OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks *

Stanislaw Jarecki[1], Hugo Krawczyk[2], and Jiayu Xu[1]

[1] University of California, Irvine. Email: {stasio@ics.,jiayux@}uci.edu.
[2] IBM Research. Email: hugo@ee.technion.ac.il.

Abstract. Password-Authenticated Key Exchange (PAKE) protocols allow two parties that only share a password to establish a shared key in a way that is immune to offline attacks. *Asymmetric* PAKE (aPAKE) strengthens this notion for the more common client-server setting where the server stores a mapping of the password and security is required even upon server compromise, that is, the only allowed attack in this case is an (inevitable) offline exhaustive dictionary attack against individual user passwords. Unfortunately, current aPAKE protocols (that dispense with the use of servers' public keys) allow for *pre-computation attacks* that lead to the *instantaneous compromise* of user passwords upon server compromise, thus forgoing much of the intended aPAKE security. Indeed, these protocols use – in essential ways – deterministic password mappings or use random "salt" transmitted *in the clear* from servers to users, and thus are vulnerable to pre-computation attacks.

We initiate the study of *Strong aPAKE* protocols that are secure as aPAKE's but *are also secure against pre-computation attacks.* We formalize this notion in the Universally Composable (UC) settings and present two modular constructions using an Oblivious PRF as a main tool. The first builds a Strong aPAKE from *any* aPAKE (which in turn can be constructed from any PAKE [23]) while the second builds a Strong aPAKE from *any* authenticated key-exchange protocol secure against reverse impersonation (a.k.a. KCI). Using the latter transformation, we show *a practical instantiation of a UC-secure Strong aPAKE* in the Random Oracle model. The protocol ("OPAQUE") consists of 2 messages (3 with mutual authentication), requires 3 and 4 exponentiations for server and client, respectively (2 to 4 of which can be fixed-base depending on optimizations), provides forward secrecy, is PKI-free, supports user-side hash iterations, has a built-in facility for password-based storage and retrieval of secrets and credentials, and accommodates a user-transparent server-side threshold implementation.

1 Introduction

Passwords constitute the most ubiquitous form of authentication in the Internet, from the most mundane to the most sensitive applications. The almost universal

* An abridged version of this paper appears in Eurocrypt 2018.

password authentication method in practice relies on TLS/SSL and consists of the user sending its password to the server under the protection of a client-to-server confidential TLS channel. At the server, the password is decrypted and verified against a one-way image typically computed via hash iterations applied to the password and a random "salt" value. Both the password image and salt are stored for each user in a so-called "password file." In this way, an attacker who succeeds in stealing the password file is forced to run an exhaustive *offline dictionary attack* to find users' passwords given a set ("dictionary") of candidate passwords. The two obvious disadvantages of this approach are: (i) the password appears in cleartext at the server during login; and (ii) security breaks if the TLS channel is established with a compromised server's public key (a major concern given today's too-common PKI failures[3]).

Password protocols have been extensively studied in the crypto literature – including in the above client-server setting where the user is assumed to possess an authentic copy of the server's public key [24,25], but the main focus has been on *password-only* protocols where the user does not need to rely on any outside keying material (such as public keys). The basic setting considers two parties that share the same low-entropy password with the goal of establishing shared session keys secure against *offline dictionary attacks*, namely, against an active attacker that possesses a small dictionary from which the password has been chosen. The only viable option for the attacker should be the inevitable *online* impersonation attack with guessed passwords. Such model, known as *password-authenticated key exchange (PAKE)*, was first studied by Bellovin and Merritt [5] and later formalized by Bellare et al. [4] in the game-based indistinguishability approach. Canetti et al. [12] formalized PAKE in the Universally Composable (UC) framework [11], which better captures PAKE security issues such as the use of arbitrary password distributions, the inputting of wrong passwords by the user, and the common use in practice of related passwords for different services.

