsf{sid}, \mathsf{S}, a, B)$; If $\mathsf{C}^{\mathsf{sid}}$ terminates with k, output $\tau \leftarrow \mathsf{prf}(k, 1)$ and $K_1 \leftarrow \mathsf{prf}(k, 0)$

Responding to AKE messages:

- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{S})$: set $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) \leftarrow \mathsf{act}$
- On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{C})$: if $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ then $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ if $pw_{\mathsf{S}}^{\mathsf{uid}}$ is $\mathsf{compromised}$, otherwise $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) \leftarrow \mathsf{rnd}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{C}, \alpha)$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ set $k_1 \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}}, \alpha)$, output $\tau \leftarrow \mathsf{prf}(k_1, 1)$
  2. Otherwise output $\tau \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$
- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{S}, \alpha)$ and τ' to $\mathsf{S}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{act}$ and $\tau' = \mathsf{prf}(k_2, 1)$ for $k_2 = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, \alpha)$ where $(A, B) = pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$, then output $(K_2, \gamma) \leftarrow (\mathsf{prf}(k_2, 0), \mathsf{prf}(k_2, 2))$
  2. If $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$ and τ' was generated by $\mathsf{C}^{\mathsf{sid}}$ where $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$, then output $K_2 \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and $\gamma \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$
  3. In all other cases output $(K_2, \gamma) \leftarrow (\perp, \perp)$
- On γ' to $\mathsf{C}^{\mathsf{sid}}$:
  1. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ and $\gamma' = \mathsf{prf}(k_1, 2)$, output $K_1 \leftarrow \mathsf{prf}(k_1, 0))$
  2. If $\mathsf{flag}(\mathsf{C}^{\mathsf{sid}}) = \mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ and γ' was generated by $\mathsf{S}^{\mathsf{sid}}$ for $\mathsf{S}^{\mathsf{sid}}$ s.t. $\mathsf{flag}(\mathsf{S}^{\mathsf{sid}}) = \mathsf{hbc}$, output K_1 equal to the key K_2 output by $\mathsf{S}^{\mathsf{sid}}$
  3. In all other cases output $K_1 \leftarrow \perp$
- On $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{P}, pk, pk', \alpha)$: send $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', \alpha)$ if $pk \in PK_{\mathsf{P}}$, $pk' \in CPK$

Fig. 13. KHAPE: $\mathcal{Z}$'s view of ideal-world interaction (Game 7)

this change introduces is due to the statistical distance $q_{\mathsf{IC}}\epsilon_{\mathsf{map}}$ between random IC domain elements and images of map on random X elements.
Then in Game 5(c) we remove compromising $A_{\mathsf{S}}^{\mathsf{uid}}$, setting x' and adding $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ to T_{IC} in $\mathsf{StorePwdFile}$, and delay them to new$^{(!)}$ (pw, y) to $\mathsf{IC.D}$. After this change, in $\mathsf{StorePwdFile}$ we now only initialize $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ and pick $e_{\mathsf{S}}^{\mathsf{uid}}$. Since $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ is no longer added to T_{IC} in $\mathsf{StorePwdFile}$, query $(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ is now new$^{(!)}$ to $\mathsf{IC.D}$, and $\mathsf{IC.D}$ responds by compromising $A_{\mathsf{S}}^{\mathsf{uid}}$, setting corresponding x' and adding $(pw_{\mathsf{S}}^{\mathsf{uid}}, x', e_{\mathsf{S}}^{\mathsf{uid}})$ to T_{IC}. For any other queries, $\mathsf{IC.D}$ reacts same as in Game 5(b). Game 5(c) and Game 5(b) is identical since we only postpone executing those steps removed from $\mathsf{StorePwdFile}$.
In Game 5(d) we further remove usage of $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ when responding to AKE messages, except for input to $R_{\mathsf{P}}^{\mathsf{sid}}$ in actively attacked sessions. We change $\mathsf{hbc}(A, B)$ in Game 5(c) to $\mathsf{hbc}_{\mathsf{S}}^{\mathsf{uid}}$ if $(A, B) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, and rnd otherwise. Similarly we change $\mathsf{act}(A, B)$ in Game 5(c) to $\mathsf{act}_{\mathsf{S}}^{\mathsf{uid}}$ if $(A, B) = (A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, which corresponds to password compromise, and rnd otherwise.
Finally, in Game 5(e) we remove steps of initializing $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via $\mathsf{SIM}_{\mathsf{AKE}}$ in $\mathsf{StorePwdFile}$ and delay them to $\mathsf{StealPwdFile}$ or $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$, depending on which happens first. In order to set $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ only after $\mathcal{A}$ finds $pw_{\mathsf{S}}^{\mathsf{uid}}$ after successful offline dictionary attack, we first mark $pw_{\mathsf{S}}^{\mathsf{uid}}$ fresh in $\mathsf{StorePwdFile}$, and mark it $\mathsf{compromised}$ in $\mathsf{StealPwdFile}$.
If $\mathcal{A}$ first runs $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$, we initialize $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ via Init calls to $\mathsf{SIM}_{\mathsf{AKE}}$, add $A_{\mathsf{S}}^{\mathsf{uid}}$ to PK_{C} and $B_{\mathsf{S}}^{\mathsf{uid}}$ to PK_{S}, and later upon query $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$, since $pw_{\mathsf{S}}^{\mathsf{uid}}$ is marked $\mathsf{compromised}$, we retrieve $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ and compromise $A_{\mathsf{S}}^{\mathsf{uid}}$. In the other case, if $\mathsf{IC.D}(pw_{\mathsf{S}}^{\mathsf{uid}}, e_{\mathsf{S}}^{\mathsf{uid}})$ runs first, which means at this moment $pw_{\mathsf{S}}^{\mathsf{uid}}$ must be fresh since $\mathsf{StealPwdFile}$ is not called yet, we init $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$, record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, and save this $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ into $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$, and later if $\mathcal{A}$ runs $\mathsf{StealPwdFile}$ and there exists record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw_{\mathsf{S}}^{\mathsf{uid}} \rangle$, then retrieve $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ from $pk_{\mathsf{S}}^{\mathsf{uid}}(pw_{\mathsf{S}}^{\mathsf{uid}})$. In addition we also record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ upon $\mathsf{IC.D}(pw, e_{\mathsf{S}}^{\mathsf{uid}})$ if $pw \neq pw_{\mathsf{S}}^{\mathsf{uid}}$. Game 5(e) is identical to Game 5(d) since we only postpone $(A_{\mathsf{S}}^{\mathsf{uid}}, B_{\mathsf{S}}^{\mathsf{uid}})$ initialization. Thus we conclude: $|\Pr[\mathsf{G5}] - \Pr[\mathsf{G4}]| \leq q_{\mathsf{IC}}\epsilon_{\mathsf{map}}$

GAME 6 *(replace* prf *output with random string in passive sessions)*: In Game 5, in passive sessions $k_1 = k_2$ which equal to $\{0,1\}^\kappa$, and τ, γ, K_2 are all derived from prf of k_1 or k_2. In Game 6 we remove usage of prf and directly assign random elements of $\{0,1\}^\kappa$ to these values. In addition, we further remove usage of k_1 and k_2, and instead copy K_2 to K_1 as in Game 5 where $K_1 = K_2$(and $k_1 = k_2$). Since there're at most q_{ses} such sessions, and from security of prf, the difference between Game 5 and Game 6 is negligble to $\mathcal{Z}$, i.e. $|\Pr[\mathsf{G6}] - \Pr[\mathsf{G5}]| \leq q_{\text{ses}}\epsilon_{\text{kdf}}^{\mathcal{Z}}(\mathsf{SIM}_{\mathsf{AKE}})$

GAME 7 *(Ideal-world game)*: This is the ideal-world interaction, i.e. an interaction of environment $\mathcal{Z}$ with simulator SIM and functionality $\mathcal{F}_{\mathsf{aPAKE}}$, shown in Figure 13.

Observe that Game 6 is identical to the ideal-world Game 7. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof of Theorem 4.

7 Concrete aPAKE Instantiation: KHAPE-HMQV

- global hash functions $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$, $\mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_p$
- group $\mathbb{G}$ of prime order p with generator g
- cipher $(\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D})$ on space $\mathbb{Z}_p \times \mathbb{G}$ (see also page 25)

Password File Initialization on S's input $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$:

S picks two fresh AKE keys (a, A) and (b, B), sets $e \leftarrow \mathsf{IC}^*.\mathsf{E}(pw, (a, B))$
S stores $\mathsf{file}[\mathsf{uid}, \mathsf{S}] \leftarrow (e, b, A)$ and discards all other ephemeral values

C on $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$

S on $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$

$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $X \leftarrow g^x$ | $y \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $Y \leftarrow g^y$

$(a, B) \leftarrow \mathsf{IC}^*.\mathsf{D}(pw, e)$ $\quad \xleftarrow{e,\ Y} \quad$ $(e, b, A) \leftarrow \mathsf{file}[\mathsf{uid}, \mathsf{S}]$

$d_\mathsf{C} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X)$
$e_\mathsf{C} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$

$\sigma_\mathsf{C} \leftarrow (Y \cdot B^{e_\mathsf{C}})^{x + d_\mathsf{C} \cdot a}$

$k_1 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma_\mathsf{C})$
$\tau \leftarrow \mathsf{prf}(k_1, 1)$ $\quad \xrightarrow{\tau,\ X} \quad$ $d_\mathsf{S} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X)$
$e_\mathsf{S} \leftarrow \mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$

$\sigma_\mathsf{S} \leftarrow (X \cdot A^{d_\mathsf{S}})^{y + e_\mathsf{S} \cdot b}$

$k_2 \leftarrow \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma_\mathsf{S})$

$\gamma \leftarrow \perp$ if $\tau \neq \mathsf{prf}(k_2, 1)$
else $\gamma \leftarrow \mathsf{prf}(k_2, 2)$ $\quad \xleftarrow{\gamma}$

$K_1 \leftarrow \perp$ if $\gamma \neq \mathsf{prf}(k_1, 2)$ | $K_2 \leftarrow \perp$ if $\tau \neq \mathsf{prf}(k_2, 1)$
else $K_1 \leftarrow \mathsf{prf}(k_1, 0)$ | else $K_2 \leftarrow \mathsf{prf}(k_2, 0)$
output K_1 | output K_2

Fig. 14. KHAPE with HMQV: Concrete aPAKE protocol KHAPE-HMQV

We include a concrete aPAKE protocol we call KHAPE-HMQV, which results from instantiating protocol KHAPE shown in Section 6 with HMQV as the key-hiding AKE (as proved in Section 4). The resulting protocol is shown in Figure 14. It uses only 1 fixed-base exponentiation plus 1 variable-base (multi)exponentiation for each party, and 1 ideal cipher decryption for the client. It has 3 flows if the server initiates and 4 if the client initiates. The communication costs include one group element and a κ-bit key authenticator for both sides plus an ideal cipher encryption of a field element a and another group element B from server to client. Implementations of an ideal cipher over field elements may expand the ciphertext by $\Omega(\kappa)$ bits and require a hash-to-curve operation, see Sec. 8.

While we are showing the protocol with the encryption of credentials done on the server side during password registration (initialization), this can be done interactively by the server sending its public key and the user encrypting it together with its private key under the password (or it can all be done on the client side if the client chooses the server's public key). It is important to highlight that the server needs a random independent pair of private-public keys per user. One optimization is to omit the encryption of the user's private key, and instead derive this key from the password. Our analysis can be adapted to this case.

We note that KHAPE can be made into a Strong aPAKE (saPAKE), secure against pre-computation attacks, using the technique of [39]. Namely, running an OPRF protocol on pw between client and server and deriving the credential encryption key from the output of the OPRF. In addition to providing saPAKE security, the OPRF strengthens the protocol against online client-side attacks (the attacker cannot have a pre-computed list of passwords to try) and it allows for distributing the server through a threshold OPRF. As discussed in the introduction, the break of the OPRF in the context of KHAPE voids the above benefits but does not endanger the password (a major advantage of KHAPE over OPAQUE).

8 Curve Encodings and Ideal Cipher

8.1 Quasi bijections

Protocol KHAPE encrypts group elements (server's public key pk_S) using an encryption function modeled as an ideal cipher which works over a space $\{0,1\}^n$ for some n. Thus, prior to encryption, group elements need to be encoded as bitstrings of length n to which the ideal cipher will be applied. We require such encoding, denoted map, from G to $\{0,1\}^n$ to be a bijection (or close to it) so that if e is an encryption of $g \in G$ under password pw, its decryption under a different pw' returns a random element in G. The following definition considers randomized encodings.

