Strong Asymmetric PAKE based on Trapdoor CKEM

Tatiana Bradley, Stanislaw Jarecki, and Jiayu Xu
{tebradle,sjarecki,jiayux}@uci.edu

University of California, Irvine

Abstract. Password-Authenticated Key Exchange (PAKE) protocols allow two parties that share a password to establish a shared key in a way that is immune to offline attacks. Asymmetric PAKE (aPAKE) [21] adapts this notion to the common client-server setting, where the server stores a one-way hash of the password instead of the password itself, and server compromise allows the adversary to recover the password only via the (inevitable) offline dictionary attack. Most aPAKE protocols, however, allow an attacker to *pre*-compute a dictionary of hashed passwords, thus instantly learning the password on server compromise. Recently, Jarecki, Krawczyk, and Xu formalized a Universally Composable *strong* aPAKE (saPAKE) [24], which requires the password hash to be salted so that the dictionary attack can only start after the server compromise leaks the salt and the salted hash. The UC saPAKE protocol shown in [24], called OPAQUE, uses 3 protocol flows, 3-4 exponentiations per party, and relies on the One-More Diffie-Hellman assumption in ROM.

We propose an alternative UC saPAKE construction based on a novel use of the encryption+SPHF paradigm for UC PAKE design [27, 20]. Compared to OPAQUE, our protocol uses only 2 flows, has comparable costs, avoids hashing onto a group, and relies on different assumptions, namely Decisional Diffie-Hellman (DDH), Strong Diffie-Hellman (SDH), and an assumption that the Boneh-Boyen function $f_s(x) = g^{1/(s+x)}$ [9] is a *Salted Tight One-Way Function* (STOWF). We formalize a UC model for STOWF and analyze the Boneh-Boyen function as UC STOWF in the generic group model and ROM.

Our saPAKE protocol employs a new form of Conditional Key Encapsulation Mechanism (CKEM), a generalization of SPHF, which we call an *implicit-statement* CKEM. This strengthening of SPHF allows for a UC (sa)PAKE design where only the client commits to its password, and only the server performs an SPHF, compared to the standard UC PAKE design paradigm where the encrypt+SPHF subroutine is used symmetrically by both parties.

1 Introduction

Passwords are the most common form of authentication on the Internet, and the almost-universal password authentication method is password-over-TLS. In

this method, the user (client) sends their encrypted password over TLS to a server, who then decrypts the password and verifies it against a locally stored password file. The password file does not contain the correct password pw in cleartext, but rather a random salt s and a salted password hash $F_s(pw)$, which is a randomized one-way function of the password. Function F must be a (salted) *tight* one-way function [7] in the sense that an attacker who compromises the server and learns its password file $(s, F_s(pw))$, can find pw only by running an exhaustive offline dictionary attack, at cost which is linear in the number of tested password guesses.

However, there are least two major disadvantages of password-over-TLS: (1) the cleartext password is handled by the server during login, which could leak the password without an offline dictionary attack if the server is compromised; and (2) a Public Key Infrastructure (PKI) is needed to authenticate the server to the client, and the client loses all security if the TLS channel uses a compromised public key, e.g., due to a phishing attack (see also multiple other PKI attacks listed in [24]). Both problems of password-over-TLS are well known and have been partly addressed by the literature on Password Authentication Key Exchange (PAKE). Only one existing PAKE proposal, however, maintains password-over-TLS's advantage of forcing a full offline dictionary attack after server compromise, while mitigating its disadvantages of potential cleartext password leakage and reliance on PKI. We propose another, and to motivate our contribution we first discuss past work on PAKE.

Cryptographic PAKE and aPAKE protocols. Standard PAKE, introduced by Bellovin and Merritt in [4], formalized in a game-based indistinguishability approach by Bellare et al. [3], and in the Universally Composable (UC) framework by Canetti et al. [14], considers secure key exchange between two parties who share the same password. Because passwords have low entropy, PAKE protocols cannot prevent the active attacker from performing an *online* impersonation attack by guessing the password, but they do guarantee security against *offline* attacks, i.e., active attacks must be the only way to verify a guessed password. Crucially, PAKE requires only passwords as inputs, and does not rely on authentic distribution of public keys via PKI, thus solving problem (2) of the password-over-TLS method. However, the shared-password PAKE model of [3, 14] makes problem (1) even worse: to participate in the protocol, the server needs to *store*, not just briefly handle, the cleartext password.

To solve problem (1) and (2) together, Bellovin and Merritt [5] introduced the notion of *asymmetric* PAKE (aPAKE), a.k.a. *augmented* or *verifier-based*, which allows the server party to execute the protocol on a one-way function of the password instead of the password itself. This notion was formalized in the simulation-based approach by Boyko et al. [11], and in the UC framework by Gentry et al. [21]. There are several aPAKE's proposed in the literature, both in the simulation-based model [11, 30, 29, 7] and in the UC model [21, 26, 22]. Several aPAKE protocols have also been proposed with only ad-hoc security arguments – see [24] for their discussion.

In all the above aPAKE protocols, however, passwords are hashed via some *deterministic* one-way function, making password files vulnerable to pre-computation attacks: An attacker who pre-computes a list of hashes for all passwords in an assumed dictionary can *instantly* find a user's password on server compromise. This attack is allowed by aPAKE models because, despite guaranteeing that the offline attack take work linear in the dictionary size, they do not require that this work be done before server compromise [11, 21].

All aPAKE protocols using a deterministic one-way function allow for the password hash to be randomized if the random salt assigned to a given user account is "public," i.e., it is revealed in the authentication protocol. This makes the pre-computation attack harder, because the adversary must engage in the online protocol to gather the (public) salt values assigned to user accounts, and pre-compute separate $(pw, F_s(pw))$ tables for each user-specific salt value s. Yet the core of the problem remains: the adversary can pre-compute hash values for a dictionary of the most probable passwords, thus learning the real password immediately on server compromise.

Strong aPAKE. To prevent pre-computation attacks, Jarecki, Krawczyk, and Xu have recently proposed a UC *strong* aPAKE (saPAKE) notion [24], which enforces that the offline dictionary attack takes $O(|D|)$ work *after* server compromise. Strong aPAKE thus bridges the security gap between password-over-TLS and PAKE, with no cleartext passwords on the server, no reliance on PKI, and no instant password retrieval on server compromise.

In the same paper [24] presented an efficient UC saPAKE protocol, called OPAQUE: It relies on standard prime-order groups, the client C and server S make respectively only 4 and 3 (multi-)exponentiations, and the protocol has 3 message flows.[1] The security of OPAQUE relies on ROM and an interactive hardness assumption of One-More Diffie-Hellman (OMDH), which states that an adversary who is allowed n queries to the Diffie-Hellman oracle $(\cdot)^k$ can compute the $(\cdot)^k$ function on at most n out of $n+1$ random challenge group elements.

Our saPAKE Contribution. We propose a new UC saPAKE construction family based on a novel application of the encryption+SPHF paradigm. Our saPAKE construction requires ROM, as OPAQUE [24] does[2], it has only 2 message flows, and it has comparable computational costs to OPAQUE, with 1 variable-base and 10 fixed-base (multi-)exponentiations for the client and 2 fixed-base and 2 variable-base (multi-)exponentiations for the server. Unlike OPAQUE, it does not require hashing onto a curve, which simplifes its implementation over some some curves. Our protocol is also based on different

[1] The conference paper [24] reported it as 2 message flows but the full version [25] explain why 3 flows seem necessary.

[2] ROM appears to be a minimal model necessary to achieve UC aPAKE [21], and thus UC saPAKE. To satisfy the UC (s)aPAKE notion, we need some idealized computation model (e.g., RO or a generic group) that allows us to "count" the number of times $F(x)$ is called in the adversary's *local computation*, and also to extract the effective inputs x on which the adversary computes F values.

hardness assumptions, namely DDH and Strong DH, and any assumptions neceessary for the (hashed) Boneh-Boyen function $F_s(x) = g^{1/(s+\mathsf{H}(x))}$ [9] to realize what we call a *Salted Tight One-Way* (STOWF) function. The protocol family we show can also be used with different STOWF function proposals, under different assumptions, possibly leading to further improvements in efficiency and weaker security assumptions for saPAKE's.

Our Approach: UC STOWF and implicit-statement CKEM. Our approach is inspired by the encryption+SPHF paradigm in UC PAKE design, which began with the work of [27, 20]. In this paradigm, both the client C and the server S commit to their passwords, respectively pw_C and pw_S, using an encryption scheme Enc whose public key pk is in the CRS. Party C uses the ciphertext c_S sent by S to derive a secret key k_C via a Smooth Projective Hash Function (SPHF) on input statement $x_\mathsf{C} = (pw_\mathsf{C}, c_\mathsf{S}, pk)$, where the SPHF operates on language $\mathcal{L}$ of statements (pw, c, pk) such that $c = \mathsf{Enc}_{pk}(pw)$. Recall that an SPHF allows the hashing party C to compute a *projection key* hp_C such that S can derive the secret key k_C given only the projection key hp_C and a witness for $x_\mathsf{C} \in \mathcal{L}$. The witness is the randomness r that S used to encrypt $c_\mathsf{S} = \mathsf{Enc}_{pk}(pw_\mathsf{S}; r)$, and r is a valid witness for the client's statement $(pw_\mathsf{C}, c_\mathsf{S}, pk) \in \mathcal{L}$ only if c_S also encrypts pw_C, i.e, $pw_\mathsf{C} = pw_\mathsf{S}$. The protocol is symmetrical, i.e. server S follows the same process with C's ciphertext to form k_S, and the final key is a combination of the two keys, e.g. $k_\mathsf{C} \oplus k_\mathsf{S}$.

It is not immediately clear how to apply this paradigm to the setting of saPAKE, because S cannot use the same SPHF as C, as it does not hold pw, but only the password file (s, z) for $z = F_s(pw)$. Jutla and Roy [26] solve this problem in the context of aPAKE (but not *strong* aPAKE) by using SPHFs for two different languages, one to verify the client's encryption of pw using a (deterministic) one-way function $F(pw)$ held by the server, and another to verify the server's encryption of $z = F(pw)$. In saPAKE, however, the client cannot compute $z = F_s(pw)$, as the random salt s must be private on the server, so this approach does not work in our setting.

Instead, we condense the encryption+SPHF paradigm so that a single SPHF authenticates both the client and the server. To do this, we start from a generalized SPHF protocol called a *Conditional Key Encapsulation Mechanism* (CKEM), introduced in [6] as an *Implicit Zero-Knowledge* protocol, and we strengthen its security properties so that the CKEM public message m (corresponding to the SPHF projection key) can be used as a one-time secret authenticator as follows: If S uses our CKEM to verify that C's ciphertext c_C encrypts pre-image pw of S's password file $z = F_s(pw_\mathsf{S})$, then S's CKEM message commits to its statement $x_\mathsf{S} = (pk, c_\mathsf{C}, F_s(pw_\mathsf{S}))$, and hence it can also act as S's authentication to C. This commitment cannot be publicly verifiable, since x_S reveals $F_s(pw_\mathsf{S})$, but the statement-committing and statement-privacy properties of our CKEM require that the commitment is verifiable only by a party (in our case, the client C) that holds a witness for $x_\mathsf{S} \in \mathcal{L}$. Furthermore, for the message m to work as an authenticator for S in the UC model we need to make the sender's statement extractable by the ideal-world simulator.

Moreover, standard CKEMs assume that both parties have the full language statements as inputs, but in our case the server's statement involves secret password file (s, z) not known to the client. For that reason we develop a CKEM variant we call an *Implicit Statement* CKEM, where the sender effectively encrypts the statement it assumes in the CKEM message, and the receiver decrypts it and verifies that it agrees with its witness.