Whereas the cryptographic literature on PAKE's focuses on the above basic setting, in practice the much more common application of password protocols is in the client-server setting. However, sharing the same password between user and server would mean that a break to the server leaks plaintext passwords for all its users. Thus, what's needed is that upon a server compromise, and the stealing of the password file, an attacker is forced to perform an exhaustive offline dictionary attack as in the above TLS scenario. No other attack, except for an inevitable online guessing attack, should be feasible. In particular, *the two main shortcomings of password-over-TLS* mentioned earlier - reliance on public keys and exposure of the password to the server - *need to be eliminated.* This setting, known as *aPAKE*, for *asymmetric PAKE* (also called *augmented* or *verifier-based*), was introduced by Bellovin and Merrit [6], later formalized in

[3] PKI failures include stealing of server private keys, software that does not verify certificates correctly, users that accept invalid or suspicious certificates, certificates issued by rogue CAs, servers that share their TLS keys with others – e.g., CDN providers or security monitoring software, information (including passwords) that traverses networks in plaintext form after TLS termination; and more.

the simulation-based approach by Boyko et al. [10], and in the UC framework by Gentry et al. [23]. Early protocols proven in the simulation-based model include [10,37,38]. Later, Gentry et al. [23] presented a compiler that transforms any UC-PAKE protocol into a UC-aPAKE (adding an extra round of communication and a client's signature). This was followed by [29] who show the first simultaneous one-round adaptive UC-aPAKE protocol. In addition, several aPAKE protocols targeting practicality have been proposed, most with ad-hoc security arguments, and some have been (and are being) considered for standardization (see below).

A common deficiency of all these aPAKE protocols, including those being proposed for practical use and regardless of their underlying formalism, is that they are *all vulnerable to pre-computation attacks*. Namely, the attacker $\mathcal{A}$ can *pre-compute* a table of values based on a passwords dictionary D, so as soon as $\mathcal{A}$ succeeds in compromising a server it can *instantly* find a user's password. This weakens the benefits of security against server compromise that motivate the aPAKE notion in the first place. Moreover, while current definitions require that the attacker cannot exploit a server compromise without incurring a workload proportional to the dictionary size $|D|$, these definitions allow all this workload to be spent *before* the actual server compromise happens. Indeed, this weakening in the existing aPAKE security definition [23] is needed to accommodate aPAKE protocols that store a one-way *deterministic* mapping of the user's password at the server, say $H(\mathsf{pw})$. Such protocols trivially fall to a pre-computation attack as the attacker $\mathcal{A}$ can build a table of $(H(\mathsf{pw}), \mathsf{pw})$ pairs for all $\mathsf{pw} \in D$, and once it compromises the server, it finds the value $H(\mathsf{pw})$ associated with a user and immediately, in $\log(|D|)$ time, finds that user's password. Such devastating attack can be mitigated by "personalizing" the password map, e.g., hashing the password together with the user id. This forces $\mathcal{A}$ to pre-compute separate tables for individual users, yet all this effort can still be spent before the actual server compromise.

Note that the standard password-over-TLS scheme prevents pre-computation by hashing passwords with a random salt visible to the server only. In contrast, existing aPAKE protocols that do not rely on PKI, either don't use salt or if they do, the salt is transmitted from server to user during login *in the clear*[4]. Given that password stealing via server compromise is the main avenue for collecting billions of passwords by attackers, the above vulnerability of existing aPAKE protocols to pre-computation attacks is a serious flaw (particularly applicable to targeted attacks), and in this aspect password-over-TLS is more secure than all known aPAKE schemes.

Our contributions

We initiate the study of *Strong aPAKE (SaPAKE)* protocols that strengthen the aPAKE security notion by *disallowing pre-computation attacks.* We formalize

[4] Note that even if the aPAKE protocol runs over TLS, the transmitted salt is open to a straightforward active attack. An additional weakness introduced by the use of public salt is enabling "username enumeration attacks" that help attackers identify users and targets – see [16].

this notion in the Universally Composable (UC) model by modifying the aPAKE functionality from [23] to eliminate an adversarial action which allowed such pre-computation attacks. As we explain above, allowing pre-computation attacks was indeed necessary to model the security of existing aPAKE protocols.