Definition 1. *A* randomized ε-quasi bijection map *with domain A, randomness space $R = \{0,1\}^\rho$ and range B consists of two algorithms* map *and* map^{-1}*,* $\mathsf{map} : A \times R \to B$ *and* $\mathsf{map}^{-1} : B \to A$ *with the following properties:*

1. map^{-1} *is deterministic and for all $a \in A, r \in R$,* $\mathsf{map}^{-1}(\mathsf{map}(a, r)) = a$*;*
2. map *maps the uniform distribution on $A \times R$ to a distribution on B that is ε-close to uniform.*

The term ε-close refers to a statistical distance of at most ε between the two distributions. It can also be used in the sense of computational indistinguishability, e.g., if implementing randomness using a PRG. To accommodate bijections whose randomized map from A to B may exceed a given time bound in some inputs, one can consider the range of map to include an additional element $\perp$ to which such inputs are mapped. A simpler way is to define that such inputs are mapped to a fixed element in B. The probability of inputs mapped to that value is already accounted for in the statistical distance bound ε. We use *quasi bijection* without specifying ε when we assume this value to be negligible.

Quasi bijections from field elements to bitstrings. We are interested in quasi-bijective encoding into the set $\{0,1\}^n$ over which the IC encryption works. Most mappings presented below have a field $\mathbb{Z}_q$ as the range, in which case a further transformation (preserving quasi-bijectiveness) may be needed. Note that when representing elements of $\mathbb{Z}_q$ as n-bit numbers for $n = \lceil \log\ q \rceil$, the uniform distribution on $\mathbb{Z}_q$ is ε-close to the uniform distribution over $\{0,1\}^n$ for $\varepsilon = (2^n \bmod q)/q$. So when q is very close to 2^n, one can use the bit representation of field elements directly, and this is the case for many of the standardized elliptic curves. When this is not the case, one maps $u \in \mathbb{Z}_q$ to a $(n+k)$-bit integer selected as $u + tq$ for t randomly chosen as a non-negative integer $< (2^{n+k} - u)/q$. The resulting distribution is 2^{-k}-close to the uniform distribution over $\{0,1\}^{n+k}$.

8.2 Implementing quasi-bijective encodings

We focus on the case where G is an elliptic curve. There is a large variety of well-studied quasi-bijective encodings in the literature (cf. [52, 16, 28, 11, 55]). We survey some representative examples for elliptic curve groups $EC(q)$ over fields of large prime-order q.

Note that we use both directions of these encodings in KHAPE: From pk_S to a bitstring when encrypting pk_S at the time of password registration, and from a bitstring to a curve point when the client decrypts pk_S. This means that the performance of the latter operation is more significant for the efficiency of the protocol. Fortunately this is always the more efficient direction, even though the other direction is quite efficient too for the maps discussed below.

Elligator-squared [55, 40]. This method applies to most elliptic curves and accommodates ε-quasi bijections for the *whole set of curve points* with negligible values of ε.

Curve points are encoded as a pair of field elements $(u, v) \in \mathbb{Z}_q^2$. There is a deterministic function f from $\mathbb{Z}_q$ to EC such that $P \in EC$ is represented by (u, v) if and only if $P = f(u) + f(v)$. Given a point P there is a randomized procedure R_f that returns such encoding (u, v).

In [55] (Theorem 1), it is proven that for suitable choices of f, R_f is an ε-quasi bijection into $(\mathbb{Z}_q)^2$, with $\varepsilon = O(q^{-1/2})$ (see Definition 1). Since u, v are field elements, a further bijection into bitstrings may be needed as specified in Section 8.1.

In [40], the above construction is improved by allowing both u and v to be represented directly as bit strings: u as a string of $\lfloor q \rfloor$ bits and v can be be shortened even further (the amount of shortening increases the statistical distance for the quasi bijection from EC to the distribution of bitstrings (u, v)). This encoding uses two functions f, g where a point P is recovered from (u, v) as $P = f(u) + g(v)$ (in this case, function g can be simply $g(v) = v \cdot P$).

The performance of Elligator-squared depends on the functions f, g whose cost with typical instantiations (e.g., Elligator, SWU) is dominated by a single base-field exponentiation at the cost of a fraction (≈10-15%) of a scalar multiplication. Implementing $g(v) = v \cdot P$ is also a low-cost option (also allowing to shorten v [40]). The cost of the inverse map, from a curve point to its bitstring encoding, for the curves analyzed in [55] is 3 base-field exponentiations.

Elligator2. This mapping from [11] is of more restricted applicability than Elligator-squared as it applies to a smaller set of curves (e.g., it requires an element of order 2). Yet, this class includes some of the common curves used in practice, particularly Curve25519. Eligator2 defines an injective mapping between the integers $\{0, \ldots, (q-1)/2\}$ and (about) half of the elements in the curve. To be used in our setting, it means that when generating a pair $(sk_S, pk_S{=}g^{sk_S})$ for the server during password registration, the key generation procedure will choose a random sk_S and will test if the resultant pk_S has a valid encoding under Elligator2. If so, it will keep this pair, otherwise it will choose another random pair and repeat until a representable point is found. The expected number of trials is 2 and the testing procedure is very efficient (and only used during registration, not for login).

The advantages of Ellligator2 include the use of a single field element as a point representation (which requires further expansion into a bit string only if q is not close to 2^n) and the map is injective, hence quasi-bijective with $\varepsilon = 0$ over the subset of encodable curve elements. Both directions of the map are very efficient, costing about a single base-field exponentiation (a fraction of the cost of a scalar multiplication).

Detailed implementation information for the components of the above transforms is found in [27, 11, 56]. See [7] for some comparison between Elligator2 and Elligator-squared.

8.3 Ideal Cipher Constructions

Protocol KHAPE uses an ideal cipher to encrypt group elements, specifically a pair $(sk_{\mathsf{C}}, pk_{\mathsf{S}})$ where both elements are encoded as bitstrings to fit the ideal cipher interface as described in previous subsections. Thus, we consider the input to the encryption simply as a bitstring of a given fixed length, and require implementations of ideal ciphers of sufficiently long block length. For example, the combined input length for curves of 256 bits ranges between 512 and 1024 bits. Constructions of encryption schemes that are indifferentiable from an ideal cipher have been investigated extensively in the literature. Techniques include domain extension mechanisms (e.g., to expand the block size for block ciphers, including AES) [21], Feistel networks and constructions from random oracles [25, 34, 22], dedicated constructions such as those based on iterated Even-Mansour and key alternating ciphers [24, 6, 26, 26], and basic components such as wide-input (public) random permutations [13, 12, 23]. A recent technique by McQuoid et al. [48], builds a dedicated transform that can replace the ideal cipher in cases where encryption is “one-time”, namely, keys (or cipher instances) are used to encrypt a single message (as in our protocols). They build a very efficient transform using a random oracle with just two Feistel rounds. A dedicated analysis for the use of this technique in our context is left for future work.

Acknowledgments. We thank the anonymous referees for their insightful comments.

References

1. Facebook stored hundreds of millions of passwords in plain text, `https://www.theverge.com/2019/3/21/18275837/facebook-plain-text-password-storage-hundreds-millions-users`.
2. Google stored some passwords in plain text for fourteen years, `https://www.theverge.com/2019/5/21/18634842/google-passwords-plain-text-g-suite-fourteen-years`.
3. M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In *Advances in Cryptology - CRYPTO 2020*, pages 278–307, 2020.
4. M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In *Topics in Cryptology – CT-RSA 2008*, pages 335–351. Springer, 2008.
5. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Topics in Cryptology – CT-RSA 2005*, pages 191–208. Springer, 2005.
6. E. Andreeva, A. Bogdanov, Y. Dodis, B. Mennink, and J. P. Steinberger. On the indifferentiability of key-alternating ciphers. In *Advances in Cryptology – CRYPTO 2013*, pages 531–550, 2013.
7. D. F. Aranha, P.-A. Fouque, C. Qian, M. Tibouchi, and J.-C. Zapalowicz. Binary elligator squared. In *SAC*, 2014.
8. M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key privacy in public-key encryption. In *Advances in Cryptology – ASIACRYPT 2001*. Springer, 2001.
9. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, pages 139–155. Springer, 2000.
10. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Computer Society Symposium on Research in Security and Privacy – S&P 1992*, pages 72–84. IEEE, 1992.
11. D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *ACM Conference on Computer and Communications Security – CCS 2013*, 2013.
12. D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F.-X. Standaert, Y. Todo, and B. Viguier. Gimli: a cross-platform permutation. Cryptology ePrint Archive, Report 2017/630, 2017. `http://eprint.iacr.org/2017/630`.

13. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak. In *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, 2013.
14. T. Bradley, J. Camenisch, S. Jarecki, A. Lehmann, G. Neven, and J. Xu. Password-authenticated public-key encryption. In *ACNS*, volume 11464 of *Lecture Notes in Computer Science*, pages 442–462. Springer, 2019.
15. T. Bradley, S. Jarecki, and J. Xu. Strong asymmetric PAKE based on trapdoor CKEM. In *Advances in Cryptology - CRYPTO 2019*, pages 798–825, 2019.
16. E. Brier, J.-S. Coron, T. Icart, D. Madore, H. Randriam, and M. Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *Advances in Cryptology – CRYPTO 2010*, 2010.
17. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science – FOCS 2001*, pages 136–145. IEEE, 2001.
18. R. Canetti. Universally composable signature, certification, and authentication. In *CSFW 2004*. IEEE, 2004.
19. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, pages 453–474. Springer, 2001.
20. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology – EUROCRYPT 2002*, pages 337–351. Springer, 2002.
21. J.-S. Coron, Y. Dodis, A. Mandal, and Y. Seurin. A domain extender for the ideal cipher. In *Theory of Cryptography Conference – TCC 2010*, pages 273–289, 2010.
22. D. Dachman-Soled, J. Katz, and A. Thiruvengadam. 10-round Feistel is indifferentiable from an ideal cipher. In *Advances in Cryptology – EUROCRYPT 2016*, pages 649–678, 2016.
23. J. Daemen, S. Hoffert, G. V. Assche, and R. V. Keer. The design of Xoodoo and Xoofff. 2018:1–38, 2018.
24. Y. Dai, Y. Seurin, J. P. Steinberger, and A. Thiruvengadam. Indifferentiability of iterated Even-Mansour ciphers with non-idealized key-schedules: Five rounds are necessary and sufficient. In *Advances in Cryptology – CRYPTO 2017*, 2017.
25. Y. Dai and J. P. Steinberger. Indifferentiability of 8-round Feistel networks. In *Advances in Cryptology – CRYPTO 2016*, pages 95–120, 2016.
26. Y. Dodis, M. Stam, J. P. Steinberger, and T. Liu. Indifferentiability of confusion-diffusion networks. In *Advances in Cryptology – EUROCRYPT 2016*, pages 679–704, 2016.
27. A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves draft-irtf-cfrg-hash-to-curve, `https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/`, June 2020.
28. P.-A. Fouque, A. Joux, and M. Tibouchi. Injective encodings to elliptic curves. In *Australasia Conference on Information Security and Privacy – ACISP 2013*, 2013.
29. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology – CRYPTO 2006*, pages 142–159. Springer, 2006.
30. S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):230–268, 1999.
31. F. Hao and S. F. Shahandashti. The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585, 2014. `http://eprint.iacr.org/2014/585`.

32. J. Hesse. Separating symmetric and asymmetric password-authenticated key exchange. In C. Galdi and V. Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 579–599. Springer, 2020.
33. D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. https://eprint.iacr.org/2017/604.
34. T. Holenstein, R. Künzler, and S. Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *STOC 2011*, 2011.
35. J. Y. Hwang, S. Jarecki, T. Kwon, J. Lee, J. S. Shin, and J. Xu. Round-reduced modular construction of asymmetric password-authenticated key exchange. In *Security and Cryptography for Networks – SCN 2018*, pages 485–504. Springer, 2018.
36. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC'89*, pages 44–61, 1989.
37. D. P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.
38. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In *Applied Cryptology and Network Security – ACNS 2017*, pages 39–58. Springer, 2017.
39. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In *Advances in Cryptology - EUROCRYPT 2018*, pages 456–486, 2018. IACR ePrint version at http://eprint.iacr.org/2018/163.
40. T. Kim and M. Tibouchi. Invalid curve attacks in a GLS setting. In *International Workshop on Security (IWSEC 2015)*, 2015.
41. H. Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *1996 Internet Society Symposium on Network and Distributed System Security (NDSS)*, pages 114–127, 1996.
42. H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Advances in Cryptology – CRYPTO 2003*, pages 400–425. Springer, 2003.
43. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.
44. H. Krawczyk, D. Bourdrez, K. Lewi, and C. Wood. The opaque asymmetric pake protocol, draft-irtf-cfrg-opaque, `https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque/`, 2021.
45. P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. `http://eprint.iacr.org/2001/057`.
46. M. Marlinspike. Simplifying OTR deniability, `https://signal.org/blog/simplifying-otr-deniability/`, 2013.
47. M. Marlinspike and T. Perrin. The X3DH key agreement protocol, `https://signal.org/docs/specifications/x3dh/`, 2016.
48. I. McQuoid, M. Rosulek, and L. Roy. Minimal symmetric PAKE and 1-out-of-n OT from programmable-once public functions. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on*

Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. `https://eprint.iacr.org/2020/1043`.
49. NIST Information Technology Lab. Post-quantum cryptography, `https://csrc.nist.gov/projects/post-quantum-cryptography`.
50. D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In *ASIACCS 17*, pages 301–312. ACM Press, 2017.
51. J. Schmidt. Requirements for password-authenticated key agreement (PAKE) schemes, `https://tools.ietf.org/html/rfc8125`, Apr. 2017.
52. A. Shallue and C. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANTS*, 2006.
53. V. Shoup. Security analysis of SPAKE2+. *IACR Cryptol. ePrint Arch.*, 2020:313, 2020.
54. N. Sullivan, H. Krawczyk, O. Friel, and R. Barnes. Opaque with tls 1.3, draft-sullivan-tls-opaque, `https://datatracker.ietf.org/doc/draft-sullivan-tls-opaque/`, Feb. 2021.
55. M. Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In *Financial Cryptography – TCC 2014*, pages 139–156, 2014.
56. R. S. Wahby and D. Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. 2019(4):154–179, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8348`.