Perhaps surprisingly, we can meet all these requirements by adding just one (multi-)exponentiation for the receiver to the cost of the standard SPHF for the same language $\mathcal{L}$, with the slight caveat that SPHF-to-strong-CKEM compiler assumes ROM. However, recall that realizing the UC (s)aPAKE functionality already seems to require an idealized model like ROM, hence it is natural to use ROM to reduce the costs of the underlying tools.

Another cost-reducing feature of our saPAKE construction is that the encryption scheme that C uses to commit to its password does not need to be non-malleable, but only indistinguishable. This weaker requirement allows us to use ElGamal encryption, which has half the cost of Cramer-Shoup encryption, and the SPHF for the language $\mathcal{L}$ of "encryptions of password file pre-image" is also correspondingly cheaper. This cost-saving is possible due to the strong properties of our CKEM: The CKEM for statement $x = (pk, c, z)$ has CCA-like properties, e.g., it is secure even to the adversary who has access to trapdoor-receiver oracle (which corresponds to a decryption oracle in CCA encryption) as long as the "trapdoor-decrypted" CKEM message is different from the challenge CKEM message. And indeed, the techniques we use to compile our (non-malleable) CKEM from (malleable) SPHF are akin to the Fujisaki-Okamoto transform [18] from IND PKE to CCA PKE.

On the use of idealized models. Our protocol saPAKE makes use of both the (non-programmable) Random Oracle Model (ROM) and the (programmable) Generic Group Model (GGM). However, GGM is isolated so that it is used only in the offline parts of the protocol, i.e., we use it to show that the adversary may only make offline queries after it steals a password file, and cannot perform any meaningful pre-computation. By contrast, we use ROM in both the offline and online parts of the protocol. As we have mentioned, it appears that *some* programmable idealized computational model like the Random Oracle Model, the Ideal Cipher Model, or the Generic Group Model, is necessary in any UC (s)aPAKE, because of the strong constraints which the UC (s)aPAKE model imposes on the local computation of the real-world adversary. Essentially, the simulator must provide the adversary with a password file before it knows the actual password, then detect offline password guesses, and, if a guess is correct, program the ideal model so that the password file does in fact correspond to the correct password. That said, it may be possible to construct a UC (s)aPAKE which uses ROM/IC/GGM only in the offline parts. (Indeed, VPAKE [7], which is a non-UC aPAKE, uses ROM only offline). We analyze the security of our saPAKE protocol candidate assuming GGM only in the offline part, and ROM in both the offline and the online part, where the latter choice is dictated by the concrete effiency of the resulting protocol.

Summary of Contributions. In summary, our contributions are as follows:

- We construct the first two round protocol that realizes the UC saPAKE functionality of [24]. Our protocol is based on public-key encryption (PKE), and two primitives which we introduce: implicit-statement trapdoor CKEM and salted tight one-way functions (STOWF).
- We introduce and realize the notion of an implicit-statement conditional key encapsulation mechanism (CKEM). Implicit-statement CKEM differs from standard CKEM in that the sender's is private to anyone who does not a witness for it, and the receiver uses only a witness, and not a statement, in order to compute its key.
- We formalize the notion of a UC salted tight one way function (STOWF) as a UC functionality $\mathcal{F}_{\mathsf{STOWF}}$, and we prove that the (hashed) Boneh-Boyen signature function $F_s(x) = g^{1/(s+\mathsf{H}(x))}$ realizes $\mathcal{F}_{\mathsf{STOWF}}$ in the programmable generic group model and ROM.

Table 1 compares our saPAKE protocol with previous work on asymmetric PAKE's in terms of efficiency, the notion of security achieved, and the required security assumptions.

	Security	Client	Server	Rounds	Assumptions
This work	UC saPAKE	$1v$+$10f$	$2v$+$2f$	2	DDH, 2-SDH, GGM[(1)]
OPAQUE[(2)] [25]	UC saPAKE	$2v$+$2f$	$2v$+$1f$	3	OMDH
Jutla-Roy [26]	UC aPAKE	$O(1)$ pairings[(3)]		1	SXDH, MDDH
HJKLSX[(4)] [22]	UC aPAKE	$2v$+$4f$	$1v$+$4f$	3	DDH
GMR[(4)] [21]	UC aPAKE	$1v$+$5f$	$2v$+$2f$	3	DDH
VPAKE [7]	GB aPAKE	$3v$+$8f$	$3v$+$4f$	2 or 1[(5)]	DDH, GGM[(1)]
PAK-X [11]	SIM aPAKE	$3v$+$1f$	$2v$+$1f$	3	DDH

Table 1. We compare several asymmetric PAKE protocol proposals regarding security claims, number of rounds, security assumptions, and client and server efficiency, where v and f are resp. variable-base (multi-)exponentiations and fixed-based exponentiations. All protocols rely on ROM, except [7] only in offline part. GB and SIM indicate resp. game-based and simulatability-based aPAKE security notions. (1) GGM used only in the offline part; (2) OPAQUE costs reflect "multiplicative blinding" OPRF optimization reported in [25]; (3) [26] also uses a significant number of exponentiations; (4) GMR and HJKLSX are compilers from UC PAKE to UC aPAKE, here instantiated with the two-round UC PAKE of [12] secure under DDH in ROM, with resp. $1v$+$4f$ and $1v$+$2f$ costs for client and server; In the Ideal Cipher (IC) the UC PAKE of [12] reduces these costs by resp. $3f$ and $1f$ for client and server; (5) The 1-flow version of aPAKE of [7] uses a few more mostly fixed-base exponentiations.

Paper Roadmap. In Section 2 we define the strong (simulation-sound and statement-private) notion of implicit-statement CKEM described above, and

show a generic construction of such CKEM from an SPHF for the same language in ROM. In Section 3 we propose the UC model for Salted Tight One-Way Functions (STOWF), and we show that the (hashed) Boneh-Boyen function $F_s(x) = g^{1/(s+\mathsf{H}(x))}$ realizes this UC functionality in GGM and ROM, and that the simulator's work is linear to the real-world adversary's work. In Section 4 we show our saPAKE protocol based on these STOWF and strong CKEM tools, and we show that it realizes the UC saPAKE functionality of [24] in ROM assuming that function f is a UC STOWF. Finally, in Section 5 we show a highly efficient 2-flow UC saPAKE obtained by instantiating the construction of Section 4 with the Boneh-Boyen function as the STOWF and the generic CKEM construction of Section 2 with the specific language used in our saPAKE construction.

2 Conditional Key Encapsulation Mechanisms

Notation. Throughout the paper, κ denotes the security parameter; "$\leftarrow$" denotes either a deterministic or a randomized assignment; and "$\leftarrow_{\mathrm{R}}$" denotes uniform sampling from a given set.

Basic CKEM. A *Conditional Key Encapsulation Mechanism (CKEM)* scheme [6, 17, 16, 1, 28, 8] implements a transfer of a random key between two parties, the sender and the receiver, under the condition that a given statement, known by both parties, belongs to some language. A CKEM can be implemented with a Smooth Projective Hash Function (SPHF), but it generalizes SPHF to interactive and only computationally secure protocols.

Let $\mathcal{L}$ be an NP language, a subset of implicit universe $\mathcal{U}$, and let $\mathcal{R}[\mathcal{L}]$ be an efficiently verifiable relation associated with $\mathcal{L}$. A CKEM for language $\mathcal{L}$ is a triple $(\mathsf{PG}, \mathsf{Snd}, \mathsf{Rec})$ where PG is a parameter generation algorithm that on input a security parameter κ outputs public parameters π, while Snd and Rec are interactive algorithms executed resp. by Sender and Receiver, where Snd runs on input parameters π, label ℓ identifying the protocol instance, and statement x, and Rec runs on inputs π, ℓ, x and a witness w, and both algorithms locally output a key ck. (To reduce visual clutter we will denote inputs π, ℓ as indices, e.g. $\mathsf{Snd}_{\pi,\ell}(x)$ will denote $\mathsf{Snd}(\pi, \ell, x)$, etc.) CKEM correctness requires that if the sender and the receiver hold the same statement in $\mathcal{L}$, and the receiver holds a witness for it, then they output the same key, i.e. for all κ, ℓ, all $\pi \leftarrow \mathsf{PG}(1^\kappa)$ and all $(x, w) \in \mathcal{R}[\mathcal{L}]$, if $(ck_{\mathsf{S}}, ck_{\mathsf{R}}) \leftarrow [\mathsf{Snd}_{\pi,\ell}(x), \mathsf{Rec}_{\pi,\ell}(x, w)]$ then $ck_{\mathsf{S}} = ck_{\mathsf{R}}$. The basic security property of CKEM is *soundness*, which states that if $x \notin \mathcal{L}$ then an efficient adversary interacting with $\mathsf{Snd}_{\pi,\ell}(x)$ cannot distinguish Snd's output key from a random string. In other words, if the sender generates its key on a false statement then this key is pseudorandom to the receiver.

Trapdoor CKEM. Benhamouda et al. [6] defined a *trapdoor* CKEM, called an *Implicit Zero-Knowledge* therein, which allows a simulator holding a global trapdoor to compute the sender's key even on false statements. This

simulatability property makes CKEM a stronger protocol building block because it allows the simulator to perform any subsequent actions an honest party would do using a key received via a CKEM on a statement proving honest behavior. (Recall that the simulator typically does not have a witness for such statement because it needs to simulate an honest party without knowing its private data.) A trapdoor CKEM scheme includes two additional algorithms, the trapdoor parameter generation procedure TPG, which outputs parameters π together with a *simulation trapdoor* td, and a trapdoor receiver algorithm TRec which satisfies the *zero-knowledge* property [6], which states that for any $(x, w) \in \mathcal{R}[\mathcal{L}]$, an interaction with $\mathsf{TRec}_{\pi,\ell}(x, td)$ (including its local output) is indistinguishable from an interaction with $\mathsf{Rec}_{\pi,\ell}(x, w)$. Moreover, in parallel to zero-knowledge proofs, a trapdoor CKEM should satisfy *simulation-soundness* [6], which states that for any $x \notin \mathcal{L}$ an adversary interacting with $\mathsf{Snd}_{\pi,\ell}(x)$ cannot distinguish Snd's output key from a random string, even given access to oracle $\mathsf{TRec}_{\pi,\ell'}(x', td)$ for any $(\ell', x') \neq (\ell, x)$.

Implicit-Statement CKEM. We introduce a new CKEM variant we call an *implicit-statement* CKEM, in which the receiver might not know the statement used by the sender, and has only a witness as its input. This makes a difference for languages where the same value can be a witness for many statements. Note that in the context of saPAKE application a language of pre-images of a salted hash function in an example of such language because password pw is a witness to correctness of password file $(s, f_s(pw))$ for every salt value s (see Section 3). Recall also that an saPAKE server must hide the salt value, to prevent pre-computation in a dictionary attack that can be staged after server compromise, so the statement assumed by the saPAKE server cannot be sent to the client in the clear. An implicit-statement CKEM allows the sender to embed its statement into the CKEM message so that the receiver reconstructs it together with the sender's key *only if* this statement is matched by the receiver's witness.

