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 [49]. 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 [27]. For example, the overhead of the recently analyzed SPAKE2+ protocol [51] 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 [27, 33]. Known *strong* aPAKEs (see below), add similar or larger overhead costs [37, 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 [34, 29]. 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, 46].[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 Key-Hiding Asymmetric PakE) can be seen as a variant of the OPAQUE protocol [37] that is being developed into an Internet standard [42] and intended for use within TLS 1.3 [52]. 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 [35, 43, 30] and TBPEKE [48], 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 [44] and HMQV [41], and KEM-based protocols with key-hiding KEMs (e.g., SKEME [39]). 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 (see Section 8). 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 [37]. 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 [37], 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 [40] 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 [36] 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 [44] and HMQV [41] realize the key-hiding AKE functionality (in the ROM and under the Gap CDH assumption). We prove a similar result for SKEME [39] 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 [47] 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, KHAPE-HMQV, that instantiates KHAPE with HMQV as the key-hiding AKE. Finally, in Section 8 we survey potential instantiations of our ideal cipher encryption. In the full version of the paper [28], we include more background material as well as full proofs for all our theorems.

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 we show in the final version [28], 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 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

[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 $R^{\mathsf{sid}}_{\mathsf{P}} : (\{0,1\}^*)^3 \to \{0,1\}^\kappa$
- 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

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').

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 $R_{\mathsf{P}}^{\mathsf{sid}}$ 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 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 Interfere, NewKey, and SessionKey below.

Function $R_{\mathsf{P}}^{\mathsf{sid}}$. When command NewSession creates a session for $(\mathsf{sid}, \mathsf{P})$ the functionality initializes a *random* function $R_{\mathsf{P}}^{\mathsf{sid}}$ specific to this session. Function $R_{\mathsf{P}}^{\mathsf{sid}}$ 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 $R_{\mathsf{P}}^{\mathsf{sid}}$ on any point α via a 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 SessionKey actions below.

(Interfere, sid, P). This action represents an active attack on session $(\mathsf{P}, \mathsf{sid})$ and makes the session change its status from fresh to 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 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.

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 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 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

[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.

upon key generation and via the 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 $R^{\mathsf{sid}}_{\mathsf{P}}$ 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 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 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 [44] is a simple, implicitly authenticated key exchange used as the basis of the X3DH protocol [45] 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.

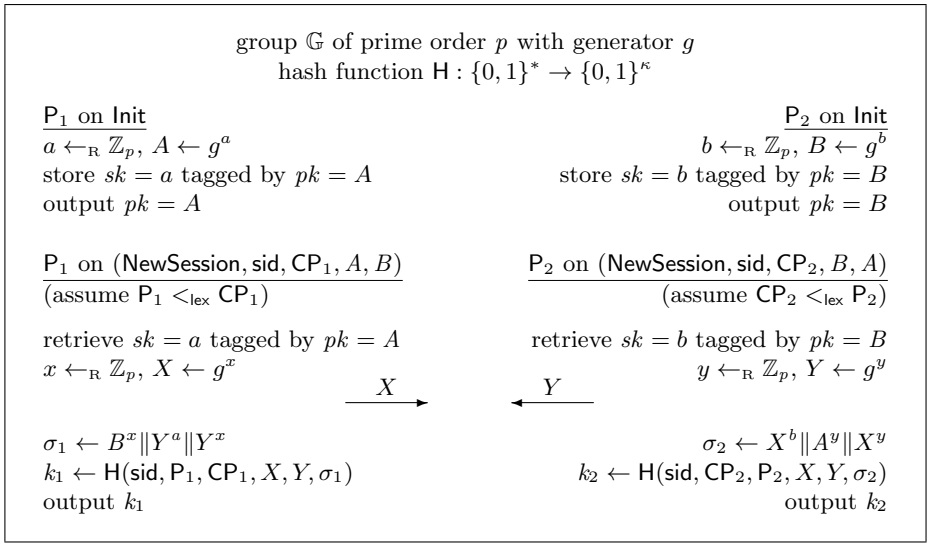

Fig. 2. Protocol $3DH$: "Triple Diffie-Hellman" Key Exchange

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_{P} for each 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_{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 H:
if $\exists\ \langle (\mathsf{sid}, \mathsf{C}, \mathsf{S}, X, Y, \sigma), k \rangle$ in 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 T_{H} and output k

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

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

Initialization: Initialize an empty list KL_{P} for each P

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

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. The full proof comprising the transitions between these games is presented in the full version [28].