A Standard AKE with Entity Authentication

We show that adding key confirmation to a protocol that realizes the UC key-hiding AKE functionality, defined in Section 2, converts such protocol into a secure UC AKE protocol with explicit entity authentication (AKE-EA). In particular, this result applies to the 3DH and HMQV protocols analyzed in Sections 3 and 4, respectively. We model AKE-EA as UC functionality $\mathcal{F}_{\mathsf{AKE\text{-}EA}}$ shown in Figure 16 extending prior UC formulations of AKE (cf., [20, 18]). In addition to explicit entity authentication and forward secrecy, AKE-EA captures properties such as KCI security, AKE executions on incorrect public keys, and adaptive compromise of the long-term private keys, all elements that are inherited from our kh-AKE modeling.

A compiler from kh-AKE to AKE-EA is in Figure 15. The proof of the following theorem should not be difficult to make, and it is on our "TBD" list:

Theorem 5. *Protocol of Figure 15 realizes the UC AKE-EA functionality* $\mathcal{F}_{\mathsf{AKE\text{-}EA}}$ *assuming that* prf *is a secure PRF and that the underlying key-hiding AKE protocol realizes the UC kh-AKE functionality* $\mathcal{F}_{\mathsf{khAKE}}$.

B Definition of UC aPAKE with entity authentication

We recall a notion of UC aPAKE defined by Gentry, Mackenzie, and Ramzan [29], shown in Figure 17. We present the functionality of [29] with a few notational modifications, and a simple security "upgrade" that makes the resulting notion

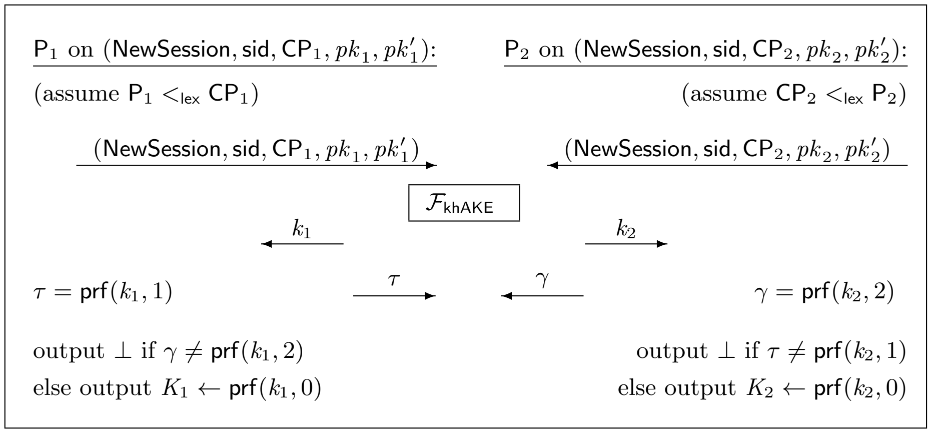

P_1 on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_1, pk_1, pk_1')$:
(assume $\mathsf{P}_1 <_{\mathsf{lex}} \mathsf{CP}_1$)
P_2 on $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_2, pk_2, pk_2')$:
(assume $\mathsf{CP}_2 <_{\mathsf{lex}} \mathsf{P}_2$)
$(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_1, pk_1, pk_1')$
$(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}_2, pk_2, pk_2')$
$\mathcal{F}_{\mathsf{khAKE}}$
k_1
k_2
$\tau = \mathsf{prf}(k_1, 1)$
τ
γ
$\gamma = \mathsf{prf}(k_2, 2)$
output $\perp$ if $\gamma \neq \mathsf{prf}(k_1, 2)$
else output $K_1 \leftarrow \mathsf{prf}(k_1, 0)$
output $\perp$ if $\tau \neq \mathsf{prf}(k_2, 1)$
else output $K_2 \leftarrow \mathsf{prf}(k_2, 0)$

Fig. 15. Compiler from kh-AKE protocol to AKE-EA protocol.

implement explicit entity authentication as part of the aPAKE protocol. We explain these changes below, and discuss the assumptions necessary for this functionality to appropriately model adversarial offline dictionary attack queries.

Changes in password file and session identifiers. The first modification we make is to rename what [29] calls a *session identifier* sid as a *user account identifier* uid, and rename what [29] calls a *sub-session identifier* ssid as a *session identifier* sid. The sid, ssid terminology of [29] complies with the general UC conventions established by Canetti [17], where the top-level state of some cryptographic application, e.g. long-term keys of an encryption or a signature scheme, is referred to as its "session", while particular instances which use these long-term keys, e.g. an authentication scheme instance, is referred to as its "sub-session". The downside of this terminology is that its application-obliviousness can unintentionally obscure the consequences of these conventions for a given application. Moreover, in many applications of UC framework the sid and ssid strings are thought of as nonces, and their implications to correctness, security, and privacy, are often overlooked.

In the context of asymmetric PAKE the sid of [29] is effectively a pointer to a password file, a.k.a. a password hash, stored by some server S for some user U. In the aPAKE functionality of [29] the user must know this sid to start the protocol, but as was pointed out by Hesse [32], the goal of (a)PAKE is to facilitate password authentication in the *password-only* setting, hence it is not clear whether it is applicable to assume that the user has to remember also this password-file-identifying string sid. On the other hand, to run aPAKE authentication, an aPAKE user U must know at least two things in addition to her password, namely (1) an identifier S of the server to which U wants to authenticate, e.g. the server's domain name, and (2) the *user ID* by which S identifies U. The UC syntax demands that identity S is the client's input in an aPAKE instance, and our terminology makes it explicit that the second necessary input is simply the user account identifier, denoted uid. A secondary effect of

- PK is the list of all public keys created via Init, initially empty
- PK_{P} is the list of all public keys created by P, initially empty for all P
- CPK is the list of all compromised keys in PK, initially empty

Keys: Initialization and Attacks

(same as in $\mathcal{F}_{\mathsf{khAKE}}$, Figure 1)

Login Sessions: Initialization and Attacks

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk, pk_{\mathsf{CP}})$ from P:

If $pk \in PK_{\mathsf{P}}$ and there is no prior session record $\langle \mathsf{sid}, \mathsf{P}, \cdot, \cdot, \cdot, \cdot \rangle$ then:

- create session record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, \perp \rangle$ marked fresh
- send $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathcal{A}$

On $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{A}$:

If session record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, \perp \rangle$ is fresh then do the following: If $pk_{\mathsf{CP}} \in CPK$ or $pk_{\mathsf{CP}} \notin PK$ then re-label it $\mathsf{compromised}$, else re-label it $\mathsf{interfered}$

Login Sessions: Key Establishment

On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, k^*)$ from $\mathcal{A}$:

If $\exists$ session record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, \perp \rangle$ then:

- If rec is marked $\mathsf{compromised}$ then set $k \leftarrow k^*$
- If rec is marked $\mathsf{interfered}$ then set $k \leftarrow \perp$
- If rec is marked fresh then:
  - If there is record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk_{\mathsf{CP}}, pk, k' \rangle$ marked fresh then:
    - If $k' \neq \perp$ then set $k \leftarrow k'$
    - If $k' = \perp$ then set $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$
  - Otherwise set $k \leftarrow \perp$
- Update rec to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk, pk_{\mathsf{CP}}, k \rangle$ and output (sid, k) to P

Fig. 16. Functionality $\mathcal{F}_{\mathsf{AKE\text{-}EA}}$: AKE with Entity Authentication

renaming sid as uid is that the word "session" can now be used in a way that matches standard KE terminology, i.e. it is an on-line authentication protocol instance. Each such instance is identified by a *session identifier* nonce, denoted sid, effectively replacing what in [29] was called ssid.

These changes bring up to the fore another salient issue in the formalization of [29], inherited from the conventions of [17]), namely that identifiers $\mathsf{sid}, \mathsf{ssid}$ are assumed to be public. As we have seen in the case of key-hiding AKE in Section 2, the UC session identifiers can identify long-term authentication tokens, and their privacy might be a concern. Since the aPAKE session identifier of [29] is a de-facto user account identifier uid, it matters whether or not this user account identifier is public and e.g. revealed in every authentication attempt. The aPAKE protocol KHAPE of Section 6 seems to protect uid privacy in the same way as it protects password privacy (the two are hashed together to form the AKE-layer key identifiers, see Figure 8). Our functionality $\mathcal{F}_{\mathsf{aPAKE}}$ of Figure 17, just like

Password Registration

- On $(\mathsf{StorePwdFile}, \mathsf{uid}, pw)$ from S create record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked fresh.

Stealing Password Data

- On $(\mathsf{StealPwdFile}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is no record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, return "no password file". Otherwise mark this record $\mathsf{compromised}$, and if there is a record $\langle \mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw \rangle$ then send pw to $\mathcal{A}$.
- On $(\mathsf{OfflineTestPwd}, \mathsf{S}, \mathsf{uid}, pw^*)$ from $\mathcal{A}$, then do:
  - If $\exists$ record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$, do the following: If $pw^* = pw$ then return "correct guess" to $\mathcal{A}$ else return "wrong guess."
  - Else record $\langle \mathsf{offline}, \mathsf{S}, uid, pw^* \rangle$

Password Authentication

- On $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{S}, pw)$ from C, if there is no record $\langle \mathsf{sid}, \mathsf{C}, ... \rangle$ then record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ marked fresh and send $(\mathsf{CltSession}, \mathsf{sid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$.
- On $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{C}, \mathsf{uid})$ from S, if there is no record $\langle \mathsf{sid}, \mathsf{S}, ... \rangle$ then retrieve record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$, and if it exists then create record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, pw, 1 \rangle$ marked fresh and send $(\mathsf{SvrSession}, \mathsf{sid}, \mathsf{S}, \mathsf{C}, \mathsf{uid})$ to $\mathcal{A}$.

Active Session Attacks

- On $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{P}, pw^*)$ from $\mathcal{A}$, if there is a record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role} \rangle$ marked fresh, then do: If $pw^* = pw$ then mark it $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."
- On $(\mathsf{Impersonate}, \mathsf{sid}, \mathsf{C}, \mathsf{S}, \mathsf{uid})$ from $\mathcal{A}$, if there is a record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 0 \rangle$ marked fresh, then do: If there is a record $\langle \mathsf{file}, \mathsf{S}, \mathsf{uid}, pw \rangle$ marked $\mathsf{compromised}$ then mark $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, pw, 1 \rangle$ $\mathsf{compromised}$ and return "correct guess" to $\mathcal{A}$; else mark it $\mathsf{interrupted}$ and return "wrong guess."

Key Generation and Authentication

- On $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, K^*)$ from $\mathcal{A}$, if there is a record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{P}', pw, \mathsf{role} \rangle$ not marked $\mathsf{completed}$, then do:
  - If rec is marked $\mathsf{compromised}$ set $K \leftarrow K^*$;
  - Else if $\mathsf{role} = 0$, rec is fresh, there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 1 \rangle$ s.t. $\mathcal{F}_{\mathsf{aPAKE}}$ sent (sid, K') to P' while that record was marked fresh, set $K \leftarrow K'$;
  - Else if $\mathsf{role} = 1$, rec is fresh, there is record $\langle \mathsf{sid}, \mathsf{P}', \mathsf{P}, pw, 0 \rangle$ which is marked fresh, pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$;
  - Else set $K \leftarrow \perp$.

  Finally, mark rec as $\mathsf{completed}$ and send output (sid, K) to P.

Fig. 17. $\mathcal{F}_{\mathsf{aPAKE}}$: asymmetric PAKE with explicit C-to-S authentication the aPAKE functionality of [29], does not model this privacy protection (e.g. $\mathsf{CltSession}$ and $\mathsf{SvrSession}$ messages reveal uid to $\mathcal{A}$), and we leave extending it so that to capture this property to future work.