Since our implicit-statement CKEM construction is non-interactive we state all definitions below in this context, but they can be easily adapted to the interactive setting. Thus we define CKEM scheme as a tuple of non-interactive algorithms $(\mathsf{PG}, \mathsf{TPG}, \mathsf{Snd}, \mathsf{Rec}, \mathsf{TRec})$ where $\mathsf{PG}, \mathsf{TPG}$ are as above, $\mathsf{Snd}_{\pi,\ell}$ on input x outputs key ck and a message m, and $\mathsf{Rec}_{\pi,\ell}$ and $\mathsf{TRec}_{\pi,\ell}$, output (ck, x) on inputs resp. (w, m) and (td, m). The implicit-statement CKEM *correctness* requires that for all κ, ℓ and $(x, w) \in \mathcal{R}[\mathcal{L}]$ if $\pi \leftarrow \mathsf{PG}(1^\kappa)$, $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x)$, and $(ck', x') \leftarrow \mathsf{Rec}_{\pi,\ell}(w, m)$ then $(ck', x') = (ck, x)$. We require two additional syntactic properties of CKEM: *statement verification* for the receiver, which states that for all w, m, ℓ if $\pi \leftarrow \mathsf{PG}(1^\kappa)$ and $(ck, x) \leftarrow \mathsf{Rec}_{\pi,\ell}(w, m)$ then (a) if $x \neq \perp$ then $(x, w) \in \mathcal{R}[\mathcal{L}]$ and (b) if $x = \perp$ then ck is a fresh uniform random string, and *statement recovery* for the trapdoor receiver, which states that for all ℓ and $x \in \mathcal{U}$, if $(\pi, td) \leftarrow \mathsf{PG}(1^\kappa)$, $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x)$, and $(ck', x') \leftarrow \mathsf{TRec}_{\pi,\ell}(td, m)$, then $\Pr[x' \neq x] \leq \mathsf{negl}(\kappa)$.

An implicit-statement CKEM scheme must satisfy the *parameter indistinguishability*, *zero-knowledge*, and *simulation soundness* properties [6], which we adjust to implicit-statement CKEM's as follows:

(I) parameter indistinguishability: Distributions $\{\pi\}_{\pi \leftarrow \mathsf{PG}(1^\kappa)}$ and $\{\pi\}_{(\pi,td) \leftarrow \mathsf{TPG}(1^\kappa)}$ are computationally indistinguishable.

(II) zero-knowledge: The zero-knowledge defined in [6] required that the real-world receiver output $\mathsf{Rec}_{\pi,\ell}(x, w, m)$ is indistinguishable from the simulator output $\mathsf{TRec}_{\pi,\ell}(x, td, m)$. By contrast, in implicit-statement CKEM, Rec runs only on w, so it is not obvious what the corresponding ideal-world interaction should be, because w might correspond to many statements. However, since we require that Rec outputs statement x extracted from m along with key ck, and that Rec outputs $x \neq \perp$ only if $(x, w) \in \mathcal{R}[\mathcal{L}]$, the ZK property for implicit-statement CKEM will compare Rec's output with TRec's output modified by a wrapper that overwrites TRec's output (ck, x) with $(\$, \perp)$ if w is not a witness for x.[3] Formally, implicit-statement CKEM is *zero-knowledge* if for every efficient algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have:

$$\{\mathcal{A}_2(st, ck, x)\}_{(ck,x) \leftarrow \mathsf{Rec}_{\pi,\ell}(w,m)} \;\approx\; \{\mathcal{A}_2(st, ck, x)\}_{(ck,x) \leftarrow \mathsf{Wrap}_w(\mathsf{TRec}_{\pi,\ell}(td,m))}$$

where $(\pi, td) \leftarrow \mathsf{TPG}(1^\kappa)$ and $(st, \ell, w, m) \leftarrow \mathcal{A}_1(\pi, td)$ in both distributions, and Wrap_w is an algorithm which outputs (ck, x) on input (ck, x) if $(x, w) \in \mathcal{R}[\mathcal{L}]$ and otherwise outputs $(ck', \perp)$ for $ck' \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.

(III) simulation soundness: If $x \notin \mathcal{L}$ then $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x)$ is indistinguishable from $(\$, m)$ even if the adversary interacts with a trapdoor receiver on any $(\ell', m') \neq (\ell, m)$. Formally, for every efficient algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\{\mathcal{A}_2^{\mathsf{TRec}_{\mathsf{Block}(\ell,m)}(\pi,\cdot,td,\cdot)}(st, ck, m)\} \;\approx\; \{\mathcal{A}_2^{\mathsf{TRec}_{\mathsf{Block}(\ell,m)}(\pi,\cdot,td,\cdot)}(st, ck', m)\}$$

where $(\pi, td) \leftarrow \mathsf{TPG}(1^\kappa)$, $(st, \ell, x) \leftarrow \mathcal{A}_1^{\mathsf{TRec}(\pi,\cdot,td,\cdot)}(\pi)$ s.t. $x \notin \mathcal{L}$, $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x)$, $ck' \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$, and oracle $\mathsf{TRec}_{\mathsf{Block}(\ell,m)}(\pi, \cdot, td, \cdot)$ returns $\mathsf{TRec}_{\pi,\ell'}(td, m')$ on any query $(\ell', m') \neq (\ell, m)$.

Statement Privacy. As said above, an implicit-statement CKEM can support applications in which the statement assumed by the sender is hidden from everyone except the receiver who holds the matching witness. This is captured by the *statement privacy* property of CKEM, that for any statements x_0, x_1, both *not* in language $\mathcal{L}$, an adversary cannot tell whether the sender's message is produced on x_0 or x_1. Formally, we call CKEM *statement private* if for every efficient algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\{\mathcal{A}_2^{\mathsf{TRec}_{\mathsf{Block}(\ell,m_0)}(\pi,\cdot,td,\cdot)}(st, ck_0, m_0)\} \approx \{\mathcal{A}_2^{\mathsf{TRec}_{\mathsf{Block}(\ell,m_1)}(\pi,\cdot,td,\cdot)}(st, ck_1, m_1)\}$$

for $(\pi, td) \leftarrow \mathsf{TPG}(1^\kappa)$, and $(st, \ell, x_0, x_1) \leftarrow \mathcal{A}_1^{\mathsf{TRec}(\pi,\cdot,td,\cdot)}(\pi)$ s.t. $x_0, x_1 \notin \mathcal{L}$, $(ck_b, m_b) \leftarrow \mathsf{Snd}(\pi, \ell, x_b)$ for $b \in \{0,1\}$, and oracle $\mathsf{TRec}_{\mathsf{Block}(\ell,m)}(\pi, \cdot, td, \cdot)$ acts as in the definition of simulation soundness above.

[3] To see that this wrapper is necessary, observe that TRec should output $(\cdot, x)$ on input m output by $\mathsf{Snd}(x)$, whereas $\mathsf{Rec}(w)$ outputs $(\cdot, \perp)$ if $(x, w) \notin \mathcal{R}[\mathcal{L}]$.

Note on Statement Privacy: It may be surprising that we define statement privacy only for incorrect statements, $x_0, x_1 \notin \mathcal{L}$. Indeed, there are many ways to express the intuitive notion of statement privacy. We chose to state it only for statements $x_0, x_1 \notin \mathcal{L}$ but to allow the adversary to see both the sender's message m and the sender's local output ck, for $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x_b)$ for $b = 0, 1$. We could have instead allowed *any* adversarially chosen statements, including those in $\mathcal{L}$, but let the adversary see only the message m, and not the sender's local output ck. We cannot allow both because if the adversary chooses $(x_b, w_b) \in \mathcal{R}[\mathcal{L}]$ and then learns $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x_b)$, it can then run the receiver algorithm on (w_b, m) to test if it returns the same value ck. Our choice to restrict statements works better in the context of the higher-level saPAKE protocol of Section 4: Even though statement x used by the real-world sender party might be true (which is the case if the two parties have matching passwords), in the protocol simulation the statement is guaranteed to be false (the saPAKE simulator does not know any party's password when the protocol starts), and therefore the above statement privacy property suffices.

CKEM Security and Privacy Combined. The notion of simulation soundness and statement privacy can be combined into a single notion we call *simulatability*. Let $\mathsf{Snd}^{\mathsf{sim}}_{\pi,\ell}$ be the following simulator algorithm: $\mathsf{Snd}^{\mathsf{sim}}_{\pi,\ell}$ picks an arbitrary false statement $x' \notin \mathcal{L}$, computes $(ck, m) \leftarrow \mathsf{Snd}_{\pi,\ell}(x')$, picks $ck' \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$, and outputs (ck', m). We say that an implicit-statement CKEM is *simulatable* if for any efficient algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\{\mathcal{A}_2^{\mathsf{TRec}_{\mathsf{Block}(\ell,m)}(\pi,\cdot,td,\cdot)}(st, ck_0, m_0)\} \;\approx\; \{\mathcal{A}_2^{\mathsf{TRec}_{\mathsf{Block}(\ell,m')}(\pi,\cdot,td,\cdot)}(st, ck', m_1)\}$$

for $(\pi, td) \leftarrow \mathsf{TPG}(1^\kappa)$, and $(st, \ell, x) \leftarrow \mathcal{A}_1^{\mathsf{TRec}(\pi,\cdot,td,\cdot)}(\pi)$ s.t. $x \notin \mathcal{L}$, $(ck_0, m_0) \leftarrow \mathsf{Snd}_{\pi,\ell}(x)$ and $(ck', m_1) \leftarrow \mathsf{Snd}^{\mathsf{sim}}_{\pi,\ell}$.

Lemma 1. *An Implicit-statement CKEM is simulatable if and only if it is simulation sound and statement private.*

Proof Sketch: Simulatability implies simulation soundness because $\mathcal{A}_1$ can output $x_0 = x_1$. Simulatability also implies statement privacy, because if $(ck_0, m_0) \approx (\$, m_1)$, i.e., a distribution where the ck_1 part of (ck_1, m_1) is overwritten by a random string, and by simulation soundness $(\$, m_1) \approx (ck_1, m_1)$, then $(ck_0, m_0) \approx (ck_1, m_1)$. For the opposite direction, since statement privacy implies $(ck_0, m_0) \approx (ck_1, m_1)$, and simulation soundness implies $(ck_1, m_1) \approx (\$, m_1)$, together they imply that $(ck_0, m_0) \approx (\$, m_1)$.

Relation to CCA Security and Privacy. We note that simulation soundness and statement privacy for CKEM are analogous to CCA-security and CCA-anonymity for public key KEM. If we view a statement as an encryption public key, and its witness as a private key, then the Snd procedure is analogous to public key KEM encryption and the Rec procedure is analogous to decryption.

Moreover, the TRec procedure can be used by the simulator to implement access to the decryption oracle in the CCA security experiment: Recall that in the CCA security notion of KEM the adversary receives a challenge KEM ciphertext and must distinguish from random the KEM key encrypted in this ciphertext, given access to a decryption oracle which blocks the challenge ciphertext. The simulation soundness game follows the same structure, with the $\mathsf{TRec}_{\mathsf{Block}}$ oracle acting as the decryption oracle and the KEM message m playing the role of the ciphertext. The CCA anonymity KEM game is similar to the CCA security game, but with (key,ciphertext) challenge being generated on two randomly generated public keys. If the public keys are implemented as language statement, and if the language is a hard promise problem, i.e., if a random correct statement (=public key) cannot be distinguished from a random incorrect statement, then statement privacy implies CCA anonymity. Our notion of CKEM can be thus thought of as a generalization of CCA secure and anonymous PKE to CCA secure and anonymous *witness encryption* [19]. Indeed, the construction of simulation-sound CKEM from an SPHF in Section 2.1 below can be seen as a generalization of the Fujisaki-Okamoto transform [18] from IND PKE to CCA PKE.

2.1 Implicit-Statement CKEM Construction in ROM from SPHF's

We construct a non-interactive implicit-statement CKEM which is zero-knowledge, simulation sound, and statement private, for any language $\mathcal{L}$ which has a statement private Smooth Projective Hash Function (SPHF) [15]. Our construction, which is secure in the Random Oracle Model (ROM), is efficient: Its costs are as in the underlying SPHF plus, for the receiver, the cost of verifying that a projection key and a hash were computed correctly given the hash key. This verification can be done with a single multi-exponentiation using known batch signature verification techniques. We first describe the SPHF notion, and then show how to create a CKEM assuming an SPHF with the desired properties. We exemplify this generic construction in Section 5.1 for the case of language $\mathcal{L}$ used in our saPAKE construction.