4 HMQV as Key-Hiding AKE

We show that protocol HMQV [41], 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 [41] under the game-based AKE model of Canetti and Krawczyk [18], 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 [41] 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.

[8] However, we do not include adaptive session state compromise considered in [18, 41].

Initialization: Initialize empty lists: PK, CPK, and KL_{P} for all P

On message Init to P:
set $sk \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $pk \leftarrow g^{sk}$, send (Init, pk) to P, add pk to PK and (sk, pk) to KL_{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 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 $\langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \bot \rangle$ as fresh, output $W = g^w$

On message Z to session $\mathsf{P}^{\mathsf{sid}}$ (only first such message is processed):
if $\exists$ record $\mathsf{rec} = \langle \mathsf{sid}, \mathsf{P}, \mathsf{CP}, pk_{\mathsf{P}}, pk_{\mathsf{CP}}, \mathsf{role}, w, \bot \rangle$:
  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 rec' is fresh, $(pk_{\mathsf{P}}, pk_{\mathsf{CP}}) = (pk'_{\mathsf{P}}, pk'_{\mathsf{CP}})$, and $k' \neq \bot$:
      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 rec as $\mathsf{interfered}$
  update 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 P

On 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)

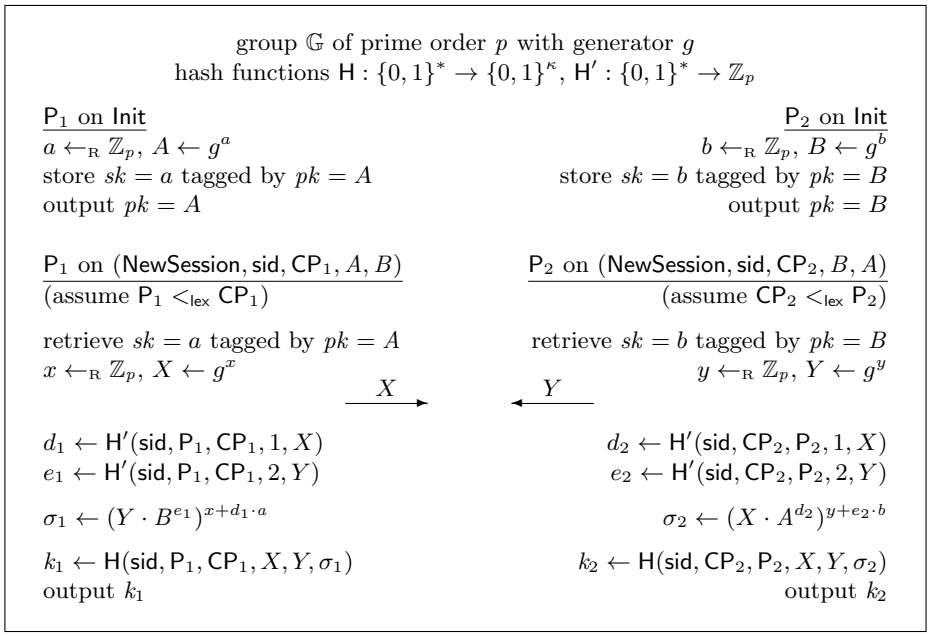

Fig. 6. Protocol HMQV [41]

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 present it in the full version [28].

5 SKEME as Key-Hiding AKE

We present a KEM-based instantiation of the SKEME protocol [39] in Figure 7. 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 $\{\mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma\}_{\text{ord}}$ 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 $k \leftarrow \mathsf{H}(\mathsf{sid}, \{\mathsf{P}, \mathsf{CP}, A, B, X, c, Y, d, \sigma\}_{\text{ord}})$.