Other notational changes. Since we view uid as a pointer to a long-term secret material on the server, we put it as a last parameter in several $\mathcal{F}_{\mathsf{aPAKE}}$ commands. However, once sessions are established we use the (party,sessionID) pair $(\mathsf{P}, \mathsf{sid})$ as the unique pointer to any aPAKE session $\mathsf{P}^{\mathsf{sid}}$, while in [29] these handles were $(\mathsf{sid}, \mathsf{ssid})$, aligned with conventions of [17]. In other cosmetic changes [29] identifies password files by $(\mathsf{S}, \mathsf{U}, \mathsf{sid})$ tuple, which in our notation would be $(\mathsf{S}, \mathsf{U}, \mathsf{uid})$, but it is not clear identity U is meaningful: If party identity P in a UC protocol is interpreted as a domain name or an IP address, either of these identities would be unstable for a web used authenticating to a web service, and in our notation uid is the sole long-term identifier, for a given server S, of a user, hence in $\mathcal{F}_{\mathsf{aPAKE}}$ in Figure 17 we identify password files by $(\mathsf{S}, \mathsf{uid})$ tuple.

Entity authentication. Our AKE-to-aPAKE compiler, protocol KHAPE of Section 6, requires both parties to send a key confirmation message for security, but this message also upgrades the functionality of the resulting aPAKE by implementing explicit entity authentication within the protocol. To capture this aPAKE property we modify the aPAKE mode of [29] by incorporating entity authentication. This results from the following modification of the $\mathcal{F}_{\mathsf{aPAKE}}$ key-establish rules, as modeled by the NewKey query: A fresh session $\mathsf{S}^{\mathsf{sid}}$ outputs key K which is *not* a rejection symbol $\perp$ only if session $\mathsf{C}^{\mathsf{sid}}$ runs on the same password, and $\mathsf{C}^{\mathsf{sid}}$ can output a non-$\perp$ key only by copying key K' which was output by a fresh session $\mathsf{S}^{\mathsf{sid}}$ which runs on a password file for the same password pw. The functionality in Figure 17 is customized to protocols, like ours, where S terminates first, but that assumption is used only to simplify $\mathcal{F}_{\mathsf{aPAKE}}$ syntax in NewKey query handling.

Modeling password file compromise and offline password queries. [29] attempted to model both password file compromise and offline password queries in a way akin to the way UC framework models party corruption [17]. Several subsequent works [39, 32, 53] have pointed out that this treatment poses non-standard requirements on the communication between the environment and the real-world and ideal-world security games. The main issue is that one of the main goals of a UC aPAKE model is to impose a tight bound on the amount (and timing!) of *local computation* of a real-world adversary performing offline dictionary attacks. First thing to note is that such goal seems realizable only in an idealized computation model, e.g. assuming ROM or IC, where adversary's local computation can be modeled as an interaction with an oracle. Moreover, this oracle must have some "back-channel" to the environment, to allow the environment to monitor offline password tests in the real-world (or, more precisely, in a hybrid world where e.g. a hash function or symmetric cipher are modeled as ideal oracles, respectively RO or IC). Finally, if the real-world and ideal-world executions are to be indistinguishable, then the ideal-world model $\mathcal{F}_{\mathsf{aPAKE}}$ must allow for the same environmental monitoring of offline password tests.

In this work we adopt a simplified convention for how the above is enforced by essentially following the formal conventions of [29], but with the following caveats. In the $\mathcal{F}_{\mathsf{aPAKE}}$ model in Figure 17 the ideal-world adversary can freely request password file compromises, via $\mathsf{StealPwdFile}$ and $\mathsf{offline}$ commands to $\mathcal{F}_{\mathsf{aPAKE}}$. However, this convention makes sense *only* if we assume that these commands are "environmentally controlled", i.e. that $\mathcal{F}_{\mathsf{aPAKE}}$ (or the ideal-world adversary) has a back-channel to the environment informing it of compromised files and offline dictionary queries, or that $\mathcal{F}_{\mathsf{aPAKE}}$ services these queries only if the environment sends an explicit permission for it. In the security proof of the aPAKE protocol KHAPE, in Section 2, we assume that $\mathsf{StealPwdFile}$ is a message sent by the real-world adversary, see Fig. 9, but the proof would not change if the simulator received (the permission to issue) this message instead from $\mathcal{F}_{\mathsf{aPAKE}}$ or directly from the environment. Likewise, an offline dictionary query pw against password file tagged by $(\mathsf{S}, \mathsf{uid})$ is equated with a query to RO hash oracle H on inputs $(\mathsf{S}, \mathsf{uid}, pw, s)$. In the KHAPE proof the simulator then sends $(\mathsf{offline}, \mathsf{S}, \mathsf{uid}, pw)$ to $\mathcal{F}_{\mathsf{aPAKE}}$, and here again the proof would not change if this message was sent instead directly to the environment and/or the simulator had to wait to process it if it receives an environmental "permit" to do so.

C Proof of HMQV as Key-Hiding AKE

We present the proof of Theorem 2 from Section 4.

Proof. We describe how the security proof for 3DH should be adapted to the case of HMQV. The two proofs follow the same template. In particular, the HMQV simulator algorithm SIM, shown in Figure 18, acts very similarly to the 3DH simulator in Figure 3. The proof shows the indistinguishability between the real-world game (Game 0) shown in Figure 19, which captures an interaction with parties running the HMQV protocol, and the ideal-world game (Game 7) shown in Figure 20, which is defined by a composition of SIM and functionality $\mathcal{F}_{\mathsf{khAKE}}$. The sequence of games which shows this indistinguishability is exactly the same as the sequence used in the proof of theorem 1, and below we sketch where the match is exact and how we deal with the HMQV-specific differences when they occur. In particular, in the discussions below we often re-use the notation introduced used in the proof of theorem 1.

As in the case of 3DH, for each AKE session $\mathsf{P}^{\mathsf{sid}}$ we define function $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ which is used by session $\mathsf{P}^{\mathsf{sid}}$ to compute its session key given counterparty's message Z. The definition of $R^{\mathsf{sid}}_{\mathsf{P}}$ is exactly the same as in the case of 3DH, i.e. equation (2), except the last argument, σ, is now defined using the HMQV function, $\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$. Below we define function $\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}$ for session $\mathsf{P}^{\mathsf{sid}}$ running on inputs $(\mathsf{sid}, \mathsf{CP}, pk, pk')$, i.e. pk is its own public key and pk' is the public key of the intended counterparty. Function

```
*Initialization:* Initialize an empty list $KL_{\mathsf{P}}$ for each $\mathsf{P}$

On $(\mathsf{Init}, \mathsf{P})$ from $\mathcal{F}$:
pick $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $pk \leftarrow g^{sk}$, add $(sk, pk)$ to $KL_{\mathsf{P}}$, and send $pk$ to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ send $sk$ to $\mathcal{A}$ and $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w \rangle$, send $W = g^w$ to $\mathcal{A}$

On $\mathcal{A}$'s message $Z$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, w \rangle$:
    if $\exists$ *no* record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, z \rangle$ s.t. $g^z = Z$ then send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
    send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, Z)$ to $\mathcal{F}$

On query $(\mathsf{st}, \sigma)$ to random oracle $\mathsf{H}$, for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y)$:
if $\exists\ \langle (\mathsf{st}, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, otherwise pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and:

    if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x \rangle$, $(a, A) \in KL_{\mathsf{C}}$, and tuples $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$,
    $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in $\mathsf{T_{H'}}$ s.t. $(X, \sigma) = (g^x, (Y \cdot B^e)^{x+da})$ for some $B$:
        send $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{C}, A, B, Y)$ to $\mathcal{F}$, if $\mathcal{F}$ returns $k^*$ reset $k \leftarrow k^*$

    if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, y \rangle$, $(b, B) \in KL_{\mathsf{S}}$, and tuples $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$,
    $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in $\mathsf{T_{H'}}$ s.t. $(Y, \sigma) = (g^y, (X \cdot A^d)^{y+eb})$ for some $A$:
        send $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{S}, B, A, X)$ to $\mathcal{F}$, if $\mathcal{F}$ returns $k^*$ reset $k \leftarrow k^*$

    add $\langle (\mathsf{st}, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$

On query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z)$ to random oracle $\mathsf{H}'$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ in $\mathsf{T_{H'}}$ then output $r$
else pick $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ to $\mathsf{T_{H'}}$, and output $r$
```

Fig. 18. Simulator SIM showing that HMQV realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$")

$\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}$ can be defined separately for cases for $\mathsf{P}^{\mathsf{sid}}$ playing the client-role, denoted $\mathsf{P} = \mathsf{C}$, and $\mathsf{P}^{\mathsf{sid}}$ playing the server-role, denoted $\mathsf{P} = \mathsf{S}$, as follows:

$$\begin{aligned}
\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', Y) = \;& \mathsf{cdh}_g(X, Y) \cdot \mathsf{cdh}_g(pk', X)^e \cdot \mathsf{cdh}_g(Y, pk)^d \cdot \mathsf{cdh}_g(pk, pk')^{ed} \\
\text{for } X = X^{\mathsf{sid}}_{\mathsf{C}} \text{ and } \;& d = \mathsf{H}'(\mathsf{st}, 1, X), e = \mathsf{H}'(\mathsf{st}, 2, Y), \mathsf{st} = \mathsf{sid}|\mathsf{C}|\mathsf{S} \\
\mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', X) = \;& \mathsf{cdh}_g(X, Y) \cdot \mathsf{cdh}_g(pk, X)^e \cdot \mathsf{cdh}_g(Y, pk')^d \cdot \mathsf{cdh}_g(pk, pk')^{ed} \\
\text{for } Y = Y^{\mathsf{sid}}_{\mathsf{S}} \text{ and } \;& d = \mathsf{H}'(\mathsf{st}, 1, X), e = \mathsf{H}'(\mathsf{st}, 2, Y), \mathsf{st} = \mathsf{sid}|\mathsf{C}|\mathsf{S}
\end{aligned}$$

Game 0 *(real world)*: The real-world game is shown in Figure 19.

Game 1 *(past* H *queries are irrelevent to new sessions)*: We add an abort if session $\mathsf{P}^{\mathsf{sid}}$ starts with W which appeared in some prior inputs to H. As in the case of 3DH, $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (2q_{\mathsf{H}})/p$.

Game 2 *(programming* $R^{\mathsf{sid}}_{\mathsf{P}}$ *values into* H *outputs)*: We make the same change of using random but pair-wise correlated functions $R^{\mathsf{sid}}_{\mathsf{P}}$, i.e. correlated

```
*Initialization:* Initialize an empty list $KL_{\mathsf{P}}$ for each $\mathsf{P}$

On message $\mathsf{Init}$ to $\mathsf{P}$:
pick $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $pk \leftarrow g^{sk}$, add $(sk, pk)$ to $KL_{\mathsf{P}}$, and output $(\mathsf{Init}, pk)$

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ then output $sk$

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathsf{P}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$; if $\exists\ (sk_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$, pick
$w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, sk_{\mathsf{P}}, pk_{\mathsf{CP}}, w \rangle$, output $W = g^w$

On message $Z$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, sk_{\mathsf{P}}, pk_{\mathsf{CP}}, w \rangle$:
    if $\mathsf{role} = 1$:
        set $d \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\mathrm{ord}}, 1, g^w)$ and $e \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\mathrm{ord}}, 2, Z)$
        set $\sigma \leftarrow (Z \cdot pk_{\mathsf{CP}}^{e})^{w + d \cdot sk_{\mathsf{P}}}$
    if $\mathsf{role} = 2$:
        set $d \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\mathrm{ord}}, 1, Z)$ and $e \leftarrow \mathsf{H}'(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}\}_{\mathrm{ord}}, 2, g^w)$
        set $\sigma \leftarrow (Z \cdot pk_{\mathsf{CP}}^{d})^{w + e \cdot sk_{\mathsf{P}}}$
    set $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W, Z, \sigma\}_{\mathrm{ord}})$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On $\mathsf{H}$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T}_{\mathsf{H}}$ then output $k$
else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T}_{\mathsf{H}}$, and output $k$

On $\mathsf{H}'$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ in $\mathsf{T}_{\mathsf{H}'}$ then output $r$
else pick $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ to $\mathsf{T}_{\mathsf{H}'}$, and output $r$
```

Fig. 19. HMQV: Environment's view of real-world interaction (Game 0)

as in equation (3), and programming $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ values into outputs of $\mathsf{H}(\mathsf{sid}, \{\mathsf{C}, \mathsf{S}, W, Z\}_{\mathrm{ord}}, \sigma)$ if W matches the value sent by $\mathsf{P}^{\mathsf{sid}}$ and $\sigma = \mathsf{HMQV}_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$. As in the case of 3DH we need to argue that if the same hash query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$, for $(X, Y) = (X_{\mathsf{C}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}})$, matches both the client-side equation and the server-side equation, i.e. if

$$\sigma = \mathsf{HMQV}_{\mathsf{C}}^{\mathsf{sid}}(A, B', Y) = \mathsf{HMQV}_{\mathsf{S}}^{\mathsf{sid}}(B, A', X)$$

where $A \in PK_{\mathsf{C}}, B' \in PK^{+}(\mathsf{C}^{\mathsf{sid}}), B \in PK_{\mathsf{S}}, A' \in PK^{+}(\mathsf{S}^{\mathsf{sid}})$, as defined in the 3DH proof, then either condition programs the same value into H output.