Statement Private SPHF. We say that an algorithm tuple $(\mathsf{Hash}, \mathsf{PHash})$ is an SPHF for language $\mathcal{L} \subseteq \mathcal{U}$ (for $\mathcal{U}$ an implicit universe) if Hash on input a statement x outputs a *projection key* hp and a *hash value* v,[4] and PHash on input a witness w and hp outputs another hash value v'. Procedure Hash must be randomized and we will refer to its randomness as a *hash key* hk. Note that we assume that procedure PHash does not take a statement x as input, which is important for languages where one witness can correspond to many statements. Correctness requires that for all $(x, w) \in \mathcal{R}[\mathcal{L}]$, if $(v, hp) \leftarrow \mathsf{Hash}(x)$ then $v \leftarrow \mathsf{PHash}(w, hp)$. We will consider SPHF schemes that satisfy the statistical smoothness and statement privacy properties, defined as follows:

[4] Standard SPHF syntax uses two separate algorithms, $\mathsf{KG} \rightarrow (hp, hk)$ and $\mathsf{Hash}(x, hk) \rightarrow v$, which we combine for notational convenience in our context.

(I) Smoothness: For all $x \notin \mathcal{L}$

$$\{v, hp\}_{(v,hp)\leftarrow \mathsf{Hash}(x)} \overset{(s)}{\approx} \{v', hp\}_{(v,hp)\leftarrow \mathsf{Hash}(x), v'\leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}}$$

(II) Statement Privacy: For all $x_0, x_1 \in \mathcal{U}$ close:

$$\{hp\}_{(v,hp)\leftarrow \mathsf{Hash}(x_0)} \overset{(s)}{\approx} \{hp\}_{(v,hp)\leftarrow \mathsf{Hash}(x_1)}$$

CKEM Construction. Let $\mathsf{SPHF} = (\mathsf{Hash}, \mathsf{PHash})$ be a statement private SPHF for $\mathcal{L}$, let $\mathsf{H}_0 : \{0,1\}^* \to (\{0,1\}^{\kappa})^2$ and $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^{\kappa}$ be hash functions, and let (E, D) be an indistinguishable symmetric encryption with κ-bit keys. Consider $\mathsf{CKEM} = (\mathsf{PG}, \mathsf{TPG}, \mathsf{Snd}, \mathsf{Rec}, \mathsf{TRec})$ defined as follows:

- $\mathsf{PG}(1^{\kappa})$ and $\mathsf{TPG}(1^{\kappa})$ output descriptions of H_0 and H_1 as π. The trapdoor td output by TPG is access to the adversary's queries to random oracles $\mathsf{H}_0, \mathsf{H}_1$.
- $\mathsf{Snd}_{\pi,\ell}(x)$:
  1. Generate $(v, hp) \leftarrow \mathsf{SPHF.Hash}(x; hk)$ for random hk;
  2. Compute $(ek, ck) \leftarrow \mathsf{H}_0(v)$, $e \leftarrow \mathsf{E}_{ek}(hk, x)$, and $\tau \leftarrow \mathsf{H}_1(v, hp, e, \ell)$;
  3. Output (ck, m) for $m = (hp, e, \tau)$.
- $\mathsf{Rec}_{\pi,\ell}(w, m)$ for $m = (hp, e, \tau)$:
  1. Compute $v \leftarrow \mathsf{SPHF.PHash}(w, hp)$;
  2. Compute $(ek, ck) \leftarrow \mathsf{H}_0(v)$, $(hk, x) \leftarrow \mathsf{D}_{ek}(e)$, and $\tau' \leftarrow \mathsf{H}_1(v, hp, e, \ell)$;
  3. Output (ck, x) if $\tau' = \tau$, $(v, hp) = \mathsf{SPHF.Hash}(x; hk)$, and $(x, w) \in \mathcal{R}[\mathcal{L}]$; Otherwise output $(ck, \perp)$ for $ck \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.
- $\mathsf{TRec}_{\pi,\ell}(td, m)$ for $m = (hp, e, \tau)$:
  1. Among adversary's H_1 queries find $\tilde{v}$ s.t. $\tau = \mathsf{H}_1(\tilde{v}, hp, e, \ell)$; If none or more than one $\tilde{v}$ found, output $(ck, \perp)$ for $ck \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$;
  2. If unique $\tilde{v}$ found, set $(ek, ck) \leftarrow \mathsf{H}_0(\tilde{v})$ and $(hk, x) \leftarrow \mathsf{D}_{ek}(e)$;
  3. Output (ck, x) if $(\tilde{v}, hp) = \mathsf{SPHF.Hash}(x; hk)$; Otherwise output $(ck, \perp)$ for $ck \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.

Theorem 1. *If* SPHF *is a smooth and statement private SPHF for* $\mathcal{L}$ *and* E *is an indistinguishable encryption, then the above is a zero-knowledge, simulation sound, and statement private implicit-statement CKEM for* $\mathcal{L}$ *in ROM.*

Due to limited space, we defer the full proof of Theorem 1 to the full version, and give a short sketch of the main arguments below.

Proof of Theorem 1 (Sketch). Correctness, statement verification, and parameter indistinguishability can be easily verified by inspection of the protocol. In particular, correctness follows directly from correctness of the SPHF and correctness of the encryption scheme E. Property (a) of statement verification holds because Rec checks if $(x, w) \in \mathcal{L}$, and property (b) holds because Rec picks a random ck when $x = \perp$. Finally, parameter

indistinguishability holds trivially because PG and TPG generate public parameters in an identical way.

Statement recovery holds because a sender message commits to values (hp, e, τ) via the random oracle, and TRec, given access to the RO table, can then deterministically compute the statement x. In particular, the sender sets $\tau = \mathsf{H}_1(v, hp, e, \ell)$, where v is the SPHF value, so TRec will find v in the RO table except with the negligible probability of a collision in H_1. If v is found, then TRec will get the same statement x from $\mathsf{D}_{ek}(e)$ by correctness of the symmetric encryption, as the key ek is deterministically computed by $\mathsf{H}_0(v)$.

Zero-knowledge follows from the correctness of the underlying SPHF scheme. Recall that zero-knowledge requires, essentially, that no efficient adversary can distinguish between the outputs of Rec and TRec for valid (statement,witness) pairs. We give an intuitive argument as to why this is the case. In our construction, as long as Rec and TRec find the same SPHF hash key v that was used to generate the message m, they will end up with the same statement x, which will lead to identical outputs (if the message is invalid, i.e., not computed by Snd, this also will be detected). A challenge is then in showing that Rec and TRec do indeed find the same SPHF hash value v. Intuitively, this is the case because the adversary commits to message elements hp and e along with hash value v via the random oracle hash H_1. Because of this commitment, TRec will find the hash value v in the random oracle table, and can check its validity through the call to $\mathsf{SPHF.Hash}$. By SPHF correctness, if $(w, x) \in \mathcal{R}[\mathcal{L}]$, and hp is valid, then Rec's call to $\mathsf{SPHF.PHash}(w, hp)$ will produce the same hash value v. Rec checks both that $\mathcal{A}$ indeed committed to the hash value by checking if $\tau = \mathsf{H}_1(v, hp, e, \ell)$, and checks the validity of the hash value through the call to $\mathsf{SPHF.Hash}$. Rec also checks directly if $(x, w) \in \mathcal{R}[\mathcal{L}]$, which is not performed by TRec (since TRec has no witness w), but it is performed in the ZK experiment by the wrapper Wrap_w over the (ck, x) output of TRec. The full proof captures the above in a sequence of game changes showing that adversary's interactions with the Rec and the (wrapped) TRec procedures are indistinguishable.

By Lemma 1, simulation soundness and statement privacy follow from the *simulatability* property that captures them both. Simulatability of our CKEM construction follows from the smoothness and statement privacy of the underlying SPHF scheme, and the indistinguishability of the symmetric encryption scheme. Simulatability requires that no efficient adversary can distinguish between the real output (key, message) from Snd on input x_0, and a pair (random key, message) where the message is from Snd on input x_1, even in the presence of a trapdoor receiver oracle. Because the CKEM key is computed as the hash of the SPHF hash value v, we know that the CKEM key will appear random as long as the hash value is not known. SPHF smoothness gives us that hash values v computed from $\mathsf{SPHF.Hash}(x; hk)$ on $x \notin \mathcal{L}$ appear random and independent of projection keys hp as long as key hk used in this hash remains secret. However, an encryption of the randomness hk used to generate v is provided to the adversary, and the encryption key is in turn

created from $\mathsf{H}_0(v)$. This is a circular encryption, but in the random oracle model we avoid this circularity by first assuming that the adversary does not query either H_1 on v, which means that the encryption key appears random, which in turn means, by indistinguishability of E, that the ciphertext e does not reveal anything about hk. Further, if the adversary also does not query H_1 on v, then the tag τ appears random as well. At this point, we can safely invoke the SPHF smoothness property to say that v appears random and independent of hp, and the probability of querying either random oracle on v is then negligible. Finally, we use SPHF statement privacy, which gives us that projection keys hp_0, hp_1 are indistinguishable if created via Hash on two different statements not in the language $\mathcal{L}$, to show that we may always create the projection key with x_1 without being detected. □

3 Security of a Password File against Dictionary Attacks

In a strong asymmetric PAKE protocol, the salted function F used to hash a password must be a *one-way* function. However, not all one-way functions will work: To use the Encryption+SPHF approach to UC PAKE construction F needs to have an arithmetic structure that admits an efficient SPHF for the authentication protocol. Unfortunately, an arithmetic structure makes it harder to characterize F's resistance to brute-force attacks, i.e., to lower-bound the computational complexity of finding pw given $(s, F_s(pw))$ where s is a random salt and pw is sampled from a polynomial-size "password dictionary" set D.

To quantify post-compromise resistance of a password file to brute-force attacks, Benhamouda and Pointcheval introduced the notion of *tight one-wayness* [7], which we will adapt to the case where the one-way function is *salted*, i.e., randomized, as is necessary in an saPAKE, and we propose to model the resulting *Strong* Tight One-Way Function (STOWF) notion with a UC functionality. We explain why some STOWF candidates do not work for our purposes, we propose a new STOWF candidate, an SPHF-friendly function that uses the Boneh-Boyen signature [9] with the hashed password as a key and the salt as a signed message, and we show that this function realizes the $\mathcal{F}_{\mathsf{STOWF}}$ functionality in the *programmable* Generic Group Model (GGM) and the Random Oracle Model.

While it seems unavoidable to use GGM to analyze the fine-grained hardness of an algebraic salted one-way function F_s, we do not rely on GGM in the security analysis of the saPAKE protocol that uses $(s, F_s(pw))$ as the password file. Abstracting the one-wayness property as a UC functionality $\mathcal{F}_{\mathsf{STOWF}}$ helps keep this argument modular, and in Section 4 we show an saPAKE protocol that realizes the ideal saPAKE functionality given *any* realization of $\mathcal{F}_{\mathsf{STOWF}}$ which satisfies some additional properties which we explain below.

Modeling the Password File with *Salted* Tight One-Way Function. Since we assume that passwords come from a polynomial-sized dictionary, the adversary can learn the correct password by computing $F(pw)$ for all passwords

pw in the dictionary. More precisely, if the argument pw is known to come from some domain subset D then given $z = f(pw)$ an algorithm that evaluates F on any fraction ϵ of D will find pw with probability ϵ. This sets an upper bound on the hardness for the problem of inverting F on a subdomain, and we call F a *tight one-way function* (TOWF) [7] if this is also the lower bound, i.e., if for any polynomial-size subset D of the domain of F any algorithm which runs in time $\epsilon \cdot |D|$ has at most ϵ probability of inverting $F(x)$ for $x \leftarrow_{\mathrm{R}} D$. For example, this holds if function F is a random oracle. On the other hand, any additional structure in the one-way function might make it not tight. For example, if F is additively homomorphic, e.g. $F(x) = g^x$ where g generates a multiplicative group, and D is an integer interval, then the Baby-Step Giant-Step algorithm finds x given $F(x)$ in time $O(\sqrt{|D|})$ with probability 1.