The security of the protocol relies on two properties of the underlying KEM. First, we assume KEM to be One-Way under Plaintext-Checking-Attack, abbreviated as OW-PCA[31], where the attacker is given access to a Plaintext-Checking Oracle that on input a key K and

ciphertext c, it tells if c decapsulates to K under a given KEM key. Second, we require the KEM to be perfectly key-private, namely, given two pairs of private-public keys and a key encapsulation under one of them, one cannot distinguish (information-theoretically) which pair generated that ciphertext. Note the correspondence to the notion of key-hiding PKE [8]. See more details in the full version [28].

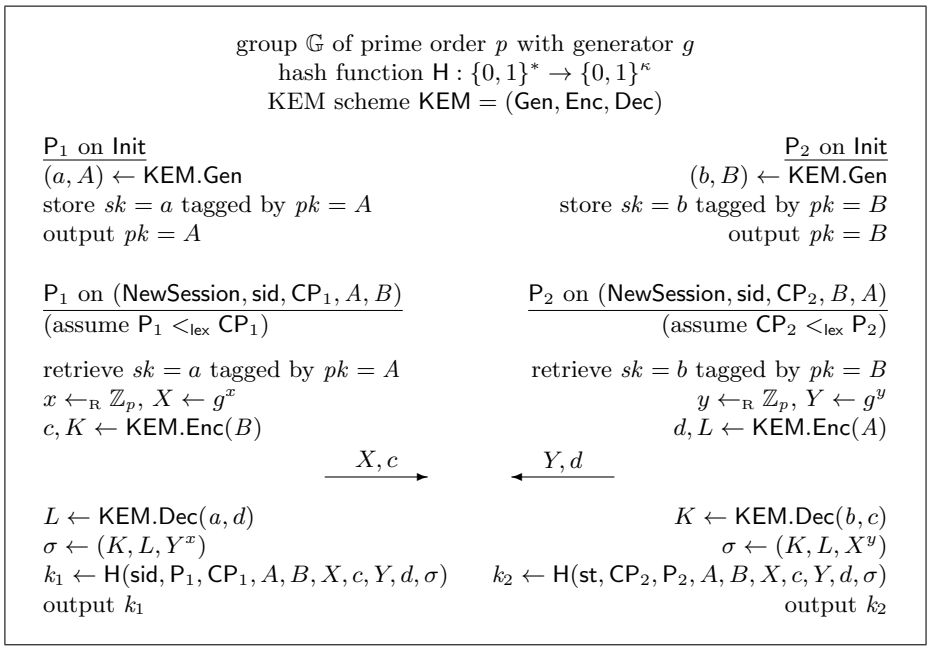

Fig. 7. Protocol SKEME: KEM-authenticated Key Exchange

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 present it in [28].

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.