The next contribution is building Strong aPAKE (SaPAKE) protocols. For this we present two generic constructions. The first builds the SaPAKE protocol from any aPAKE protocol (namely one that satisfies the original definition from [23]) so that one can "salvage" existing aPAKE protocols. To do so we resort to Oblivious PRF (OPRF) functions [22, 27], namely, a PRF with an associated two-party protocol that in our case is run between a server S that stores a PRF key k and a user U with a password pw. At the end of the interaction, U learns the PRF output $F_k(\mathsf{pw})$ and S learns nothing (in particular, nothing about pw). We show that by preceding any aPAKE protocol with an OPRF interaction in which U computes the value $\mathsf{rw} = F_k(\mathsf{pw})$ with the help of S and uses rw as the password in the aPAKE protocol, one obtains a Strong aPAKE protocol. We show that if the OPRF and the given aPAKE protocol are, respectively, UC realizations of the OPRF functionality (defined in [27]) and the original aPAKE functionality from [23], the resultant scheme realizes our UC functionality $\mathcal{F}_{\mathsf{SaPAKE}}$.

Our second transformation consists of the composition of an OPRF as above with a regular authenticated key exchange protocol AKE. We require UC security for the AKE protocol as well as a property known as resistance to KCI attacks. The latter means that an attacker that learns the secret keys of one party P, but does not actively control P, cannot use this information to impersonate another party P' to P. KCI resistance is a common property of most AKE protocols. In our SaPAKE construction, U first runs the OPRF with S to compute $\mathsf{rw} = F_k(\mathsf{pw})$; then it runs the AKE protocol with S using a private key stored, encrypted using an *authenticated* encryption under rw, at S who sends it to U. Crucial to the security of the protocol is the use of authenticated encryption with a "random-key robustness" property, which is achieved naturally by some schemes or otherwise can be easily ensured, e.g., by adding an HMAC to a symmetric encryption scheme. Under these conditions we show that the composed scheme realizes our UC functionality $\mathcal{F}_{\mathsf{SaPAKE}}$.

Next, we use the above second transformation to instantiate a Strong aPAKE protocol with a very efficient OPRF and any efficient AKE with the KCI property. The OPRF scheme we use, essentially a Chaum-type blinded DH computation, has been proven UC-secure by Jarecki et al. [26, 27]. We show that this OPRF scheme, which we call DH-OPRF(called 2HashDH in [26, 27]), remains secure in spite of changes to the OPRF functionality that we introduce for supporting a stronger OPRF notion needed in our setting. We call the result of this instantiation, the *OPAQUE protocol.*

OPAQUE combines the best properties of existing aPAKE protocols and of the standard password-over-TLS. As any aPAKE-secure protocol, it offers two fundamental advantages over the TLS-based solution: It does not rely on PKI and the plaintext password is never in the clear at the server. The only way for an attacker that observes (or actively controls) a session at a server to

learn the password is via an exhaustive offline dictionary attack. Watching or participating in a session with the user does not help the attacker. At the same time, OPAQUE resolves the major flaw of existing aPAKE protocols relative to password-over-TLS, namely, their vulnerability to pre-computation attacks.

In addition to the above fundamental properties, OPAQUE enjoys important properties for use in practice. Its modularity allows for its use with different key-exchange schemes that can provide different features and performance tradeoffs. When implemented with a 2-message implicit-authentication KE protocol (e.g., HMQV [33]), OPAQUE takes only 2 messages (or 3 with mutual explicit authentication). The computational cost (using the DH-OPRF scheme from Section A) is one exponentiation for the server and two for the client[5] in addition to the KE protocol cost (with HMQV, this cost is 2.17 exponentiations per party). OPAQUE offers forward secrecy (a particularly crucial property for password protocols) if the KE does. OPAQUE further supports password hardening for increasing the cost of offline dictionary attacks (upon server compromise) through user-side iterated hashing without the need to transmit salt from S to U. In Figure 9 in Section 6 we show an instantiation of OPAQUE in the RO model with HMQV as the AKE.