The goal of a *strong* asymmetric PAKE is to further constrain the adversary so that an ϵ-advantage attacker must perform $\epsilon \cdot |D|$ computation *after* server compromise. This means that the password file must be created by a randomized, a.k.a. *salted*, one-way function. Let $\{F_s\}_{s \in R}$ be a family of functions that share the same domain and range and are indexed by values s we call *salt*. Informally, we call $\{F_s\}$ a family of *salted tight one-way functions* (STOWF) if for any domain subset $D \subseteq X$ it holds that any efficient algorithm $\mathcal{A}$ that has ϵ probability of computing $F_s^{-1}(z)$ given $(s, z) = (s, F_s(x))$ for $s \leftarrow_{\mathrm{R}} R$ and $x \leftarrow_{\mathrm{R}} D$, must perform at least $\epsilon \cdot |D|$ computation *after* receiving (s, z) as an input. In other words, no efficient pre-computation can help the adversary to avoid the *post-compromise* cost $\Omega(|D|)$ to recover pw given the compromised server password file $(s, F_s(pw))$ if $(s, pw) \leftarrow_{\mathrm{R}} R \times D$.

Formally, STOWF is defined by a pair of efficient algorithms $(\mathsf{PG}, \mathsf{Eval})$ where (1) $\mathsf{PG}(1^\kappa)$ outputs a description of a function family F, with domain X, salt domain R, and range Y, such that $F_s : X \to Y$ for every $s \in R$, and (2) algorithm Eval evaluates $F_s(x)$ given $(s, x) \in R \times X$ and the description of F.

Examples of Salted Tight One-Way Functions. In the Password-over-TLS authentication used on the web today the server-held password file is (s, z) for $z = F_s(pw) = \mathsf{H}(s, pw)$, and to authenticate the client sends password pw' over the server-to-client PKI-authenticated TLS session and the server accepts if $\mathsf{H}(s, pw') = z$. This method enforces the STOWF lower bound if H is a random oracle: If $|s| = \Omega(\kappa)$ then regardless of any (efficient) pre-computation the adversary can recover the client's password only by computing $\mathsf{H}(s, x)$ for $x \in D$ after server compromise. However, a plain random oracle has no arithmetic structure, so it is not clear how to use it as a STOWF in an saPAKE protocol.

The recently proposed PKI-free saPAKE scheme OPAQUE [24] gives a different example of a tight STOWF. The syntax differs slightly, as the password file is $(s, F_{s,r}(pw))$ where r is an additional randomness needed to evaluate F, but the crux of the scheme is that s is implemented as a key k of an Oblivious PRF (OPRF) function F^*, and $F_{s,r}(pw)$ includes public keys pk_{C} and pk_{S} for the client and the server, the private key sk_{S} for the server, and a ciphertext $c \leftarrow \mathsf{E}_{rw}(sk_{\mathsf{C}})$ which encrypts the client private key sk_{C} under key

$rw = F^*_k(pw)$ (see [24] for details). To authenticate, the client computes rw via an OPRF instance with the server on resp. inputs pw and k, decrypts sk_{C} from c, and the two parties run a standard AKE using resp. keys sk_{C} and sk_{S}. The STOWF bound is enforced because from a server compromise the attacker learns (k, c) and needs to compute $sk_C = \mathsf{D}_{rw}(c)$ for $rw = F^*_k(pw)$. The strong UC OPRF properties [23], realizable efficiently in ROM, imply that F^* is pseudorandom even if one holds key k, hence the only strategy for finding pw (and sk_C) given (k, c) is to evaluate F^*_k on guesses pw' and verify if $sk'_{\mathsf{C}} = \mathsf{D}_{rw'}(c)$ for $rw' = F^*_k(pw')$ corresponds to pk_{C}.

A natural SPHF-friendly STOWF candidate is $F_s(pw) = s^{\mathsf{H}(pw)}$ where $s \leftarrow_{\mathrm{R}} \mathbb{G}$, where $\mathbb{G}$ is a cyclic group in which the discrete logarithm problem is hard, and H is an RO hash onto $\mathbb{Z}_p$. Using this function, we could create an authentication protocol based on an efficient SPHF scheme for the language that server-held value $z = s^{\mathsf{H}(pw)}$ and client's extractable password commitment $c = (g^r, y^r g^{\mathsf{H}(pw')})$ for $g, y \in \mathbb{G}$ are of the correct form, and indeed [7] construct their aPAKE based on this STOWF candidate along these lines. This function realizes the UC STOWF functionality we define below in GGM, and the proof is a straightforward adaptation of the proof that it satisfies a game-based TOWF property given in [7]. However, this function is *malleable* in the following sense: Given (s, z) for $z = F_s(pw)$, it is easy to create (s', z') for $z' = F_{s'}(pw)$ and $s' \neq s$, by computing $s' = s^r$ and $z' = z^r$ for any r. This creates a problem for UC security of saPAKE: An adversary who learns (s, z) via server compromise can impersonate the server using a randomized file (s', z') without learning pw via an offline attack, but the UC simulator cannot detect that such impersonation attacks because if (s, z) is an STOWF challenge and an adversary does not stage an offline attack then the simulator does not know the trapdoor $\mathsf{H}(pw) = \mathsf{DL}(s, z)$ needed to recognize (s, z, s', z') as DDH tuples. One can prove UC security of a protocol based on this STOWF function but the proof would use GGM in the online part, whereas we hope to use GGM only to analyze the tightness of F against offline computation, and not involve it in the analysis of the online protocol.[5]

Boneh-Boyen Function: SPHF-Friendly and *Unforgeable* STOWF. Fortunately, we can avoid GGM in the security proof of saPAKE based on UC secure STOWF by using the *hashed* Boneh-Boyen (BB) function $F_s(x) = g^{1/(s+\mathsf{H}(x))}$ [9] as the STOWF candidate, where $s \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and H hashes onto $\mathbb{Z}_p$. We also define another STOWF candidate we call *unhashed* BB function, $f_s(x) = g^{1/(s+x)}$, which can be used to define F as $F_s(x) = f_s(\mathsf{H}(x))$. The reason for introducing the unhashed BB function f is that the saPAKE security proof will rely on the STOWF candidate f satisfying some additional properties we define below, but it will implement the password file as $(s, f_s(\mathsf{H}(pw))$. If f is instantiated as the *unhashed* BB then the saPAKE protocol effectively uses the *hashed* BB function for its password file. This will

[5] Similar approach was taken by [7] who contain both GGM and ROM to the offline part of analysis, while we contain GGM to the offline part but use ROM throughout.

in particular imply that we can rely on the offline-hardness of STOWF F while also benefitting of the algebraic properties of f.

UC Model for Salted Tight One-Way Function. We propose a UC functionality $\mathcal{F}_{\mathsf{STOWF}}$, shown in Fig. 1, to model the offline security of the password file in an ideal saPAKE scheme, i.e., the potential leakage of the password file via server compromise and the offline dictionary attack possible after this leakage. If a function realizes this UC functionality then for every efficient real-world adversary $\mathcal{A}$ there exists an efficient ideal-world adversary (simulator) SIM, such that $\mathcal{A}$'s offline computation can be simulated by SIM on access to $\mathcal{F}_{\mathsf{STOWF}}$, which essentially implements an ideal black-box point function, namely a function which outputs 1 for $x = pw$ and 0 for all $x \neq pw$.

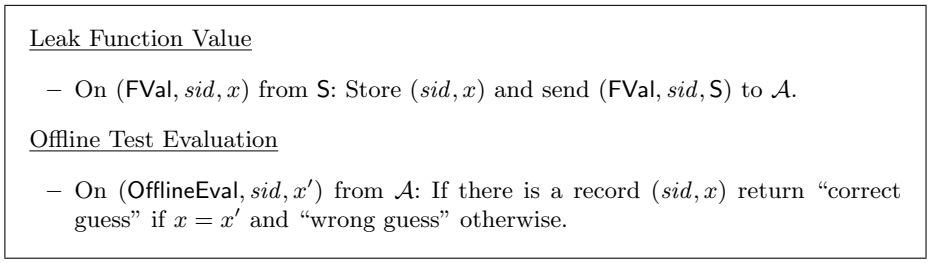

Fig. 1. UC functionality $\mathcal{F}_{\mathsf{STOWF}}$ for Tight One-Way Function (STOWF)

Modification of the UC Framework. The standard UC framework applied to functionality $\mathcal{F}_{\mathsf{STOWF}}$ does not express all required properties of the STOWF function, because it does not impose a tight relation between the time complexity of $\mathcal{A}$ and SIM. For example, the unhashed Boneh-Boyen function F realizes $\mathcal{F}_{\mathsf{STOWF}}$ in (programmable) GGM, but if $\mathcal{A}$ makes T generic group model operations then SIM must be allowed to make $O(T^2)$ queries to $\mathcal{F}_{\mathsf{STOWF}}$, which means that $\mathcal{A}$ can test $|D|$ passwords using only $O(\sqrt{|D|})$ group operations, as is indeed possible if D is an integer interval. Using a hashed Boneh-Boyen function heals this problem by effectively changing D into a *random* subset of $\mathbb{Z}_p$, but we need to make an adjustment to the UC model so that it imposes a tight relation between the work of the real-world adversary $\mathcal{A}$ and the simulator SIM.

As pointed out in [21], in order for the UC (s)aPAKE model to enforce that the real-world adversary performs some minimal local computation to offline test each password, we need to change the UC framework slightly, so that both the local computation of the real-world adversary and the $\mathsf{OfflineTestPwd}$ messages of the simulator are accounted for by the environment. For example, when the real-world adversary performs some local computation which corresponds to an offline password verification, e.g. a random oracle query or a generic group model query, this idealized computational element would send a special "flag" signal to the

environment. In the ideal world, the same flag would be sent to the environment whenever $\mathcal{F}_{\mathsf{STOWF}}$ receives an OfflineTestPwd query from the simulator. (In the context of $\mathcal{F}_{\mathsf{STOWF}}$ such flag will be sent on every OfflineEval message from the ideal-world adversary.) If the environment's view of the ideal-world and the real-world executions are indistinguishable in such regime this would prevent the simulator from sending OfflineTestPwd messages to the functionality without an offline attack taking place in the real world.[6]

The necessity of such modification to the UC framework was also observed by [24] in the context of saPAKE, because otherwise $\mathcal{F}_{\mathsf{aPAKE}}$ and $\mathcal{F}_{\mathsf{saPAKE}}$ would be equivalent. The only difference between aPAKE and saPAKE is that OfflineTestPwd queries are allowed in saPAKE only *after* the adversary compromises the server. However, if the timing of the local computation of the real-world adversary and the simulator's OfflineTestPwd queries is not observed by the environment, then the simulator may accumulate all the offline password tests made by the real-world adversary before server compromise, and then send all the OfflineTestPwd queries these password tests represented right after server compromise, effectively bypassing the intended enforcement of no pre-computation of offline dictionary queries in the real-world.

Additional STOWF Properties. For the saPAKE application we need to extend the STOWF notion with a secondary "leakage function" $\hat{F}$, because in the saPAKE protocol of Section 4 the client will commit not to the (hashed) password itself but to another algebraic function of it, which enables both straight-line extraction of the committed password and an efficient SPHF that the committed password is the pre-image of the STOWF function value stored by the server. Formally, we extend the STOWF syntax to a triple of algorithms $(\mathsf{PG}, \mathsf{Eval}, \mathsf{Leak})$, where PG and Eval are as before except PG also outputs a description of a leakage function $\hat{F}$ with domain X and range $\hat{Y}$ such that $\hat{F} : X \to \hat{Y}$, and (2) algorithm Leak evaluates $\hat{F}(x)$ given $x \in X$ and the description of $\hat{F}$. When it is unambiguous, we will use the shorthand $(\mathsf{PG}, F, \hat{F})$ to refer to a specific STOWF scheme.