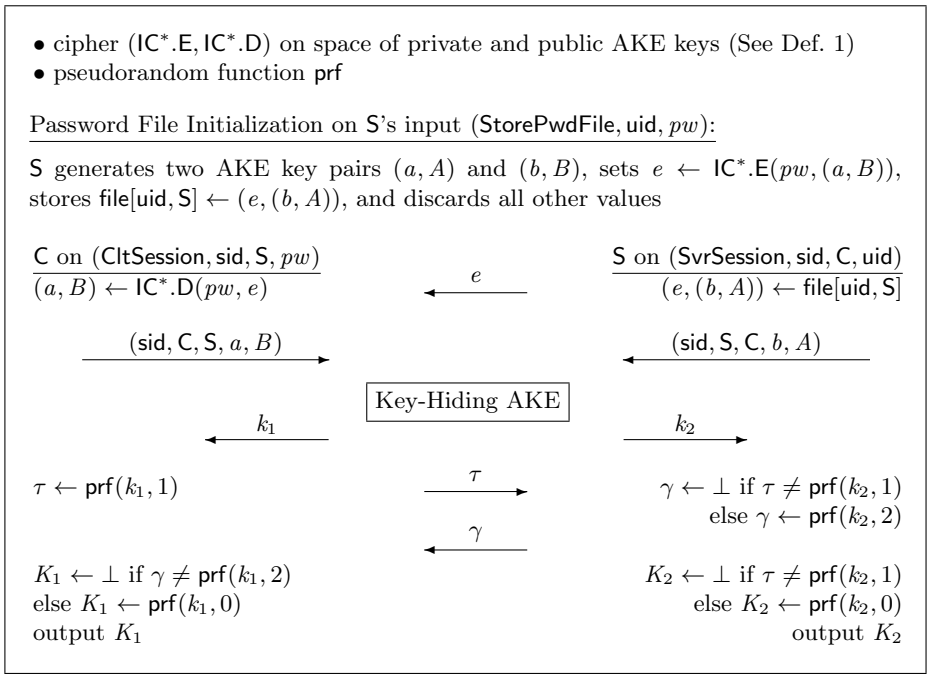

Fig. 8. Protocol KHAPE: Compiler from Key-Hiding AKE to aPAKE

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 $cred(pw)$ 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

[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.

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).

Definition 1. [$(\mathsf{IC}^*.\mathsf{E}, \mathsf{IC}^*.\mathsf{D})$ instantiation.] *Let X be the Cartesian product of the space of private keys and the space of public keys in AKE, let* $\mathsf{IC}.\mathsf{E}, \mathsf{IC}.\mathsf{D}$ *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 $x' \leftarrow_R \mathsf{map}(x)$ 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* $\mathsf{IC}.\mathsf{E}(x')$ *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* $x' = \mathsf{IC}.\mathsf{D}(y)$*.*

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. [46]). 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. The UC functionality $\mathcal{F}_{\mathsf{aPAKE}}$ with which we model aPAKE security corresponds to the functionality from Gentry et al. [27] with some slight modifications. 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. $\mathcal{F}_{\mathsf{aPAKE}}$ is described in the full version of the paper [28] where we also discuss 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.*

The proof of the theorem is presented in the full version of the paper [28].

7 Concrete aPAKE Instantiation: 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 9. It uses only 1 fixed-base exponentiation plus 1 variable-base