In the case of 3DH the corresponding equation implied that both parties must use correct counterparty keys, i.e. that $(A', B') = (A, B)$, in which case constraint (3) on $R_{\mathsf{C}}^{\mathsf{sid}}$ and $R_{\mathsf{S}}^{\mathsf{sid}}$ implies that either condition programs H to the same value.

In the case of HMQV the above equation can hold even if $(A', B') \neq (A, B)$, but it can occur with only negligible probability. The constraint above implies:

$$\left(Y \cdot B'^{e}\right)^{x+da} = \left(X \cdot A'^{d}\right)^{y+eb} \tag{4}$$

```
$\textit{Initialization:}$ Initialize empty lists: $PK$, $CPK$, and $KL_{\mathsf{P}}$ for all $\mathsf{P}$

On message $\mathsf{Init}$ to $\mathsf{P}$:
set $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $pk \leftarrow g^{sk}$, send $(\mathsf{Init}, pk)$ to $\mathsf{P}$, add $pk$ to $PK$ and $(sk, pk)$ to $KL_{\mathsf{P}}$

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ add $pk$ to $CPK$ and output $sk$

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathsf{P}$:
if $\exists\ (sk, pk_{\mathsf{P}}) \in KL_P$ then:
    initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$
    if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
    pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$ as $\mathsf{fresh}$, output $W = g^w$

On message $Z$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \perp \rangle$:
    if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \mathsf{role}', z, k' \rangle$ s.t. $g^z = Z$
        then if $\mathsf{rec}'$ is $\mathsf{fresh}$, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$, and $k' \neq \perp$:
            then $k \leftarrow k'$
            else $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
        else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z)$ and re-label $\mathsf{rec}$ as $\mathsf{interfered}$
    update $\mathsf{rec}$ to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, k \rangle$, output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On $\mathsf{H}$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$ and:
  1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, \cdot \rangle$, $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$ and $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in
     $\mathsf{T_{H'}}$ s.t. $(X, \sigma) = (g^x, (Y \cdot B^e)^{x + d \cdot a})$ for some $(a, A) \in KL_{\mathsf{C}}$ and $B$ s.t.
     $B \in CPK$ or $B \notin PK$, then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y)$
  2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, y, \cdot \rangle$, $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, X), d \rangle$ and $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y), e \rangle$ in
     $\mathsf{T_{H'}}$ s.t. $(Y, \sigma) = (g^y, (X \cdot A^d)^{y + e \cdot b})$ for some $(b, B) \in KL_{\mathsf{S}}$ and $A$ s.t.
     $A \in CPK$ or $A \notin PK$, then reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, X)$
add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$

On $\mathsf{H}'$ query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z)$:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ in $\mathsf{T_{H'}}$ then output $r$
else pick $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, add $\langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, n, Z), r \rangle$ to $\mathsf{T_{H'}}$, and output $r$
```

Fig. 20. HMQV: Environment's view of ideal-world interaction (Game 7)

where $d = \mathsf{H}'(\mathsf{st}, 1, X)$ and $e = \mathsf{H}'(\mathsf{st}, 2, Y)$ and $(X, Y) = (g^x, g^y)$. Note that equation (4) holds if and only if $(y + eb')(x + da) = (x + a'd)(y + eb)$, where a', b' are the discrete logarithms of resp. A, B. This can hold even if $(a', b') \neq (a, b)$, hence in the case of HMQV we will add an abort in the case equation (4) holds and $(A', B') \neq (A, B)$. Note that the adversary must choose the counterparty key $pk' = B'$ for session $\mathsf{C}^{\mathsf{sid}}$ before $\mathsf{C}^{\mathsf{sid}}$ starts and picks x. Likewise $pk' = A'$ for session $\mathsf{S}^{\mathsf{sid}}$ must be chosen before $\mathsf{S}^{\mathsf{sid}}$ picks y. Therefore the last value to be chosen is either x or y, i.e. either x or y are randomly sampled after (a, b, a', b') are all fixed. If x is chosen after (a, b, a', b', y) then its choice determines $d = \mathsf{H}'(\mathsf{st}, 1, g^x)$, but since H' is a random oracle, the probability that d satisfies equation (4) is $1/p$. Since a symmetric argument holds in the case y (and e) are chosen last, it follows that:

$$|\Pr[\mathsf{G2}] - \Pr[\mathsf{G1}]| \leq q_{\text{ses}}/p$$

Game 3 *(direct programming of session keys using random functions $R_{\mathsf{P}}^{\mathsf{sid}}$)*: This step is identical as in the case of 3DH, and $\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$

Game 4 *(abort on H queries for passive sessions)*: As in the case of the proof for 3DH we add an abort whenever oracle H triggers evaluate of $R_{\mathsf{P}}^{\mathsf{sid}}(pk, pk', Z)$ for any pk, pk' and $Z = W_{\mathsf{CP}}^{\mathsf{sid}}$ where $\mathsf{CP}^{\mathsf{sid}}$ is a matching session of $\mathsf{P}^{\mathsf{sid}}$, and likewise define as Bad the event that such query is made. W.l.o.g. we can consider these sessions using arbitrary pk's, e.g. B' for $\mathsf{C}^{\mathsf{sid}}$ and A' for $\mathsf{S}^{\mathsf{sid}}$, which might or might not equal to the correct public key of the intended counterparty on the respective session.

As in the case of 3DH we show that solving Gap CDH can be reduced to causing event Bad in this game, but the full reduction $\mathcal{R}'$ that exihbits that uses rewinding over two executions of a subsidiary reduction $\mathcal{R}$, which works as follows. We argue reduction $\mathcal{R}$ assuming that event Bad occurs for a client-side function $R_{\mathsf{C}}^{\mathsf{sid}}$, because the case for a server-side function $R_{\mathsf{S}}^{\mathsf{sid}}$ is symmetric.

Reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{Y})$ and embeds it in a randomized way in the messages of all simulated parties, i.e. it sends $X = \bar{X}^s$ and $Y = \bar{Y}^t$ for random s and t shifts on behalf of resp. $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ sections, just like in the 3DH case. Otherwise it emulates the security game, in particular it knows all the key pairs (a, A) and (b, B). Although $\mathcal{R}$ does not know $x = s \cdot \bar{x}$ and $y = t \cdot \bar{y}$ corresponding to these messages, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{y} = \mathsf{dlog}_g(\bar{Y})$, reduction $\mathcal{R}$ can use the DDH oracle to emulate H queries, i.e. to test if

$$\sigma = \mathsf{HMQV}_{\mathsf{C}}^{\mathsf{sid}}(A, B', Y) = (Y(B')^e)^{x+da} = \mathsf{cdh}_g(X, Y(B')^e) \cdot (Y(B')^e)^{ad}$$

for any key B', any key $A = g^a$ of $\mathsf{C}^{\mathsf{sid}}$, and $(X, Y) = (\bar{X}^s, \bar{Y}^t)$ sent by resp. $\mathsf{C}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$. Symmetrically $\mathcal{R}$ can test if $\sigma = \mathsf{HMQV}_{\mathsf{S}}^{\mathsf{sid}}(B, A', X)$.

Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3, in which case $\mathcal{R}$ can compute $v = \mathsf{cdh}_g(X, Y(B')^e)$, assuming Bad occurs for a client-side equation. Denote this (e, v) pair as (e_1, v_1), i.e. $v_1 = \mathsf{cdh}_g(X, Y(B')^{e_1})$. By the rewinding argument, as in [43], if the probability of the (client-side) Bad is ϵ, the second-layer reduction $\mathcal{R}'$ can run $\mathcal{R}$ against the adversary/environment twice, providing a fresh random output e_2 on hash query $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$. If event Bad occurs in that second execution for the same Y then $\mathcal{R}$ would extract $v_2 = \mathsf{cdh}_g(X, Y(B')^{e_2})$, in which case $\mathcal{R}'$ can compute $\mathsf{cdh}_g(X, Y) = (v_1^{e_2}/v_2^{e_1})^{1/(e_2-e_1)}$, and consequently solve for $\mathsf{cdh}_g(\bar{X}, \bar{Y}) = (\mathsf{cdh}_g(X, Y))^{1/(st)}$. By the standard rewinding argument, the probability $\mathcal{R}'$ succeeds is at least $(1/c_{\mathsf{rwnd}})\epsilon^2/q_{\mathsf{H}}$ for a small constant c_{rwnd}, which implies

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq (c_{\mathsf{rwnd}} \cdot q_{\mathsf{H}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}})^{1/2}$$

Game 5 *(random keys on passively observed sessions)*: This game change is the same as in the case of 3DH, and $\Pr[\mathsf{G5}] = \Pr[\mathsf{G3}]$

Game 6 *(decorrelating function pairs $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}}$)*: This game change is the same as in the case of 3DH, and $\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$

Game 7 *(hash computation consistent only for compromised keys)*: As in the proof for 3DH, we restrict handling H queries to only those that correspond to counterparty key pk' being either compromised or adversarial. Consequently, as in the case of 3DH, Game 7 diverges from Game 6 if event Bad occurs, defined as H query on $(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, W^{\mathsf{sid}}_{\mathsf{P}}, Z\}_{\text{ord}}, \sigma)$ for $\sigma = \mathsf{HMQV}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z)$ where $pk \in PK_{\mathsf{P}}$ and $pk' \in PK \setminus CPK$. Let Bad_n be Bad where $\mathsf{P}^{\mathsf{sid}}$ plays role $= n$. As in the case of the 3DH proof we will argue only for $n = 1$ because the other case is symmetric. Also, we will focus on sub-event $\mathsf{Bad}_1[i]$ which denotes Bad_1 occuring where $\mathsf{P}^{\mathsf{sid}}$ uses the i-th key as pk', i.e. pk' was a non-compromised public key in PK created in the i-th key initialization query. Note that $\mathsf{Bad}_1[i]$ corresponds to $\sigma = (Y \cdot (pk')^e)^{x+d\cdot a}$ where a is some private key of C and x is used on session $\mathsf{C}^{\mathsf{sid}}$, pk' equals to the honestly-generated and non-compromised public key corresponding to the i-th key record, and an arbitrary Y which adversary specifies in the hash inputs.

As in the 3DH proof we show a reduction $\mathcal{R}$ that solves a Gap *Square* DH if $\mathsf{Bad}_1[i]$ occurs. Square DH is a variant of CDH where the challenge is a single value $\bar{X}$ and the goal is to compute $\mathsf{cdh}_g(\bar{X}, \bar{X})$. It is well-known that Square DH is equivalent to CDH. As in the reduction to Gap CDH used in Game 4 above, here too we will use a subsidiary reduction $\mathcal{R}$ which computes CDH on a problem *related* to the Square DH challenge, and a top-level reduction $\mathcal{R}'$ which solves the Square DH challenge using rewinding over two executions of $\mathcal{R}$. We show the bound on the probability that $\mathcal{R}$ succeeds in terms of the probability ϵ of event $\mathsf{Bad}[i]$, and then we show the overall bound using a union bound and a symmetry of client-side and server-side equations.

Reduction $\mathcal{R}$ takes a Square DH challenge $\bar{X}$ and embeds it as $pk' \leftarrow \bar{X}$ where pk' is the public key in the i-th key record, and also embeds $\bar{X}$ into messages $X = \bar{X}^s$ sent on behalf of all client-role sessions, for random s. As in the case of 3DH $\mathcal{R}$ picks all other long-term key pairs (sk, pk) as in the original security game, and it also picks ephemeral state y of all server-side sessions. As long as pk' is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to a compromise of all other keys, and it can service H queries as follows: To test client-side σ's, i.e. if $\sigma = (Y \cdot (pk')^e)^{x+d \cdot sk_{\mathsf{C}}}$ where pk' is an arbitrary public key, Y is an arbitrary value input into the hash, $x = s \cdot \bar{x}$ is an ephemeral state of $\mathsf{C}^{\mathsf{sid}}$ unknown to $\mathcal{R}$, and sk_{C} w.l.o.g. can correspond to the i-th public key $\bar{X}$, hence also unknown to $\mathcal{R}$ (the case of any other key, where $\mathcal{R}$ knows the corresponding key sk_{C} is strictly easier), reduction $\mathcal{R}$ uses the DDH oracle to test if $\sigma = \mathsf{cdh}_g(Y \cdot (pk')^e, \bar{X}^{s+d})$. To test server-side σ's, i.e. if $\sigma = (X \cdot (pk')^d)^{y+e \cdot sk_{\mathsf{S}}}$, for arbitrary X, pk', a known ephemeral state y, and sk_{S} which again w.l.o.g. can correspond to the i-th public key $\bar{X}$, reduction $\mathcal{R}$ uses the DDH orace to test if $\sigma = (X \cdot (pk')^d)^y \cdot \mathsf{cdh}_g(X \cdot (pk')^d, \bar{X}^e)$.