We define the following three properties of an STOWF candidate. Since we will claim them only about the *unhashed* BB function f we will use notation $(\mathsf{PG}, f, \hat{f})$ to specify the properties below:

(I) One-time unforgeability. An efficient adversary who gets the server's password file $(s, f_s(x))$ for *random* $x \leftarrow_{\mathrm{R}} X$ and $s \leftarrow_{\mathrm{R}} R$, must be unable to generate an alternate file (s', z') for $z' = f_{s'}(x)$ and $s' \neq s$, except with negligible probability.

(II) Leakage-function hiding. An efficient adversary who gets the server's password file $(s, f_s(x))$ for *random* $x \leftarrow_{\mathrm{R}} X$ and $s \leftarrow_{\mathrm{R}} R$, must be unable to output $\hat{f}(x)$, except with negligible probability.

(III) Collision resistance. For all $s \in R$, if $x_1 \neq x_2$ then $f_s(x_1) \neq f_s(x_2)$.

[6] In [21] the environment sends permissions to the real-or-ideal adversary rather than receiving signals about the computation performed, but the effect seems the same.

Security of Boneh-Boyen Function as STOWF. We consider both the unhashed and hashed Boneh-Boyen functions. The parameter generator $\mathsf{PG}(1^\kappa)$ for both functions fixes a prime-order cyclic group $(\mathbb{G}, p)$ two random group elements $g, g' \leftarrow_{\mathrm{R}} \mathbb{G}$, and the randomness domain is $R = \mathbb{Z}_p$. The unhashed BB function pair $(f, \hat{f})$ is defined on domain $X = \mathbb{Z}_p$ as $f_s(x) = g^{1/(s+x)}$ and $\hat{f}(x) = (g')^x$, while the hashed BB function pair $(F, \hat{F})$ is defined as $F_s(x) = f_s(\mathsf{H}(x))$ and $\hat{F}(x) = \hat{f}(\mathsf{H}(x))$ where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$.

We make the following claims about the unhashed and hashed Boneh-Boyen functions. Recall that the q-DDH assumption [10] says that given $(g, g^x, ..., g^{x^q})$ for $x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ no efficient algorithm can find $(c, g^{1/(x+c)})$ for any $c \in \mathbb{Z}_p$.

Theorem 2. *The unhashed Boneh-Boyen function f satisfies the properties of one-time unforgeability, leakage-function hiding, and collision resistance under the q-SDH assumption for $q = 2$.*

Proof (Sketch). Boneh-Boyen [10] show that a signature/MAC scheme $S_x(m) = g^{1/(x+m}$ for key $x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ is unforgeable against chosen-message attack with q queries under the q-SDH assumption. Our one-time unforgeability notion is strictly weaker than this, because the adversary sees a signature $z = S_x(m)$ on only one randomly chosen message $m = s$. By a simple modification of Lemma 9 of [10] adapted to the setting where the challenger does not generate the public verification key, the BB function is one-time unforgeable under the q-SDH assumption for $q = 1$. Similarly, the leakage-function hiding property reduces to the 2-SDH assumption by a simple modification of the same reduction from one-time unforgeability of f to q-SDH. Finally, collision resistance holds for f because g is an element of group $\mathbb{G}$ of prime order p and division modulo p is a one-to-one function. (Formally, the domain of f_s must be restricted to exclude message value $x = -s \bmod p$.) □

Theorem 3. *The unhashed Boneh-Boyen STOWF function realizes UC functionality $\mathcal{F}_{\mathsf{STOWF}}$ in the (programmable) Generic Group Model (GGM) for $\mathbb{G}$, assuming that the i-th generic group operation made by the real-world adversary triggers i "offline password test computation" flags to the environment, which means it allows the simulator to issue a batch of i* OfflineTestPwd *queries to $\mathcal{F}_{\mathsf{STOWF}}$.*

Proof (Sketch). The proof of the above theorem, included in [13], is a straightforward generalization of the $O(T^2)$ bound on the number of discrete logarithm candidates which can be tested in GGM in T steps. Such lower-bound holds because the i-th generic group operation can test at most i new discrete logarithm candidates, limiting the total number of values tested in T steps to $\sum_{i=1}^{T}(i) = O(T^2)$. Moreover, in the programmable GGM the simulator can embed the result of OfflineTestPwd queries in the result of the generic group operation: The "correct guess" response implies that the new group element should collide with a specific previously computed element, while for the "wrong guess" the new group element representation is a fresh random string. □

Theorem 4. *The hashed Boneh-Boyen STOWF function realizes UC functionality* $\mathcal{F}_{\mathsf{STOWF}}$ *in the (programmable) Generic Group Model (GGM) for* $\mathbb{G}$ *and (non-progammable) ROM, assuming that each generic group operation made by the real-world adversary triggers* $O(1)$ *"offline password test computation" flags to the environment, which means it allows the simulator to issue a batch of (amortized)* $O(1)$ $\mathsf{OfflineTestPwd}$ *queries to* $\mathcal{F}_{\mathsf{STOWF}}$.

Proof (Sketch). The proof is a modification of the proof of Theorem 3 but it crucially utilizes a theorem shown by Schnorr [31], which shows that for a polynomial-sized subset D of *random* points in $\mathbb{Z}_p$, which we define as H outputs on values hashed by the real-world adversary, the generic group model algorithm must make $\Omega(|D|)$ operations to test if a discrete logarithm challenge is solved by the candidates in set D.[7] □

4 Strong aPAKE from Implicit-Statement CKEM

We show the saPAKE protocol we propose in Figure 2. The construction relies on several building blocks: (1) hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p$ modeled as a random oracle; (2) zero-knowledge, simulation sound, and statement private implicit-statement CKEM scheme $(\mathsf{PG}, \mathsf{Snd}, \mathsf{Rec})$ for language $\mathcal{L}$ described below; (3) one-time unforgeable, leakage-function hiding, and collision-resistant UC STOWF scheme $(\mathsf{PG}, f, \hat{f})$; and (4) CPA-secure public key encryption scheme $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ on message space $\hat{Y}$, the range of the leakage function $\hat{f}$. The common reference string (CRS) consists of a public key $pk \leftarrow \mathsf{KG}(1^\kappa)$, CKEM parameters $\pi \leftarrow \mathsf{CKEM.PG}(1^\kappa)$, and functions $(f, \hat{f})$ generated by STOWF generator $\mathsf{PG}(1^\kappa)$.

The high-level idea of this construction is as follows. The server S's password file contains a random salt and salted hash (s, z) for $z = f_s(hw_\mathsf{S})$, where $hw_\mathsf{S} = \mathsf{H}(pw_\mathsf{S})$. To authenticate, client C encrypts its hashed password $hw_\mathsf{C} = \mathsf{H}(pw_\mathsf{C})$ under the public key pk from the CRS, and sends the resulting ciphertext c to S. Server S uses the CKEM scheme to form a key k_S and message m conditioned on the statement $x_\mathsf{S} = (s, z, c)$ that c encrypts the same value hw as was used to form $z = f_s(hw)$. Client C uses hw_C and the randomness in c as its witness in the CKEM receiver procedure, which computes the key k_C. (The receiver algorithm also outputs the value x_C implicit in m, but this value is not used by the client.) The properties of CKEM guarantee that if x_C does not match C's witness then the client's key k_C will be random independent of the server's view, so S fails to authenticate to C. Likewise, if C doesn't have a witness for S's statement x_S for which S created this CKEM message, then the server's key k_S will be independent of the client's view, so C fails to authenticate to S.

[7] A variant of the theorem of Schnorr was also shown by Benhamouda-Pointecheval [7], but customized to n-bit passwords for $n < |p|/4$.

CKEM Language. The CKEM used in the saPAKE protocol saPAKE of Figure 2 is defined as follows:

$$\mathcal{L}_{pk} = \{(s, z, c) \mid \exists\ (h, r) \text{ s.t. } z = f_s(h) \text{ and } c = \mathsf{Enc}_{pk}(\hat{f}(h); r)\} \qquad (1)$$

Since key pk is part of a CRS, we will leave it as implicit and refer to $\mathcal{L}_{pk}$ as simply $\mathcal{L}$. Note that if function $\hat{f}$ is hard to invert then $\mathsf{Enc}_{pk}(\hat{f}(m))$ is not a standard encryption scheme, as there is no efficient decryption procedure. However, since it is used to encrypt a hashed password, i.e. $m = \mathsf{H}(pw)$, it still implements a commitment with straight-line extractor in ROM, which the simulator uses to extract the client's password in the proof of Theorem 5 below.

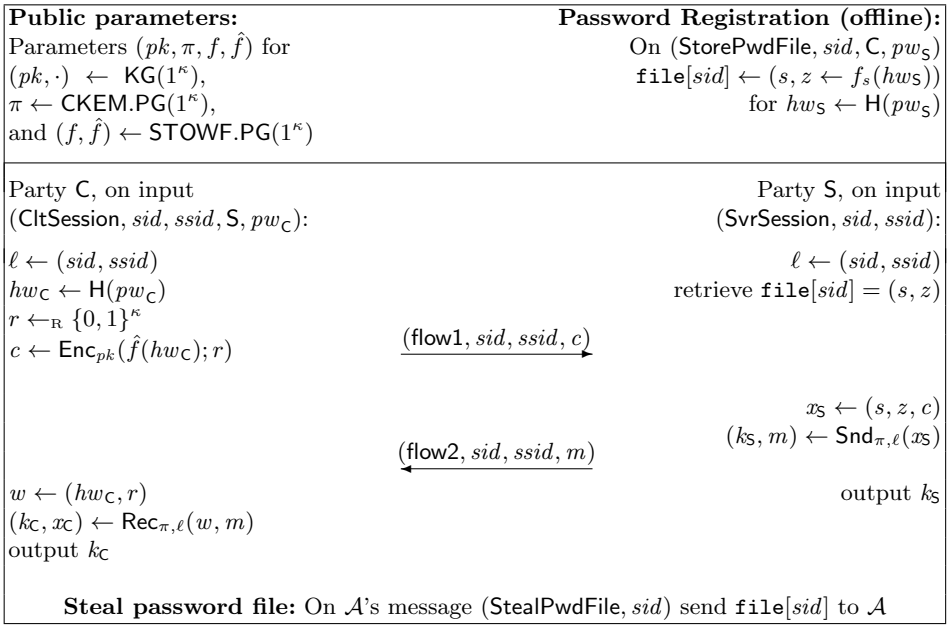

Fig. 2. Strong aPAKE scheme saPAKE based on an implicit-statement CKEM

4.1 Security Analysis

We now show that saPAKE is a secure realization of functionality $\mathcal{F}_{\mathsf{saPAKE}}$, originally proposed by [24]. The formal security claim about protocol saPAKE is as follows:

Theorem 5. *The saPAKE protocol shown in Fig. 2 securely realizes functionality $\mathcal{F}_{\mathsf{saPAKE}}$ in ROM, provided that function f securely realizes functionality $\mathcal{F}_{\mathsf{STOWF}}$ and is (one-time) unforgeable, leakage-function hiding and collision-resistant,* PKE *is a CPA-secure public-key encryption, and* CKEM *is a zero-knowledge, simulatable CKEM for language $\mathcal{L}$.*