Compared to the practical aPAKE protocols that have been and are being considered for standardization (cf., [1,42]), OPAQUE fares clearly better on the security side as the only scheme that offers resistance to pre-computation attacks. Performance-wise, OPAQUE is competitive with the more efficient among these protocols (see Sec. 6). OPAQUE also provides a unique functionality among aPAKE protocols in that it allows to *store and retrieve user's secrets* such as a bitcoin wallet, authentication credentials, encrypted backup keys, etc., thus offering a far more secure alternative to the practice of deriving low-entropy secrets directly from a user's password. Furthermore, OPAQUE allows for a user-transparent server-side threshold implementation [28] where the only exposure of the user password - or any stored secrets - is in case a threshold of servers is compromised and even then a full dictionary attack is required.

Finally, we comment that while OPAQUE can completely replace password authentication in TLS, it can also be used in conjunction with TLS for protecting account information, for bootstrapping TLS client authentication (via an OPAQUE-retrieved client signing key), or as an hedge against PKI failures. In other words, while we are accustomed to use TLS to protect passwords, OPAQUE can be used to protect TLS. We expand on this aspect in Section 6.2.

We stress that variants of OPAQUE have been studied in prior work in several settings but none of these works presents a formal analysis of the protocol as an aPAKE, let alone as a Strong aPAKE, a notion that we introduce here for the first time. While our treatment frames OPAQUE in the context of Oblivious PRFs [26, 27], its design can be seen as an instantiation of the Ford-Kaliski paradigm for password hardening and credential retrieval using Chaum's blinded

[5] A variant of the protocol discussed in Section 6.3 allows one or both of the client's exponentiations to be fixed-base and offline.

exponentiation. Boyen [9] specifies and studies the protocol (called HPAKE) in the setting of client-side *halting KDF* [8]. Jarecki et al. [26,27] study a threshold version (also using the OPRF abstraction) in the context of *password-protected secret sharing (PPSS)* protocols. Because of the relation between PPSS and Threshold PAKE protocols [26], this analysis implies security of OPAQUE as a PAKE protocol in the BPR model [4] but not as an aPAKE (let alone as a strong aPAKE).

2 The Strong aPAKE Functionality

We present the ideal UC Strong aPAKE functionality, $\mathcal{F}_{\mathsf{SaPAKE}}$, that will serve as our definition of Strong aPAKE security; namely, we call a protocol a secure *Strong aPAKE* if it realizes $\mathcal{F}_{\mathsf{SaPAKE}}$. Functionality $\mathcal{F}_{\mathsf{SaPAKE}}$ is a simple but significant variant of the UC aPAKE functionality $\mathcal{F}_{\mathsf{aPAKE}}$ from [23] (it was denoted $\mathcal{F}_{\mathsf{apwKE}}$ in [23]) which we recall in Fig. 1.

The aPAKE functionality of [23] is based on the UC PAKE functionality from [12], and it includes extensions needed for taking care of the asymmetric nature of the aPAKE setting. First, in an aPAKE scheme the server and the user run different programs: The user runs an aPAKE session on a password (via command UsrSession) while the server runs it on a "password file" $\mathsf{file}[sid]$ that represents server's user-specific state corresponding to the user's password, e.g., a password hash, which the server creates on input the user's password during aPAKE initialization, via command StorePwdFile. Furthermore, $\mathcal{F}_{\mathsf{aPAKE}}$ models a possible compromise of a server, via command StealPwdFile, from which the attacker obtains $\mathsf{file}[sid]$. Such compromise subsequently allows the attacker to (1) impersonate the server to the user, via command Impersonate, and (2) find the password via an offline dictionary attack, via command OfflineTestPwd. The way functionality $\mathcal{F}_{\mathsf{aPAKE}}$ of [23] handles the offline dictionary attack is the focus of the Strong aPAKE functionality we propose, and we discuss them below.