As in the case of 3DH proof this emulation is perfect, so $\mathsf{Bad}_1[i]$ occurs with the same probability as in Game 6, and if does then the client-side equation for σ involves $pk' = \bar{X}$, which means that $\mathcal{R}$ computes $\sigma_1 = \mathsf{cdh}_g(Y \cdot \bar{X}^{e_1}, \bar{X}^{s+d})$, where as in the rewinding reduction in the case of Game 4 above we use e_1 to denote $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$ in the first execution of $\mathcal{R}$. Let $w = \mathsf{cdh}_g(Y, \bar{X})$ and $z = \mathsf{cdh}_g(\bar{X}, \bar{X})$, and note that $\sigma_1 = w^{s+d} \cdot z^{e_1(s+d)}$. If the second run of $\mathcal{R}$ hits the event for the same Y and embeds fresh e_2 into $\mathsf{H}'(\mathsf{sid}, \mathsf{C}, \mathsf{S}, 2, Y)$ then it computes $\sigma_2 = w^{s'+d'} \cdot z^{e_2(s'+d')}$. Since $\mathcal{R}'$ knows all the coefficients, it can solve these relations for w, z and output $z = \mathsf{cdh}_g(\bar{X}, \bar{X})$.

If i is randomly chosen and w.l.o.g. Bad_1 is at least as likely as Bad_2 then the probability of $\mathsf{Bad}[i]$ is at least $\epsilon/(2q_{\mathrm{K}})$. Therefore, by the standard rewinding argument, $\mathcal{R}$ succeeds with probability at least $(1/c_{\mathsf{rwnd}})(\epsilon/2q_{\mathrm{K}})^2/q_{\mathsf{H}}$, which implies

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq (2q_{\mathrm{K}}) \cdot (c_{\mathsf{rwnd}} \cdot q_{\mathsf{H}} \cdot \epsilon^{\mathcal{Z}}_{\text{g-cdh}})^{1/2}$$

which concludes the proof. □

D Proof of SKEME as Key-Hiding AKE

In this section we present the proof of Theorem 3. We first introduce definitions for several notions used in the formulation of the theorem.

Following the notions of key-hiding PKE [8], we define general key-privacy and perfect key-privacy property of KEM:

Definition 2. ***[KPKEM]*** *Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a key-encapsulation mechanism. Let* $b \in \{0,1\}$*. Let* A *be the adversary. Now we consider the following experiment:*

```
Experiment $\boldsymbol{Exp}^{kpkem-b}_{\mathsf{KEM},A}$
$(pk_0, sk_0) \leftarrow \mathsf{KEM.Gen}; (pk_1, sk_1) \leftarrow \mathsf{KEM.Gen}$
$(c, K) \leftarrow \mathsf{KEM.Enc}(pk_b)$
$b' \leftarrow A(sk_0, pk_0, sk_1, pk_1, c)$
Return $b'$
```

We define the advantage of the adversaries as:

$$\boldsymbol{Adv}^{kpkem}_{\mathsf{KEM},A} = \left|\Pr[\boldsymbol{Exp}^{kpkem-1}_{\mathsf{KEM},A} = 1] - \Pr[\boldsymbol{Exp}^{kpkem-0}_{\mathsf{KEM},A} = 1]\right|$$

KEM *is key-private if* $\boldsymbol{Adv}^{kpkem}_{\mathsf{KEM},A}$ *is negligible for any ppt adversary* A.

Definition 3. *[perfect **KPKEM**] Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}_1, \mathsf{Enc}_2, \mathsf{Dec})$ *be a key-encapsulation mechanism. Let* $b \in \{0,1\}$*. Let* A *be the adversary. Now we consider the following experiment:*

```
Experiment $\boldsymbol{Exp}^{perfect-kpkem-b}_{\mathsf{KEM},A}$
$(pk_0, sk_0) \leftarrow \mathsf{KEM.Gen}$; $(pk_1, sk_1) \leftarrow \mathsf{KEM.Gen}$
$c, r \leftarrow \mathsf{KEM.Enc}_1(\kappa)$, $K \leftarrow \mathsf{KEM.Enc}_2(pk_b, r)$
$b' \leftarrow A(sk_0, pk_0, sk_1, pk_1, c)$
Return $b'$
```

We also define perfect key-private KEM*, which is a stronger notion of general key-private* KEM *where* $c, r \leftarrow \mathsf{KEM.Enc}_1(\kappa)$ *and* $K \leftarrow \mathsf{KEM.Enc}_2(pk_b, r)$*, and it's clear that for such* KEM $\boldsymbol{Adv}^{kpkem}_{\mathsf{KEM},A} = 0$*. We assume* KEM*s used in the following SKEME security proof all have such property.*

We also assume KEM we use suffices the security property of One-Wayness under Plaintext-Checking-Attack, abbreviated as OW-PCA[33], where a Plaintext-Checking Oracle $PCO_{sk}(K, c)$ is defined to output a bit on whether $\mathsf{KEM.Dec}(sk, c) = K$:

Definition 4. *[**OW of KEM under PCA**] Let* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a key-encapsulation mechanism. Let* A *be the adversary. Now we consider the following experiment:*

```
Experiment $\boldsymbol{Exp}^{ow-pca}_{\mathsf{KEM},A}$
$(pk, sk) \leftarrow \mathsf{KEM.Gen}$
$c^*, r \leftarrow \mathsf{KEM.Enc}_1(\kappa), K^* \leftarrow \mathsf{KEM.Enc}_2(pk, r)$
$K' \leftarrow A^{PCO_{sk}(\cdot,\cdot)}(pk, c^*)$
Return $PCO_{sk}(K', c^*)$
```

We define the advantage of the adversaries as $\boldsymbol{Adv}^{ow-pca}_{\mathsf{KEM},A}$*, and scheme* KEM *is said to be **OW-PCA** secure if* $\boldsymbol{Adv}^{ow-pca}_{\mathsf{KEM},A}$ *is negligible for any ppt adversary* A.

Fortunately many KEMs suffice above two requirements, including plain El Gamal KEM, where the encryption generates $c = g^r$ and $K = pk^r$. And this El Gamal encryption scheme is **OW-PCA** secure based on GapDH problem, since given a public key $pk = g^{sk}$ and $(c, K) = (g^r, pk^r)$ a PCO simply checks whether $(pk = g^{sk}, c = g^r, K = pk^r)$ is a DH-triple, which is exactly a DDH Oracle. It's also perfect **KPKEM** based on its definition.

Proof of Theorem 3

Proof. We use the following definitions for any $\mathsf{P}^{\mathsf{sid}}$, for $\mathsf{P} \in \{\mathsf{C}, \mathsf{S}\}$, which always uses intended counterparty public key, i.e. $\exists\ pk_{\mathsf{CP}}$ s.t. $\mathsf{P}^{\mathsf{sid}}$ runs on $(\mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$:

```
*Initialization:* Initialize empty global list $KL$ and empty lists $KL_{\mathsf{P}}$ for each $\mathsf{P}$

On $(\mathsf{Init}, \mathsf{P})$ from $\mathcal{F}$:
set $(sk, pk) \leftarrow \mathsf{KEM.Gen}$, add $(sk, pk)$ to $KL$ and $KL_{\mathsf{P}}$, and send $pk$ to $\mathcal{F}$

On $\mathcal{Z}$'s permission to send $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$:
if $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ send $sk$ to $\mathcal{A}$ and $(\mathsf{Compromise}, \mathsf{P}, pk)$ to $\mathcal{F}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{CP})$ from $\mathcal{F}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $(e, r) \leftarrow \mathsf{KEM.Enc}_1(\kappa)$
store $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \mathsf{role}, w, e, r \rangle$ and send $(W = g^w, e)$ to $\mathcal{A}$

On $\mathcal{A}$'s message $(Z, f)$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message counts):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, \cdot, w, e, \cdot \rangle$:
    if there is *no* record $\langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, \cdot, z, f', \cdot \rangle$ s.t. $g^z = Z$ and $f = f'$ then:
        send $(\mathsf{Interfere}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{F}$
    send $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{P}, (Z, f))$ to $\mathcal{F}$

On query $(\mathsf{st}, A, B, X, c, Y, d, \sigma)$ to random oracle $\mathsf{H}$, for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$:
if $\exists\ \langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ and:
if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, 1, x, c, r \rangle$ and $(a, A) \in KL_{\mathsf{C}}$ s.t.
$(X, \sigma) = (g^x, (\mathsf{KEM.Enc}_2(B, r), \mathsf{KEM.Dec}(a, d), Y^x))$:
    send $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{C}, A, B, (Y, d))$ to $\mathcal{F}$, if $\mathcal{F}$ returns $k^*$ reset $k \leftarrow k^*$
if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, 2, y, d, r \rangle$ and $(b, B) \in KL_{\mathsf{S}}$ s.t.
$(Y, \sigma) = (g^y, (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Enc}_2(A, r), X^y))$:
    send $(\mathsf{SessionKey}, \mathsf{sid}, \mathsf{S}, B, A, (X, c))$ to $\mathcal{F}$, if $\mathcal{F}$ returns $k^*$ reset $k \leftarrow k^*$
add $\langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$
```

Fig. 21. Simulator SIM showing that SKEME realizes $\mathcal{F}_{\mathsf{khAKE}}$ (abbreviated "$\mathcal{F}$")

Suppose that $e, r_{\mathsf{P}}^{\mathsf{sid}} \leftarrow \mathsf{KEM.Enc}_1(\kappa), M \leftarrow \mathsf{KEM.Enc}_2(pk_{\mathsf{CP}}, r_{\mathsf{P}}^{\mathsf{sid}})$ and $f, r_{\mathsf{CP}}^{\mathsf{sid}} \leftarrow \mathsf{KEM.Enc}_1(\kappa), N \leftarrow \mathsf{KEM.Enc}_2(pk_{\mathsf{P}}, r_{\mathsf{CP}}^{\mathsf{sid}})$ are generated by $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ correspondingly. We use $r_{\mathsf{P}}^{\mathsf{sid}}$,$e_{\mathsf{P}}^{\mathsf{sid}}$ and $M_{\mathsf{P}}^{\mathsf{sid}}$ to represent r, e, M locally generated by $\mathsf{P}^{\mathsf{sid}}$ under some $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$.
Let KL be the list of all key pairs generated so far, and KL_{P} be the set of key pairs generated for P, $KL^{+}(\mathsf{P}^{\mathsf{sid}})$ stands for $KL \cup \{(sk_{\mathsf{CP}}, pk_{\mathsf{CP}})\}$ where pk_{CP} is the counterparty public key used by $\mathsf{P}^{\mathsf{sid}}$ and sk_{CP} is corresponding sk which doesn't necessarily need to be known or verified. (If $(sk_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL$ then $KL^{+}(\mathsf{P}^{\mathsf{sid}}) = KL$). Using these notions, we define following functions for every $\mathsf{P}^{\mathsf{sid}}$:

$$\begin{aligned}
\mathsf{SKEME}_{\mathsf{C}}^{\mathsf{sid}}(pk, pk', Y, d) = {} & (\mathsf{KEM.Enc}_2(pk', r), \mathsf{KEM.Dec}(sk, d), Y^x) \text{ for} \\
r = r_{\mathsf{C}}^{\mathsf{sid}}, x = x_{\mathsf{C}}^{\mathsf{sid}}, {} & (sk, pk) \in KL_{\mathsf{C}}, (\cdot, pk') \in KL^{+}(\mathsf{C}^{\mathsf{sid}}) \\
\mathsf{SKEME}_{\mathsf{S}}^{\mathsf{sid}}(pk, pk', X, c) = {} & (\mathsf{KEM.Dec}(sk, c), \mathsf{KEM.Enc}_2(pk', r), X^y) \text{ for} \\
r = r_{\mathsf{S}}^{\mathsf{sid}}, y = y_{\mathsf{S}}^{\mathsf{sid}}, {} & (sk, pk) \in KL_{\mathsf{S}}, (\cdot, pk') \in KL^{+}(\mathsf{S}^{\mathsf{sid}})
\end{aligned}$$

$$R^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', (Y, d)) = \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, pk, pk', X^{\mathsf{sid}}_{\mathsf{C}}, c^{\mathsf{sid}}_{\mathsf{C}}, Y, d, \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{C}}(pk, pk', Y, d))$$
$$R^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', (X, c)) = \mathsf{H}(\mathsf{sid}, \mathsf{C}, \mathsf{S}, pk, pk', X, c, Y^{\mathsf{sid}}_{\mathsf{S}}, d^{\mathsf{sid}}_{\mathsf{S}}, \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{S}}(pk, pk', X, c))$$

```
*Initialization:* Initialize an empty list $KL_{\mathsf{P}}$ for each $\mathsf{P}$

On message $\mathsf{Init}$ to $\mathsf{P}$:
pick $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $pk \leftarrow g^{sk}$, add $(sk, pk)$ to $KL_{\mathsf{P}}$, and output $(\mathsf{Init}, pk)$

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ then output $sk$

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathsf{P}$:
if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
if $\exists\ (sk_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$:
    pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $(e, r) \leftarrow \mathsf{KEM.Enc}_1(\kappa)$ and $M \leftarrow \mathsf{KEM.Enc}_2(pk_{\mathsf{CP}}, r)$
    write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, sk_{\mathsf{P}}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, M \rangle$ and output $(W = g^w, e)$

On message $(Z, f)$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, sk_{\mathsf{P}}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, M \rangle$:
    set $N \leftarrow \mathsf{KEM.Dec}(sk_{\mathsf{P}}, f)$ and $\sigma \leftarrow (M, N, Z^w)$
    set $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, g^w, e, Z, f, \sigma\}_{\mathrm{ord}})$ and output $(\mathsf{sid}, \mathsf{P}, k)$

On $\mathsf{H}$ query $(\mathsf{st}, A, B, X, c, Y, d, \sigma)$ for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$:
if $\exists\ \langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$
else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$, add $\langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ to $\mathsf{T_H}$, and output $k$
```

Fig. 22. SKEME: Environment's view of real-world interaction (Game 0)

Game 0 *(real world)*: This is the real-world interaction of environment $\mathcal{Z}$ (and its subroutine $\mathcal{A}$) with the SKEME protocol, shown in Fig. 22.