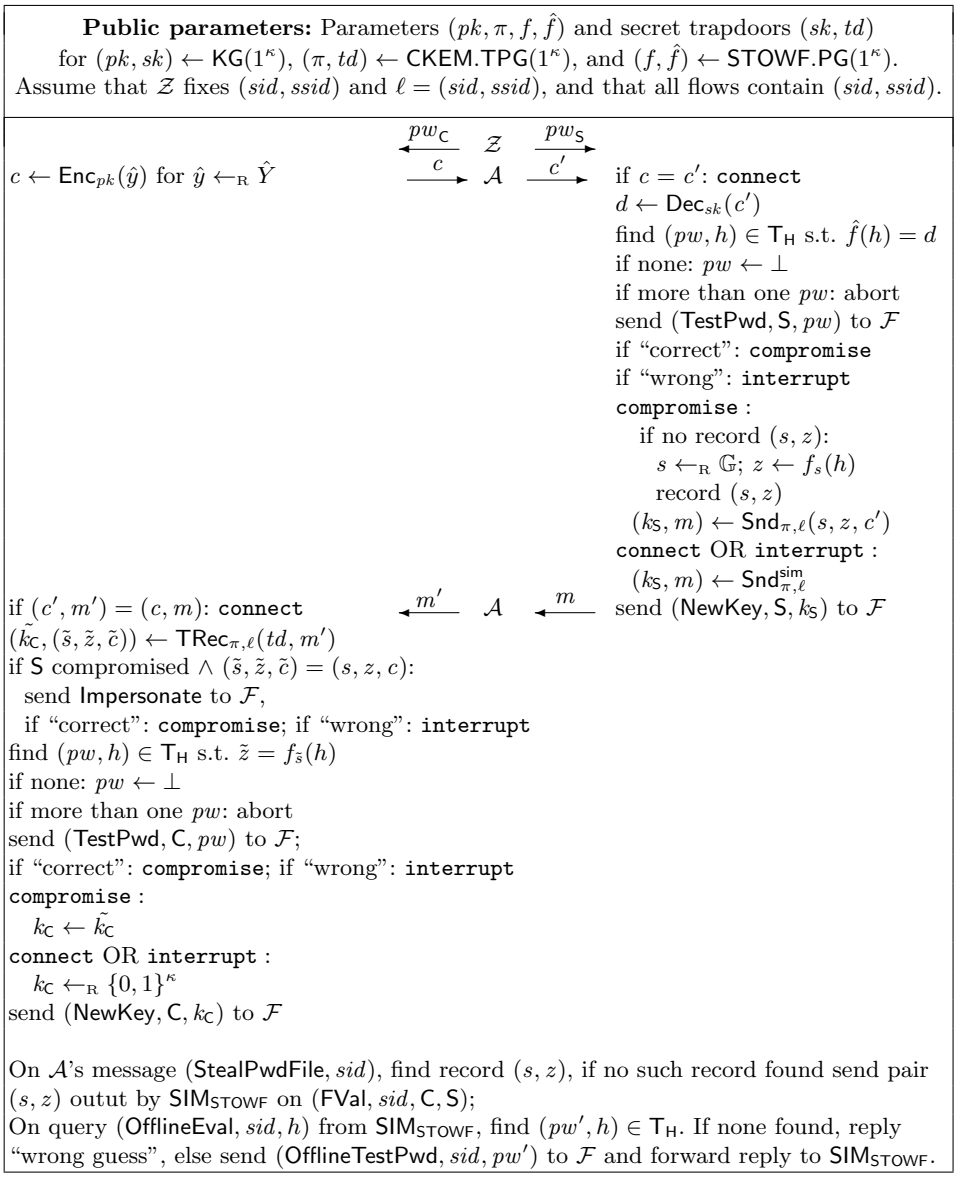

Fig. 3. Simulator alg. SIM for saPAKE interacting with $\mathcal{A}$, $\mathcal{F}$ and $\mathsf{SIM_{STOWF}}$.

Due to space limits, we include here only the simulator for the protocol and the quick overview of the proof ideas, defering the full proof of the theorem to the full version of this paper [13]. To prove Theorem 5, we construct a simulator SIM, shown in Figure 3, s.t. that for any efficient environment $\mathcal{Z}$ and adversary $\mathcal{A}$, the environment's view of the real-world execution, where adversary $\mathcal{A}$ interacts with the honest parties that execute protocol saPAKE, is indistinguishable from its view of the ideal-world execution, where simulator

SIM (which uses $\mathcal{A}$ as an oracle) ineracts with the ideal-world honest parties via the ideal functionality $\mathcal{F}_{\mathsf{saPAKE}}$. Without loss of generality, we assume that $\mathcal{A}$ is a "dummy" adversary who merely passes all messages between $\mathcal{Z}$ and SIM. In the description of the simulator in Figure 3 we shorten $\mathcal{F}_{\mathsf{saPAKE}}$ as $\mathcal{F}$. Note that because by assumption function f securely realizes functionality $\mathcal{F}_{\mathsf{STOWF}}$, there exists a simulator $\mathsf{SIM}_{\mathsf{STOWF}}$ such that no efficient environment can distinguish between interacting with the real STOWF protocol and with $\mathsf{SIM}_{\mathsf{STOWF}}$ and $\mathcal{F}_{\mathsf{STOWF}}$. The saPAKE protocol simulator SIM shown in Figure 3 uses this $\mathsf{SIM}_{\mathsf{STOWF}}$ as a black box. It also uses the CKEM sender simulator algorithm $\mathsf{Snd}^{\mathsf{sim}}$ defined for Lemma 1 in Section 2.

Proof Overview. The proof uses a sequence of games, starting from the real world and ending at the ideal world. The first step is to let the game abort if there is a collision in H, or H outputs 0. Then consider the case that $\mathcal{A}$ merely passes all messages between C and S: if C and S use the same password then they output the same key, otherwise C outputs an independent random key. The resultant game is indistinguishable to the previous one due to CKEM correctness and statement verification. Then according to the zero-knowledge property of CKEM, we modify the client-side code so that it computes k_{C} with the trapdoor receiver TRec and trapdoor td instead of the standard receiver Rec and witness w, and then performs a check that the client's statement is in the language. Now the witness $w = (hw_{\mathsf{C}}, r)$ and the secret key sk are not used in the game, so we change client's ciphertext to a "dummy" one $c \leftarrow \mathsf{Enc}_{pk}(\hat{y})$ for $\hat{y} \leftarrow_{\mathrm{R}} \hat{Y}$. After that, in the case that (1) the ciphertext sent to S, c', is new and invalid, or (2) $c' = c$, we let S output a random key k_{S} and send a "dummy" message m on a fixed false statement x'; this move can be made due to CKEM simulatability. Then we let C detect server impersonation and extract password guess, i.e., (1) if S is compromised, and $(\tilde{s}, \tilde{z}, \tilde{c}) = (s, z, c)$, or (2) if there is a pw s.t. $\tilde{z} = f_{\tilde{s}}(\mathsf{H}(pw))$, k_{C} is decided according to whether $pw_{\mathsf{C}} = pw_{\mathsf{S}}$. Then we remove the query $\mathsf{H}(pw_{\mathsf{C}})$ from the client-side code, and postpone the $\mathsf{H}(pw_{\mathsf{S}})$ query until server compromise. Finally, when $\mathcal{A}$ compromises S, invoke $\mathsf{SIM}_{\mathsf{STOWF}}$ to simulate $\mathcal{A}$'s view in offline password tests.

5 Efficient Instantiation of Strong aPAKE

The efficiency of the generic saPAKE construction in Fig. 2 depends on the choices of the salted tight one-way function (STOWF), the encryption scheme, and a CKEM. A particularly efficient instantiation of this framework, protocol $\mathsf{saPAKE\text{-}BB}$ shown in Figure 4, results from implementing the STOWF scheme $(\mathsf{PG}, f, \hat{f})$ as the unhashed Boneh-Boyen function $f_s(x) = g^{1/(s+x)}$ for $s \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and $\hat{f}(x) = (g')^x$ for $g, g' \leftarrow_{\mathrm{R}} \mathbb{G}$ (see Section 3). Since in the generic protocol in Figure 2 function f is evaluated on hashed password $hw \leftarrow \mathsf{H}(pw)$, the server's password file (s, z) in protocol $\mathsf{saPAKE\text{-}BB}$ is effectively computed using the *hashed* Boneh-Boyen function, i.e. $z = f_s(\mathsf{H}(pw)) = g^{1/(s+\mathsf{H}(pw))}$. PKE Enc is instantiated as ElGamal, i.e. $\mathsf{Enc}_{pk}(m; r) = (c, d) = (g^r, y^r \cdot m)$ where $pk = y$ for

$y \leftarrow_{\mathrm{R}} \mathbb{G}$, and CKEM is implemented using the SPHF-based CKEM construction of Section 2.1 instantiated for a language defined in equation (2) below.

In the following two subsections we explain how the generic CKEM scheme of Section 2.1 is instantiated for a language implied by the above Enc and f_s choices, and then we discuss the implications of these choices to the efficiency of protocol saPAKE-BB.

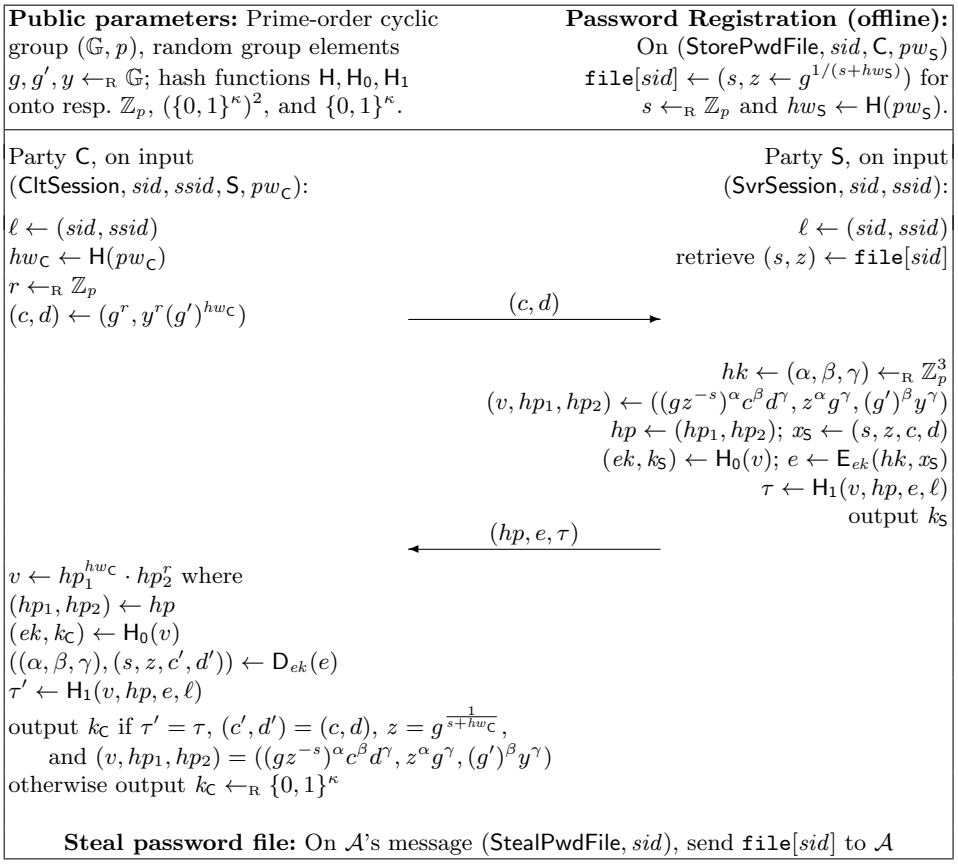

Fig. 4. saPAKE-BB: Instantiation of protocol saPAKE with Boneh-Boyen STOWF, ElGamal PKE, and SPHF-based CKEM

5.1 Efficient CKEM for Commitment to STOWF Preimage

Since the CKEM construction of Section 2.1 is based on SPHF, it is efficient for language $\mathcal{L}_{pk}$ of "encryptions of a leakage function applied to the pre-image of a tight one-way function," defined in equation (1), if PKE Enc and STOWF $(f, \hat{f})$ are instantiated so that $\mathcal{L}_{pk}$ has an efficient SPHF. Recall that there are

efficient SPHF's for "linear function" languages, i.e., languages whose relation can be expressed as

$$\mathcal{R}[\mathcal{L}] = \{(x, w) \text{ s.t. } x = (C, M) \text{ and } C = w \cdot M\}$$

where C, M are resp. vector and matrix of elements of $\mathbb{G}$, w is a vector of integers, and product $w \cdot M$ denotes an exponentiation, e.g. if $w = [\alpha_1, \ldots, \alpha_n]$ and $M = [g_1, \ldots, g_n]^T$ then $w \cdot M = \prod_{i=1}^{n} g_i^{\alpha_i}$. If C and M are resp. $1 \times m$ and $n \times m$ matrices in $\mathbb{G}$ then the following algorithms form an SPHF for $\mathcal{L}$:

$\mathsf{Hash}(x; hk)$ for $x = (C, M)$ and $hk \leftarrow_{\mathrm{R}} (\mathbb{Z}_p)^m$ outputs $(v, hp) = (C \cdot hk, M \cdot hk)$.