Strong aPAKE vs. aPAKE. Our functionality $\mathcal{F}_{\mathsf{SaPAKE}}$ is almost identical to $\mathcal{F}_{\mathsf{aPAKE}}$ except that the text with the gray background in Fig. 1 is omitted. That is, the only difference between $\mathcal{F}_{\mathsf{SaPAKE}}$ and $\mathcal{F}_{\mathsf{aPAKE}}$ are in the actions upon the stealing of the password file; specifically, $\mathcal{F}_{\mathsf{SaPAKE}}$ omits recording the $(\text{offline}, \mathsf{pw})$ pairs and does not allow for OfflineTestPwd queries made before the StealPwdFile query. Let us explain. Let's consider first the definition of $\mathcal{F}_{\mathsf{SaPAKE}}$, i.e., with the gray text omitted. In this case, the actions upon server compromise, i.e., StealPwdFile, are simple. First, a flag is defined to mark that the password file has been compromised. Second, once this event happens, the adversary is allowed to submit password guesses and be informed if a guess was correct. Note that each guess "costs" the attacker one OfflineTestPwd query. This together with the restriction that these queries can only be made after the password file is compromised ensure that shortcuts in finding the password after such compromise are not possible, namely that

In the description below, we assume $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$.

Password Registration

- On (StorePwdFile, $sid, \mathsf{U}, \mathsf{pw}$) from S, if this is the first StorePwdFile message, record $\langle \text{file}, \mathsf{U}, \mathsf{S}, \mathsf{pw} \rangle$ and mark it uncompromised.

Stealing Password Data

- On (StealPwdFile, sid) from $\mathcal{A}^*$, if there is no record $\langle \text{file}, \mathsf{U}, \mathsf{S}, \mathsf{pw} \rangle$, return "no password file" to $\mathcal{A}^*$. Otherwise, if the record is marked uncompromised, mark it compromised; regardless,
  - If there is a record (offline, pw), send pw to $\mathcal{A}^*$.
  - Else Return "password file stolen" to $\mathcal{A}^*$.
- On (OfflineTestPwd, sid, pw^*) from $\mathcal{A}^*$, do:
  - If there is a record $\langle \text{file}, \mathsf{U}, \mathsf{S}, \mathsf{pw} \rangle$ marked compromised, do: if $\mathsf{pw}^* = \mathsf{pw}$, return "correct guess" to $\mathcal{A}^*$; else return "wrong guess."
  - Else record (offline, pw).

Password Authentication

- On (UsrSession, $sid, ssid, \mathsf{S}, \mathsf{pw}'$) from U, send (UsrSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) to $\mathcal{A}^*$. Also, if this is the first UsrSession message for $ssid$, record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{pw}' \rangle$ and mark it fresh.
- On (SvrSession, $sid, ssid$) from S, retrieve $\langle \text{file}, \mathsf{U}, \mathsf{S}, \mathsf{pw} \rangle$, and send (SvrSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) to $\mathcal{A}^*$. Also, if this is the first SvrSession message for $ssid$, record $\langle ssid, \mathsf{S}, \mathsf{U}, \mathsf{pw} \rangle$ and mark it fresh.

Active Session Attacks

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

Key Generation and Authentication

- On (NewKey, $sid, ssid, \mathsf{P}, SK$) from $\mathcal{A}^*$ where $|SK| = \tau$, if there is a record $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw}' \rangle$ not marked completed, do:
  - If the record is marked compromised, or either P or P' is corrupted, send $(sid, ssid, SK)$ to P.
  - If the record is marked fresh, a $(sid, ssid, SK')$ tuple was sent to P', and at that time there was a record $\langle ssid, \mathsf{P}', \mathsf{P}, \mathsf{pw}' \rangle$ marked fresh, send $(sid, ssid, SK')$ to P.
  - Else pick $SK'' \leftarrow_{\text{R}} \{0,1\}^\tau$ and send $(sid, ssid, SK'')$ to P.