Game 1 *(past* H *queries are irrelevent to new sessions)*: We add an abort if session $\mathsf{P}^{\mathsf{sid}}$ starts with W which appeared in some prior inputs to H. As in the case of 3DH, $|\Pr[\mathsf{G1}] - \Pr[\mathsf{G0}]| \leq (2q_{\mathsf{H}})/p$.

Game 2 *(programming* $R^{\mathsf{sid}}_{\mathsf{P}}$ *values into* H *outputs)*: Define sessions $\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}}$ to be *matching* if $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{C}} = \mathsf{S}$ and $\mathsf{CP}^{\mathsf{sid}}_{\mathsf{S}} = \mathsf{C}$. By correctness of SKEME for any matching sessions and any public keys A, B it holds that $R^{\mathsf{sid}}_{\mathsf{C}}(A, B, (Y_{\mathsf{S}}, d_{\mathsf{S}})) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, (X_{\mathsf{C}}, c_{\mathsf{C}}))$. In Game 2 we set H outputs using functions $R^{\mathsf{sid}}_{\mathsf{P}}$. For every pair of matching sessions $(\mathsf{C}^{\mathsf{sid}}, \mathsf{S}^{\mathsf{sid}})$ we consider a pair of random functions $R^{\mathsf{sid}}_{\mathsf{C}}, R^{\mathsf{sid}}_{\mathsf{S}} : (\{0,1\}^*)^3 \to \{0,1\}^{\kappa}$ s.t. for all A, B

$$R^{\mathsf{sid}}_{\mathsf{C}}(A, B, (Y^{\mathsf{sid}}_{\mathsf{S}}, d^{\mathsf{sid}}_{\mathsf{S}})) = R^{\mathsf{sid}}_{\mathsf{S}}(B, A, (X^{\mathsf{sid}}_{\mathsf{C}}, c^{\mathsf{sid}}_{\mathsf{C}})) \tag{5}$$

Since they're matching sessions, above equation satisfy $c^{\mathsf{sid}}_{\mathsf{S}} = c^{\mathsf{sid}}_{\mathsf{C}}, d^{\mathsf{sid}}_{\mathsf{C}} = d^{\mathsf{sid}}_{\mathsf{S}}$. More precisely, for any session $\mathsf{P}^{\mathsf{sid}}$ with no matching session $R^{\mathsf{sid}}_{\mathsf{P}}$ is set as a random function, and for $\mathsf{P}^{\mathsf{sid}}$ for which a matching session exists $R^{\mathsf{sid}}_{\mathsf{P}}$ is set as a random function subject to constraint (5). Consider an oracle H which responds to each new query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ as follows:

1. If $\exists \mathsf{C}^{\mathsf{sid}}$ s.t. $(\mathsf{S}, X) = (\mathsf{CP}_{\mathsf{C}}^{\mathsf{sid}}, X_{\mathsf{C}}^{\mathsf{sid}})$, and $\exists\ A, B$ s.t. $(\cdot, A) \in KL_{\mathsf{C}}, (\cdot, B) \in KL^{+}(\mathsf{C}^{\mathsf{sid}})$ and $\sigma = \mathsf{SKEME}_{\mathsf{C}}^{\mathsf{sid}}(A, B, Y, d)$: Set $k \leftarrow R_{\mathsf{C}}^{\mathsf{sid}}(A, B, (Y, d))$
2. If $\exists \mathsf{S}^{\mathsf{sid}}$ s.t. $(\mathsf{C}, Y) = (\mathsf{CP}_{\mathsf{S}}^{\mathsf{sid}}, Y_{\mathsf{S}}^{\mathsf{sid}})$, and $\exists\ B, A$ s.t. $(\cdot, B) \in KL_{\mathsf{S}}$, $(\cdot, A) \in KL^{+}(\mathsf{S}^{\mathsf{sid}})$ and $\sigma = \mathsf{SKEME}_{\mathsf{S}}^{\mathsf{sid}}(B, A, X, c)$: Set $k \leftarrow R_{\mathsf{S}}^{\mathsf{sid}}(B, A, (X, c))$
3. In any other case pick $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$

Since the game knows each key pair $(sk_{\mathsf{P}}, pk_{\mathsf{P}})$ generated for each P, randomness r used in $\mathsf{KEM.Enc}$, and the ephemeral state w of each session $\mathsf{P}^{\mathsf{sid}}$, it can decide for any Z, f, pk' whether $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk', Z, f)$ if pk' is honestly generated, i.e. $(sk', pk') \in KL$. Moreover, even if pk' is adversarially generated, i.e. $(sk', pk') \in KL^{+}(\mathsf{P}^{\mathsf{sid}}) \setminus KL$, the game can still decide, since it records r and can generate M via $\mathsf{KEM.Enc}_2(pk', r)$, it can instead check whether the corersponding part of σ equals to M. Note that each value of $R_{\mathsf{P}}^{\mathsf{sid}}$ is used to program H on at most one query. Also, if H query $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ satisfies both conditions then $(X, Y) = (g^x, g^y)$ and $(c, d) = (c_{\mathsf{C}}^{\mathsf{sid}}, d_{\mathsf{S}}^{\mathsf{sid}})$, and $\exists\ A', B', a, b$ s.t.

$$\mathsf{SKEME}_{\mathsf{C}}^{\mathsf{sid}}(g^a, B', Y, d) = (\mathsf{KEM.Enc}_2(B', r_{\mathsf{C}}^{\mathsf{sid}}), \mathsf{KEM.Dec}(a, d), Y^x)$$

$$= (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Enc}_2(A', r_{\mathsf{S}}^{\mathsf{sid}}), X^y) = \mathsf{SKEME}_{\mathsf{S}}^{\mathsf{sid}}(g^b, A', X, c)$$

Since by security of $\mathsf{KEM.Enc}_2$ the above equations imply that $(A', B') = (A, B)$(e.g.$\mathsf{KEM.Dec}(b, c) = \mathsf{KEM.Enc}_2(B, r_{\mathsf{C}}^{\mathsf{sid}}) = \mathsf{KEM.Enc}_2(B', r_{\mathsf{C}}^{\mathsf{sid}})$), and by (5) we already know that $R_{\mathsf{C}}^{\mathsf{sid}}(A, B, (Y_{\mathsf{S}}^{\mathsf{sid}}, d_{\mathsf{S}}^{\mathsf{sid}})) = R_{\mathsf{S}}^{\mathsf{sid}}(B, A, (X_{\mathsf{C}}^{\mathsf{sid}}, c_{\mathsf{C}}^{\mathsf{sid}}))$, it follows that if both conditions are satisfied then both program H output to the same value. Thus we conclude:

$$\Pr[\mathsf{G2}] = \Pr[\mathsf{G1}]$$

Game 3 *(direct programming of session keys using random functions $R_{\mathsf{P}}^{\mathsf{sid}}$)*: As in 3DH, in Game 3 we make the following changes: We mark each initialized session $\mathsf{P}^{\mathsf{sid}}$ as fresh, and when $\mathcal{A}$ sends (Z, f) to $\mathsf{P}^{\mathsf{sid}}$ then it re-labels $\mathsf{P}^{\mathsf{sid}}$ as $\mathsf{interfered}$ unless (Z, f) equals to the message sent by the intended counterparty of P. In other words, $\mathsf{C}^{\mathsf{sid}}$ is re-labeled $\mathsf{interfered}$ if $Z_{\mathsf{C}}^{\mathsf{sid}} \neq Y_{\mathsf{S}}^{\mathsf{sid}}$ or $f_{\mathsf{C}}^{\mathsf{sid}} \neq d_{\mathsf{S}}^{\mathsf{sid}}$ and $\mathsf{S}^{\mathsf{sid}}$ is re-labeled $\mathsf{interfered}$ if $Z_{\mathsf{S}}^{\mathsf{sid}} \neq X_{\mathsf{C}}^{\mathsf{sid}}$ or $f_{\mathsf{S}}^{\mathsf{sid}} \neq c_{\mathsf{C}}^{\mathsf{sid}}$. Secondly, we say session $\mathsf{P}^{\mathsf{sid}}$ runs "under keys $(pk_{\mathsf{P}}, pk_{\mathsf{CP}})$" if it runs on its own key pair $(sk_{\mathsf{P}}, pk_{\mathsf{P}})$ and intended counterparty public key pk_{CP}. Using this notation Game 3 computes $k_{\mathsf{P}}^{\mathsf{sid}}$ as follows:

1. If $\mathsf{P}^{\mathsf{sid}}$ and $\mathsf{CP}^{\mathsf{sid}}$ are *matching*, both are fresh, $\mathsf{CP}^{\mathsf{sid}}$ runs under $(pk_{\mathsf{CP}}, pk_{\mathsf{P}})$, and $k_{\mathsf{CP}}^{\mathsf{sid}} \neq \perp$, then $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow k_{\mathsf{CP}}^{\mathsf{sid}}$
2. In all other cases set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$

We argue that this change makes no difference to the environment. In Game 2 value $k_{\mathsf{P}}^{\mathsf{sid}}$ is derived from $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, W, e_{\mathsf{P}}^{\mathsf{sid}}, Z, f\}_{\mathrm{ord}}, \sigma)$, where

$\sigma = \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$. However, H is programmed in Game 2 to output $R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ if $\exists\ (Z, f)$ s.t. $\sigma = \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$ for any $(\cdot, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. Since pk_{CP} used by $\mathsf{P}^{\mathsf{sid}}$ must be in set $KL^{+}(\mathsf{P}^{\mathsf{sid}})$, setting $k^{\mathsf{sid}}_{\mathsf{P}}$ directly as $R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ only short-circuits this process. Moreover, since $R^{\mathsf{sid}}_{\mathsf{C}}$ and $R^{\mathsf{sid}}_{\mathsf{S}}$ are correlated by equation (5), setting $k^{\mathsf{sid}}_{\mathsf{C}}$ as $k^{\mathsf{sid}}_{\mathsf{S}}$ or vice versa, in the case both are fresh, does not change the game. Thus we conclude:

$$\Pr[\mathsf{G3}] = \Pr[\mathsf{G2}]$$

Game 4 *(abort on* H *queries for passive sessions)*: We add an abort if oracle H triggers evaluation of $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', (Z, f))$ for any pk, pk' and $(Z, f) = (W^{\mathsf{sid}}_{\mathsf{CP}}, e^{\mathsf{sid}}_{\mathsf{CP}})$ where $\mathsf{CP}^{\mathsf{sid}}$ is a matching session of $\mathsf{P}^{\mathsf{sid}}$. Note that if $\mathsf{P}^{\mathsf{sid}}$ is passively observed, then value $(W^{\mathsf{sid}}_{\mathsf{CP}}, e^{\mathsf{sid}}_{\mathsf{CP}})$ either has been delivered to $\mathsf{P}^{\mathsf{sid}}$, i.e. $(Z^{\mathsf{sid}}_{\mathsf{P}}, f^{\mathsf{sid}}_{\mathsf{P}}) = (W^{\mathsf{sid}}_{\mathsf{CP}}, e^{\mathsf{sid}}_{\mathsf{CP}})$, or $\mathsf{P}^{\mathsf{sid}}$ is still waiting for message Z. By the code of oracle H in Game 2 the call to $R^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', (W^{\mathsf{sid}}_{\mathsf{CP}}, e^{\mathsf{sid}}_{\mathsf{CP}}))$ is triggered only if H query $(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, W, e, Z, f, \sigma\}_{\mathrm{ord}})$ satisfies the following for $Z = W^{\mathsf{sid}}_{\mathsf{CP}}$ and $f = e^{\mathsf{sid}}_{\mathsf{CP}}$:

$$\sigma \ = \ \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{P}}(pk, pk', Z, f)$$