$\mathsf{PHash}(w, hp)$ outputs $v = w \cdot hp$.

Correctness follows because if $C = w \cdot M$ then $w \cdot hp = w \cdot (M \cdot hk) = (w \cdot M) \cdot hk = C \cdot hk$, smoothness because $(C, M) \notin \mathcal{L}$ if and only if C is not in the row span of M, in which case $v = C \cdot hk$ is independent of $hp = M \cdot hk$, and statement privacy holds if for every (C, M) in universe $\mathcal{U}$ matrix M has full row rank.

CKEM for ElGamal Encryption and Boneh-Boyen Function. If f_s, $\hat{f}$, and Enc are defined as $f_s(h) = g^{1/(s+h)}$, $\hat{f}(h) = (g')^h$, and $\mathsf{Enc}_y(m; r) = (g^r, y^r m)$ then language $\mathcal{L}_{pk}$ in equation (1) is an example of a linear function language which admits an SPHF defined above. Note that if $z = g^{1/(s+h)}$ then $z^h = gz^{-s}$, and therefore in this instantiation language $\mathcal{L}_{pk}$ becomes:

$$\mathcal{L}_{pk} = \{(s, z, c, d) \mid \exists\ w = (h, r) \text{ s.t. } (gz^{-s}, c, d) = w \cdot \begin{bmatrix} z & 1 & g \\ 1 & g' & y \end{bmatrix} \qquad (2)$$

Note on shared group setting. Note that the Boneh-Boyen function parameters are $\pi = (\mathbb{G}, p, g, g')$ and the ElGamal public key is $pk = (\mathbb{G}, p, g, y)$. The two schemes share group setting $(\mathbb{G}, p)$, but note that prime-order groups are typically standardized and re-used across many cryptosystems. All group elements g, g', y in the CRS are chosen at random, because the unforgeability of the Boneh-Boyen function assumes that base g is a random group element and the leakage-function hiding property of STOWF assumes that base g' is another random group element. Note that while typically ElGamal encryption is defined for a fixed group generator g, under the DDH assumption on $\mathbb{G}$ it can be instantiated instaed with a random generator g.

Efficient CKEM for $\mathcal{L}_{pk}$. The generic CKEM construction of Section 2.1 instantiated with the linear-language SPHF for $\mathcal{L}_{pk}$ results in the following CKEM procedures $(\mathsf{Snd}, \mathsf{Rec})$ for $\pi = (\mathbb{G}, p, g, g', y)$, hash functions $\mathsf{H_0} : \{0,1\}^* \to (\{0,1\}^\kappa)^2$ and $\mathsf{H_1} : \{0,1\}^* \to \{0,1\}^\kappa$, and IND-SKE encryption scheme (E, D):

$\mathsf{Snd}_\pi(\ell, x)$ for $x = (s, z, c, d)$:

1. Set $(v, hp_1, hp_2) \leftarrow ((gz^{-s})^\alpha c^\beta d^\gamma, z^\alpha g^\gamma, (g')^\beta y^\gamma)$ for $(\alpha, \beta, \gamma) \leftarrow_{\mathrm{R}} \mathbb{Z}_p^3$;
2. Set $hp \leftarrow (hp_1, hp_2)$ and $hk \leftarrow (\alpha, \beta, \gamma)$;
3. Compute $(ek, ck) \leftarrow \mathsf{H_0}(v)$, $e \leftarrow \mathsf{E}_{ek}(hk, x)$, and $\tau \leftarrow \mathsf{H_1}(v, hp, e, \ell)$;
4. Output (ck, m) for $m = (hp, e, \tau)$.

$\mathsf{Rec}_\pi(\ell, w, m)$ for $w = (h, r)$ and $m = (hp, e, \tau)$:

1. Compute $v \leftarrow hp_1^h \cdot hp_2^r$ where $hp = (hp_1, hp_2)$;
2. Set $(ek, ck) \leftarrow \mathsf{H}_0(v)$, $(hk, x) \leftarrow \mathsf{D}_{ek}(e)$;
3. Parse $(\alpha, \beta, \gamma) \leftarrow hk$ and $(s, z, c, d) \leftarrow x$, and set $\tau' \leftarrow \mathsf{H}_1(v, hp, e, \ell)$;
4. Output (ck, x) if $\tau' = \tau$, $(v, hp_1, hp_2) = ((gz^{-s})^\alpha c^\beta d^\gamma, z^\alpha g^\gamma, (g')^\beta y^\gamma)$ and $(gz^{-s}, c, d) = (z^h, (g')^r, g^h y^r)$;
   Otherwise output $(ck, \perp)$ for $ck \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.

Note that Step 4 of Rec, which validates that $(v, hp) = \mathsf{Hash}_{pk}(x; hk)$ and $(x, w) \in \mathcal{R}[\mathcal{L}]$, involves a *verification* of six multiexponentiation equations, rather than their recomputation. Batch verification techniques, e.g. [2], allow this to be done with a single multi-exponentiation. However, using fixed-base exponentiations instead is likely to be more efficient, because when this CKEM is used in the context of the saPAKE-BB protocol shown in Figure 4 the client who runs CKEM Rec algorithm already knows that elements (c, d) are formed correctly, and it will know the representation of (c, d) in bases (g, g', y). Likewise after verifying that $z = g^{(1/(s+h))}$ for $h = hw_{\mathsf{C}}$, base z can be replaced by base g in the verification equations for (v, hp_1, hp_2), hence all these values can be verified by the client using at most 7 fixed-base exponentiations. (We note that these costs can be reduced further if some of the base elements g, g', y are combined, i.e. if g' is equated with either g or y, but we leave the verification of security of such variants to future work.)

5.2 Communication and Computation Costs of Protocol saPAKE-BB

Protocol saPAKE-BB uses only 2 message flows whose total bandwidth is 7 group elements and 2κ additional bits: c, d in flow1 and hp_1, hp_2 in flow2, as well as z, c, d encrypted in e and κ bits each in salt s and hash τ. It is easy to see, however, that (c, d) do not need to be included in the server's ciphertext e, and that they can instead be added to the inputs of hash τ. This optimized protocol would thus take 5 group elements plus 2κ bits, which e.g., on EC-224 comes to only $1120 + 320 = 1440$ bits.

The client's verification in the last step that $(v, hp_1, hp_2) = \mathsf{Hash}_{pk}(x; hk)$ and that statement (s, z, c', d') extracted from the CKEM message m corresponds to the client's witness (hw_{C}, r), can be implemented with a single multi-exponentiation, but as explained in the previous subsection, it can also be implemented with seven fixed-base exponentiation. The total computational cost will therefore be dominated by **1** variable-base multi-exps and **10** fixed-base exps for the client, and **2** variable base multi-exps and **2** fixed-base exps for the server.

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Advances in Cryptology – EUROCRYPT 2001. pp. 119–135. Springer (2001)

2. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Advances in Cryptology – EUROCRYPT 1998. pp. 236–250. Springer (1998)
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Advances in Cryptology – EUROCRYPT 2000. pp. 139–155. Springer (2000)
4. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: IEEE Symposium on Security and Privacy – S&P 1992. pp. 72–84. IEEE (1992)
5. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In: ACM Conference on Computer and Communications Security – CCS 1993. pp. 244–250. ACM (1993)
6. Benhamouda, F., Couteau, G., Pointcheval, D., Wee, H.: Implicit zero-knowledge arguments and applications to the malicious setting. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 9216, pp. 107–129. Springer (2015)
7. Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. IACR Cryptology ePrint Archive 2013, 833 (2013)
8. Blake, I.F., Kolesnikov, V.: Strong conditional oblivious transfer and computing on intervals. In: Advances in Cryptology – ASIACRYPT 2004. pp. 515–529. Springer (2004)
9. Boneh, D., Boyen, X.: Efficient selective-id secure identity based encryption without random oracles. In: Advances in Cryptology – EUROCRYPT 2004. pp. 223–238. Springer (2004)
10. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptology 21(2), 149–177 (2008), https://doi.org/10.1007/s00145-007-9005-7
11. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Advances in Cryptology – EUROCRYPT 2000. pp. 156–171. Springer (2000)
12. Bradley, T., Camenisch, J., Jarecki, S., Lehmann, A., Neven, G., Xu, J.: Password-authenticated public-key encryption. In: ACNS. Lecture Notes in Computer Science, vol. 11464, pp. 442–462. Springer (2019)
13. Bradley, T., Jarecki, S., Xu, J.: Strong asymmetric PAKE based on trapdoor CKEM. IACR Cryptology ePrint Archive (2019), https://eprint.iacr.org/2019/
14. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Advances in Cryptology – EUROCRYPT 2005. pp. 404–421. Springer (2005)
15. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Advances in Cryptology - EUROCRYPT 2002. pp. 45–64. Springer (2002)
16. Di Crescenzo, G.: Conditional oblivious transfer and timed-release encryption. In: Financial Cryptography – FC 2000. pp. 74–89. Springer (2000)
17. Di Crescenzo, G., Ostrovsky, R., Rajagopalan, S.: Conditional oblivious transfer and timed-release encryption. In: Advances in Cryptology – EUROCRYPT 1999. pp. 74–89. Springer (1999)
18. Fujisaki, F., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Advances in Cryptology – CRYPTO 1999. pp. 537–554. Springer (1999)

19. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: ACM Conference on Computer and Communications Security – CCS 2013. pp. 467–476. ACM (2013)
20. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Advances in Cryptology – EUROCRYPT 2003. pp. 524–543. Springer (2003)
21. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Advances in Cryptology – CRYPTO 2006. pp. 142–159. Springer (2006)
22. Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Shin, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: Security and Cryptography for Networks – SCN 2018. pp. 485–504. Springer (2018)
23. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In: IEEE European Symposium on Security and Privacy – EuroS&P 2016. pp. 276–291. IEEE (2016)
24. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Advances in Cryptology – EUROCRYPT 2018 (2018)
25. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. IACR Cryptology ePrint Archive 2018, 163 (2018)
26. Jutla, C.S., Roy, A.: Smooth NIZK arguments. In: Theory of Cryptography – TCC 2018. pp. 235–262. Springer (2018)
27. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Advances in Cryptology – EUROCRYPT 2001. pp. 475–494. Springer (2001)
28. Laur, S., Lipmaa, H.: A new protocol for conditional disclosure of secrets and its applications. In: Applied Cryptography and Network Security – ACNS 2007. pp. 207–225. Springer (2007)
29. MacKenzie, P.D.: More efficient password-authenticated key exchange. In: Topics in Cryptology – CT-RSA 2001. pp. 361–377. Springer (2001)
30. MacKenzie, P.D., Patel, S., Swaminathan, R.: Password-authenticated key exchange based on RSA. In: Advances in Cryptology – ASIACRYPT 2000. pp. 599–613. Springer (2000)
31. Schnorr, C.P.: Small generic hardcore subsets for the discrete logarithm: Short secret DL-keys. Information Processing Letters 79(2), 93–98 (2001)