  Finally, mark $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw}' \rangle$ completed.
- On (TestAbort, $sid, ssid, \mathsf{P}$) from $\mathcal{A}^*$, if there is a record $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw}' \rangle$ not marked completed, do:
  - If it is marked fresh and record $\langle ssid, \mathsf{P}', \mathsf{P}, \mathsf{pw}' \rangle$ exists, send Succ to $\mathcal{A}^*$.
  - Else send Fail to $\mathcal{A}^*$ and (abort, $sid, ssid$) to P, and mark $\langle ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw}' \rangle$ completed.

Fig. 1: Functionalities $\mathcal{F}_{\mathsf{aPAKE}}$ (full text) and $\mathcal{F}_{\mathsf{SaPAKE}}$ (shadowed text omitted)

the attacker needs to pay with one OfflineTestPwd query for each password it wants to test. Thus, pre-computation attacks are made infeasible.

Now, consider the $\mathcal{F}_{\mathsf{aPAKE}}$ functionality from [23] which includes the text in gray too. This functionality allows the attacker, via (offline, pw) records, to make guess queries against the password even before the password file is compromised. The restriction is that the responses to whether a guess was correct or not are provided to the attacker only after a StealPwdFile event. But note that if one of these guesses was correct, the attacker learns it *immediately* upon server compromise. This provision was necessary in [23] because the file[*sid*] in their aPAKE construction contains a deterministic publicly-computable hash of the password, thus allowing for a pre-computation attack which lets the adversary instantaneously identify the password with a single table lookup upon server compromise. Indeed, one can think of the pairs (offline, pw) in the original $\mathcal{F}_{\mathsf{aPAKE}}$ functionality as a pre-computed table that the attacker builds overtime and which it can use to identify the password as soon as the server is compromised. By eliminating the ability to get guesses (offline, pw) answered before server compromise in our $\mathcal{F}_{\mathsf{SaPAKE}}$ functionality, we make such pre-computation attacks infeasible in the case of a Strong aPAKE.

Modeling Server Compromise and Offline Dictionary Queries. As in [23], we specify that StealPwdFile and OfflineTestPwd messages from $\mathcal{A}^*$ to $\mathcal{F}_{\mathsf{SaPAKE}}$ are accounted for by the environment. This is consistent with the UC treatment of adaptive corruption queries and is crucial to our modeling. Note that if the environment does not observe adaptive corruption queries then the ideal model adversary, i.e., the simulator, could immediately corrupt all parties at the beginning of the protocol, learning their private inputs and thus making the work of simulation easier. By making the player-corruption queries, modeled by StealPwdFile command in our context, observable by the environment, we ensure that the environment's view of both the ideal and the real execution includes the same player-corruption events. This way we keep the simulator "honest," because it can only corrupt a party if the environment accounts for it.

The same concern pertains to offline dictionary queries OfflineTestPwd, because if they were not observable by the environment, the ideal adversary could make such queries even if the real adversary does not. In particular, without environmental accounting for these queries the $\mathcal{F}_{\mathsf{aPAKE}}$ and $\mathcal{F}_{\mathsf{SaPAKE}}$ functionalities would be equivalent because the simulator could internally gather all the offline dictionary attack queries made by the real-world adversary before server corruption, and it would send them all via the OfflineTestPwd query to $\mathcal{F}_{\mathsf{SaPAKE}}$ after server corruption via the StealPwdFile query. Such simulator would make the ideal-world view indistinguishable from the real-world view to the environment *if* the environment does not observe the sequence of OfflineTestPwd and StealPwdFile queries.