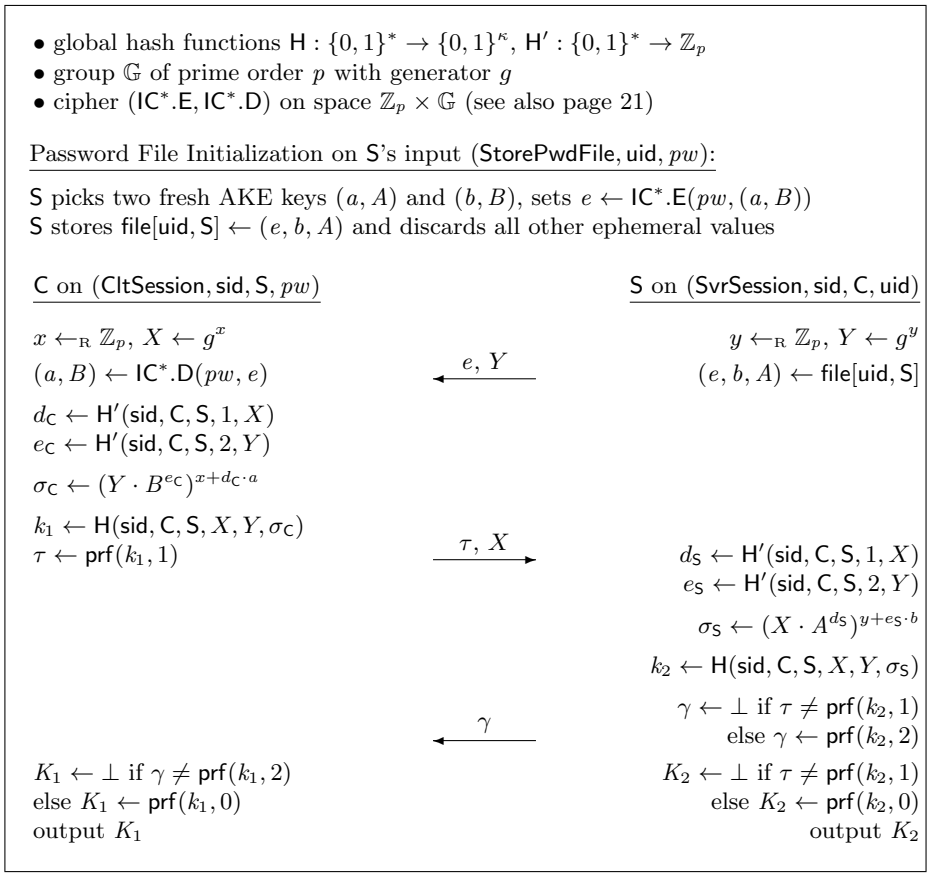

Fig. 9. KHAPE with HMQV: Concrete aPAKE protocol KHAPE-HMQV

(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 [37]. 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 2. *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. [50, 16, 26, 11, 53]). 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 [53, 38]. 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 [53] (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 2). Since u, v are field elements, a further bijection into bitstrings may be needed as specified in Section 8.1.

In [38], 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 ($\approx$10-15%) of a scalar multiplication. Implementing $g(v) = v \cdot P$ is also a low-cost option (also allowing to shorten v [38]). The cost of the inverse map, from a curve point to its bitstring encoding, for the curves analyzed in [53] 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 [25, 11, 54]. 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) [19], Feistel networks and constructions from random oracles [23, 32, 20], dedicated constructions such as those based on iterated Even-Mansour and key alternating ciphers [22, 6, 24, 24], and basic components such as wide-input (public) random permutations [13, 12, 21]. A recent technique by McQuoid et al. [46], 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 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.
19. 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.
20. 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.
21. J. Daemen, S. Hoffert, G. V. Assche, and R. V. Keer. The design of Xoodoo and Xoofff. 2018:1–38, 2018.
22. 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.
23. Y. Dai and J. P. Steinberger. Indifferentiability of 8-round Feistel networks. In *Advances in Cryptology – CRYPTO 2016*, pages 95–120, 2016.
24. 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.
25. 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.
26. P.-A. Fouque, A. Joux, and M. Tibouchi. Injective encodings to elliptic curves. In *Australasia Conference on Information Security and Privacy – ACISP 2013*, 2013.
27. 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.
28. Y. Gu, S. Jarecki, and H. Krawczyk. KHAPE: Asymmetric PAKE from Key-Hiding Key Exchange. *IACR Cryptology ePrint Archive*, June 2021. `http://eprint.iacr.org/2021`.
29. S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):230–268, 1999.
30. F. Hao and S. F. Shahandashti. The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585, 2014. `http://eprint.iacr.org/2014/585`.
31. 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.
32. T. Holenstein, R. Künzler, and S. Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *STOC 2011*, 2011.
33. 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.
34. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC'89*, pages 44–61, 1989.
35. 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.

36. 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.
37. 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.
38. T. Kim and M. Tibouchi. Invalid curve attacks in a GLS setting. In *International Workshop on Security (IWSEC 2015)*, 2015.
39. 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.
40. 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.
41. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.
42. 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.
43. 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`.
44. M. Marlinspike. Simplifying OTR deniability, `https://signal.org/blog/simplifying-otr-deniability/`, 2013.
45. M. Marlinspike and T. Perrin. The X3DH key agreement protocol, `https://signal.org/docs/specifications/x3dh/`, 2016.
46. 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`.
47. NIST Information Technology Lab. Post-quantum cryptography, `https://csrc.nist.gov/projects/post-quantum-cryptography`.
48. D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In *ASIACCS 17*, pages 301–312. ACM Press, 2017.
49. J. Schmidt. Requirements for password-authenticated key agreement (PAKE) schemes, `https://tools.ietf.org/html/rfc8125`, Apr. 2017.
50. A. Shallue and C. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANTS*, 2006.
51. V. Shoup. Security analysis of SPAKE2+. *IACR Cryptol. ePrint Arch.*, 2020:313, 2020.
52. 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.
53. 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.
54. 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`.