As in proof of 3DH, the hardness of computing σ relies on hardness of computing $\mathsf{cdh}_g(W, Z)$, which is the last element in σ. We show that solving Gap CDH can be reduced to causing event Bad, defined as event that such query happens. The reduction $\mathcal{R}$ takes a CDH challenge $(\bar{X}, \bar{Y})$ and embeds it in a message of all simulated parties in a randomized way: on $\mathsf{NewSession}$ to P, if $\mathsf{role} = 1$ then $\mathcal{R}$ sends $X = \bar{X}^s$ as the message from $\mathsf{C}^{\mathsf{sid}}$ for $s \leftarrow \mathbb{Z}_p$, and if $\mathsf{role} = 2$ then $\mathcal{R}$ sends $Y = \bar{Y}^t$ as the message from $\mathsf{S}^{\mathsf{sid}}$ for $t \leftarrow \mathbb{Z}_p$. Otherwise it emulates the security game, in particular it knows all of the key pairs (a, A) and (b, B). Let KL be the list of all key pairs generated so far, and KL_{P} be the set of key pairs generated for P.
Although $\mathcal{R}$ doesn't know $x = s \cdot \bar{x}$ and $y = t \cdot \bar{y}$, where $\bar{x} = \mathsf{dlog}_g(\bar{X})$ and $\bar{y} = \mathsf{dlog}_g(\bar{Y})$, $\mathcal{R}$ can use DDH oracle to emulate the way Game 3 services H queries: To test if H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ for $X = \bar{X}^s$ satisfies $\sigma = \mathsf{SKEME}^{\mathsf{sid}}_{\mathsf{C}}(A, B, Y, d)$, because $\mathcal{R}$ knows all $(sk, pk) \in KL$ and records locally generated r, it can check first two part of $\sigma = (K, L, V)$.Then to test if $V = Y^x$, reduction $\mathcal{R}$ checks if $V = \mathsf{cdh}_g(\bar{X}, Y^s)$. Symmetrically on server side, to test if H input $(\mathsf{sid}, \mathsf{C}, \mathsf{S}, A, B, X, c, Y, d, \sigma)$ for $Y = \bar{Y}^t$ satisfies $V = X^y$, reduction $\mathcal{R}$ checks if $V = \mathsf{cdh}_g(X^t, \bar{Y})$. Since $\mathcal{R}$ emulates Game 3 perfectly, event Bad occurs with the same probability as in Game 3, and if it does $\mathcal{R}$ detects it because it occurs if both above conditions hold, in which case $\mathcal{R}$ outputs $V^{1/(st)}$ as $\mathsf{cdh}_g(\bar{X}, \bar{Y})$. It follows that $\Pr[\mathsf{Bad}] \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$, thus we conclude:

$$|\Pr[\mathsf{G4}] - \Pr[\mathsf{G3}]| \leq \epsilon^{\mathcal{Z}}_{\text{g-cdh}}$$

Game 5 *(random keys on passively observed sessions)*: Same as in 3DH, if session $\mathsf{S}^{\mathsf{sid}}$ remains fresh when $\mathcal{A}$ sends (Z, f) to $\mathsf{P}^{\mathsf{sid}}$ then instead of setting

$k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ as in Game 3, we now set $k_{\mathsf{P}}^{\mathsf{sid}} \leftarrow \{0,1\}^{\kappa}$. Since by Game 4 oracle H never queries $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ on (Z, f) sent from $\mathsf{P}^{\mathsf{sid}}$'s counterparty, which is a condition for $\mathsf{P}^{\mathsf{sid}}$ to remain fresh, it follows by randomness of $R_{\mathsf{P}}^{\mathsf{sid}}$ that the modified game remains externally identical, hence:

$$\Pr[\mathsf{G5}] = \Pr[\mathsf{G4}]$$

Game 6 *(decorrelating function pairs $R_{\mathsf{C}}^{\mathsf{sid}}, R_{\mathsf{S}}^{\mathsf{sid}}$)*: This game is same as in 3DH, and

$$\Pr[\mathsf{G6}] = \Pr[\mathsf{G5}]$$

Game 7 *(hash computation consistent only for compromised keys)*: Recall that in Game 6, as in Game 2, $\mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \cdot, \cdot, W_{\mathsf{P}}^{\mathsf{sid}}, Z, \sigma\}_{\mathrm{ord}})$ is defined as $R_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ if $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$ for some $(sk_{\mathsf{P}}, pk_{\mathsf{P}}) \in KL_{\mathsf{P}}$,$(sk_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. In Game 7 we add a condition that this programming of H can occur only if (1)$(sk_{\mathsf{CP}}, pk_{\mathsf{CP}})$ is honestly generated and $\mathsf{compromised}$ or (2)$(sk_{\mathsf{CP}}, pk_{\mathsf{CP}})$ is adversarially generated and pk_{CP} is the counterparty key $\mathsf{P}^{\mathsf{sid}}$ runs under. In both cases adversary can know sk_{CP}.
Let $CPKL$ be the list of generated key pairs that were compromised so far, Game 7 diverges from Game 6 if bad event occurs where H is queried on inputs as above for $\sigma = \mathsf{SKEME}_{\mathsf{P}}^{\mathsf{sid}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, Z, f)$ and $(sk_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL \setminus CPKL$, i.e. honestly generated but compromised key pairs, while in Game 6, as in Game 2, this programming was done whenever $(sk_{\mathsf{CP}}, pk_{\mathsf{CP}}) \in KL^{+}(\mathsf{P}^{\mathsf{sid}})$. As in the case of 3DH proof we only argue for client side since the other case is symmetric. We define event Bad, where in client side's H query value $\sigma = (\mathsf{KEM.Enc}_2(B, r), \mathsf{KEM.Dec}(a, d), Y^{x})$ for some $(a, A) \in KL_{\mathsf{C}}$ and $(b, B) \in KL \setminus CPKL$ which is fresh.
We show a reduction $\mathcal{R}$ that breaks $OWPCA$ security if Bad occurs. On input a $OWPCA$ challenge $(\bar{B}, c^{*})$, $\mathcal{R}$ has access to $PCO_{sk}(.,.)$ whose inner sk corresponds to $\bar{B}$, and $\mathcal{R}$ doesn't know randomness r used to generate c^{*}. $\mathcal{R}$ sets each $X_{\mathsf{C}}^{\mathsf{sid}}$ as g^{x} for random x and each $Y_{\mathsf{S}}^{\mathsf{sid}}$ as g^{y} for random y. $\mathcal{R}$ also picks all key pairs except that in the i-th session, for a chosen index $j \in [1, \ldots, q_{\mathrm{K}}]$, where $\mathcal{R}$ set the j-th public key pk_{CP} as $\bar{B}$, and sets c as c^{*}. Let $\mathsf{Bad}_{i,j}$ denote Bad occurring for this j-th public key in the i-th session, i.e. $pk_{\mathsf{CP}} = \bar{B}$.
As long as the corresponding sk_{CP} is not compromised, $\mathcal{R}$ can emulate Game 6 because it can respond to compromise of all other keys, and serve H queries as follows: To test server side H query input $(\mathsf{sid}, \mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma)$, i.e. if $\sigma = (K, L, V)$, $\mathcal{R}$ tests as in Game 6 except for b that corrsponds to the public key $\bar{B}$, in which case $\mathcal{R}$ tests if $K = \mathsf{KEM.Enc}_2(pk, r) = \mathsf{KEM.Dec}(sk, c)$ via checking if $PCO_{sk}(K, c)$ returns 1. To test client side, i.e. if $\sigma = (K, L, V)$ for any pk including $pk = \bar{B}$, $\mathcal{R}$ also tests as in Game 6, except for the case that sk is the private key corresponding to the public key $\bar{B}$, in which case $\mathcal{R}$ replaces testing $K = \mathsf{KEM.Enc}_2(pk, r)$ with checking if $PCO_{sk}(K, c)$ returns 1.

$\mathsf{Bad}_{i,j}$ can happen only before $(sk_{\mathsf{CP}}, pk_{\mathsf{CP}})$ used in that session is compromised, so it occurs in reduction with same probability as in Game 6. $\mathcal{R}$

can detect event $\mathsf{Bad}_{i,j}$ because it occurs if H query involves the j-th credential and on client side, given c and $\bar{B}$, it can output correct K that satisfies $K = \mathsf{KEM.Enc}_2(\bar{B}, r) = \mathsf{KEM.Dec}(b, c)$, without knowing the value of r and b, in which case it outputs correct K corresponding to c^* and breaks $OWPCA$ security. If $\mathcal{R}$ picks index i and j at random it follows that $\Pr[\mathsf{Bad}] \leq q_{\mathrm{K}} \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{ow-pca}_{\mathsf{KEM},A}$.
Since a symmetric argument holds also for server side, we conclude:

$$|\Pr[\mathsf{G7}] - \Pr[\mathsf{G6}]| \leq (2q_{\mathrm{K}}) \cdot q_{\mathrm{ses}} \cdot \mathbf{Adv}^{ow-pca}_{\mathsf{KEM},A}$$

Observe that Game 7 is identical to the ideal-world game shown in Figure 22: By Game 6 all functions $R^{\mathsf{sid}}_{\mathsf{P}}$ are random, by Game 5 the game responds to (Z, f) messages to $\mathsf{P}^{\mathsf{sid}}$ as the game in Figure 22, and after the modification in oracle H done in Game 7 this oracle also acts as in Figure 22. This completes the argument that the real-world and the ideal-world interactions are indistinguishable to the environment, and hence completes the proof of Theorem 3.

$\square$

```
*Initialization:* Initialize empty lists: $PK$, $CPK$, $KL$ and $KL_{\mathsf{P}}$ for all $\mathsf{P}$

On message $\mathsf{Init}$ to $\mathsf{P}$:
set $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $pk \leftarrow g^{sk}$, send $(\mathsf{Init}, pk)$ to $\mathsf{P}$, add $pk$ to $PK$ and $(sk, pk)$ to $KL$
and $KL_{\mathsf{P}}$

On message $(\mathsf{Compromise}, \mathsf{P}, pk)$:
If $\exists\ (sk, pk) \in KL_{\mathsf{P}}$ add $pk$ to $CPK$ and output $sk$

On message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}})$ to $\mathsf{P}$:
if $\exists\ (sk, pk_{\mathsf{P}}) \in KL_P$ then:
    initialize random function $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$
    if $\mathsf{P} <_{\mathsf{lex}} \mathsf{CP}$ then set $\mathsf{role} \leftarrow 1$ else set $\mathsf{role} \leftarrow 2$
    pick $w \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $(e, r) \leftarrow \mathsf{KEM.Enc}_1(\kappa)$
    write $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, r, \perp \rangle$ as $\mathsf{fresh}$, output $(W = g^w, e)$

On message $(Z, f)$ to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, r, \perp \rangle$:
    if $\exists$ record $\mathsf{rec}' = \langle \mathsf{sid}, \mathsf{CP}, \mathsf{P}, pk'_{\mathsf{CP}}, pk'_{\mathsf{P}}, \cdot, z, f', \cdot, k' \rangle$ s.t. $g^z = Z$ and $f = f'$
        then if $\mathsf{rec}'$ is $\mathsf{fresh}$, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$ and $k' \neq \perp$:
            then $k \leftarrow k'$
            else $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
        else set $k \leftarrow R^{\mathsf{sid}}_{\mathsf{P}}(pk_{\mathsf{P}}, pk_{\mathsf{CP}}, (Z, f))$ and re-label $\mathsf{rec}$ as $\mathsf{interfered}$
    update $\mathsf{rec}$ to $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, e, r, k \rangle$ and output $(\mathsf{NewKey}, \mathsf{sid}, k)$

On $\mathsf{H}$ query $(\mathsf{st}, A, B, X, c, Y, d, \sigma)$ for $\mathsf{st} = (\mathsf{sid}, \mathsf{C}, \mathsf{S})$:
if $\exists\ \langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ in $\mathsf{T_H}$ then output $k$, else pick $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$ and:
  1. if $\exists$ record $\langle \mathsf{sid}, \mathsf{C}, \mathsf{S}, \cdot, \cdot, 1, x, c, r, \cdot \rangle$ s.t.
     $(X, \sigma) = (g^x, (\mathsf{KEM.Enc}_2(B, r), \mathsf{KEM.Dec}(a, d), Y^x))$ for some $(a, A) \in KL_{\mathsf{C}}$
     and $B$ s.t. $B \in CPK$ or $B \notin PK$: reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{C}}(A, B, (Y, d))$
  2. if $\exists$ record $\langle \mathsf{sid}, \mathsf{S}, \mathsf{C}, \cdot, \cdot, 2, y, d, r, \cdot \rangle$ s.t.
     $(Y, \sigma) = (g^y, (\mathsf{KEM.Dec}(b, c), \mathsf{KEM.Enc}_2(A, r), X^y))$ for some $(b, B) \in KL_{\mathsf{S}}$
     and $A$ s.t. $A \in CPK$ or $A \notin PK$: reset $k \leftarrow R^{\mathsf{sid}}_{\mathsf{S}}(B, A, (X, c))$
add $\langle (\mathsf{st}, A, B, X, c, Y, d, \sigma), k \rangle$ to $\mathsf{T_H}$ and output $k$
```

Fig. 23. SKEME: Environment's view of ideal-world interaction (Game 7)