Finally, we note that the functionality $\mathcal{F}_{\mathsf{SaPAKE}}$, like $\mathcal{F}_{\mathsf{aPAKE}}$, has effectively two separate notions of a server corruption. Formally, it considers a *static*

adversarial model where all entities, including users and servers, are either honest or corrupt throughout the life-time of the scheme. In addition, it allows for an *adaptive* server compromise of an honest server, via the StealPwdFile, which leaks to the adversary the server's private state corresponding to a particular password file, but it does not give the adversary full control over the server's entity. In particular, the accounts on the same server for which the adversary does not explicitly issue the StealPwdFile command must remain unaffected. We adopt this convention from [23] and we call a server "corrupted" if it is (statically) corrupt and adversarially controlled, and we call an aPAKE instance "compromised" if the adversary steals its password file from the server.

Non-black-box Assumptions. Note that the aPAKE functionality requires the simulator, playing the role of the ideal-model adversary, to detect offline password guesses made by the real-world adversary. As pointed out by [23], this seems to require a non-black-box hardness assumption on some cryptographic primitive, e.g., the Random Oracle Model (ROM), which would allow the simulator to extract a password guess from adversary's *local* computation, e.g., a local execution of aPAKE interaction on a password guess and a stolen password file.

Server Initialization. We note that while $\mathcal{F}_{\mathsf{aPAKE}}$ defines password registration as an internal action of server S, with the user's password as a local input, one can modify it to support an interactive procedure between user and server, e.g., to prevent S from ever learning the plaintext password. To that end one needs to assume that during the Password Registration phase there is an *authenticated channel* from server to user, so the user can verify that it is registering the password with the correct server. (Functionality $\mathcal{F}_{\mathsf{aPAKE}}$ effectively also assumes such authenticated channel because otherwise the user's password cannot be safely transported to S.) In practice, the server also needs to verify the user's identity, and the password file could be created by the user and transported to the server. However, this is beyond the scope of the formal aPAKE functionality.

3 Oblivious Pseudorandom Function

Oblivious Pseudorandom Functions (OPRF) are a central tool in all our constructions. An OPRF consists of a pseudorandom function family F with an associated two-party protocol run between a server that holds a key k for F and a user with an input x. At the end of the interaction, the user learns the PRF output $F_k(x)$ and nothing else, and the server learns nothing (in particular, nothing about x). The notion of OPRF was introduced in [22]. The first UC formulation of it was given in [26], including a verifiability property that lets the user check the correct behavior of the server during the OPRF execution. Later [27] gave an alternative UC definition of OPRF which dispensed with the verifiability property, allowing for more efficient instantiations. The main idea in the OPRF formulations of [26, 27] is the use of

Public Parameters: PRF output-length ℓ, polynomial in security parameter τ.

Convention on F and tx: For every (sid, S) value $\mathsf{tx}(sid, \mathsf{S})$ is initially set to 0, and for every x value $F_{sid,\mathsf{S}}(x)$ is initially undefined. Whenever $F_{sid,\mathsf{S}}(x)$ is referenced below and it is undefined, pick $F_{sid,\mathsf{S}}(x) \leftarrow_{\mathrm{R}} \{0,1\}^{\ell}$.

Initialization

- On (INIT, sid, x) from server S, send (INIT, $sid, F_{sid,\mathsf{S}}(x)$) to S.

Server Compromise

- On (COMPROMISE, sid) from $\mathcal{A}^*$ for a server S, mark S COMPROMISED.
- On (OFFLINEEVAL, sid, S, x) from $\mathcal{A}^*$, if S is corrupted or marked COMPROMISED, send (OFFLINEEVAL, $sid, F_{sid,\mathsf{S}}(x)$) to $\mathcal{A}^*$.

Evaluation

- On (EVAL, $sid, ssid, \mathsf{S}, x$) from party $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}^*\}$, record $\langle ssid, \mathsf{P}, \mathsf{S}, x\rangle$ and send (EVAL, $sid, ssid, \mathsf{P}, \mathsf{S}$) to $\mathcal{A}^*$.
- On (SNDRCOMPLETE, $sid, ssid$) from S, send (SNDRCOMPLETE, $sid, ssid, \mathsf{S}$) to $\mathcal{A}^*$. If this is the first SNDRCOMPLETE message for $ssid$, set $\mathsf{tx}(sid, \mathsf{S})$++.
- On (RCVCOMPLETE, $sid, ssid, \mathsf{S}^*$) from $\mathcal{A}^*$, retrieve $\langle ssid, \mathsf{P}, \mathsf{S}, x\rangle$ (where $\mathsf{P} \in \{\mathsf{U}, \mathcal{A}^*\}$); abort if (i) such record does not exist, or (ii) S is honest and not marked COMPROMISED and $\mathsf{S}^* \neq \mathsf{S}$, or (iii) $\mathsf{tx}(sid, \mathsf{S}^*) = 0$. Otherwise set $\mathsf{tx}(sid, \mathsf{S}^*)$−− and send (EVAL, $sid, ssid, F_{sid,\mathsf{S}^*}(x)$) to P.

Fig. 2: Functionality $\mathcal{F}_{\mathsf{OPRF}}$ with Adaptive Compromise

a *ticketing mechanism* $\mathsf{tx}(\cdot)$ that ensures that the number of input values on which anyone can compute the OPRF on a key held by an honest server S is no more than the number of executions of the OPRF recorded by S. This mechanism dispenses with the need to extract users' inputs as is typically needed in UC simulations and it leads to much more efficient OPRF instantiations.

Here we adopt the formulation from [27] as the basis for our definition of functionality $\mathcal{F}_{\mathsf{OPRF}}$ presented in Fig. 2. We refer to [27] for detailed rationale, but we note that it requires PRF outputs to be pseudorandom even to the owner of the PRF key k. This does not seem achievable under non-black-box assumptions, but it is achievable, indeed very efficiently, in the Random Oracle Model (ROM). Note that the reliance on non-black-box assumptions like ROM is called for in the aPAKE context, see Section 2.

Changes from OPRF functionality of [27]. To use UC OPRF in our application(s) we need to make some changes to the way functionality $\mathcal{F}_{\mathsf{OPRF}}$ was defined in [27], as described below. Changes (2), (3) and (4) are essentially syntactic and require only cosmetic changes in the security argument. Change

(1) is the only one which influences the security argument in a more essential way. Fortunately, the DH-OPRF protocol that we use for OPRF instantiation in our protocols, shown in [27] to realize their version of the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$, also realizes our modified $\mathcal{F}_{\mathsf{OPRF}}$ functionality. We recall the DH-OPRF protocol in Figure 11 in Appendix A, adapting its syntax to our changes in $\mathcal{F}_{\mathsf{OPRF}}$, and we argue that the security proof of [27] which shows that it realizes $\mathcal{F}_{\mathsf{OPRF}}$ defined by [27] extends to the modified functionality $\mathcal{F}_{\mathsf{OPRF}}$ presented here.

(1) We extend the OPRF functionality to allow the *adaptive compromise* of a server holding the PRF key via a Compromise message. Such action is needed in the aPAKE setting where the attacker $\mathcal{A}^*$ can compromise a server's password file that contains the server's OPRF key. After the compromise, $\mathcal{A}^*$ is allowed to compute that server's PRF function by itself on any value of its choice using OfflineEval and without the restrictions of the ticketing mechanism. We note that functionality $\mathcal{F}_{\mathsf{OPRF}}$ distinguishes between (statically) *corrupted servers* and (adaptively) *compromised sessions* (the latter representing different OPRF keys at the same server). This distinction allows for a granular separation between compromised and uncompromised OPRF keys held by the same server. We adopt this distinction for consistency with the aPAKE functionality from Fig. 1 that distinguishes between an entirely corrupted server and particular aPAKE instances that can be adaptively compromised by an adversary.

(2) We change the SndrComplete message such that it is sent from S instead of $\mathcal{A}$, thus restricting the number of OPRF invocations per *ssid* to one. This enforces a single password guess per aPAKE sub-session which is crucial for aPAKE security.

(3) We change the session-id syntax used in [27] to model the use of multiple OPRF keys by the same server. In the formulation of [27] each PRF key was identified with a server identity making a one-to-one correspondence between OPRF keys and servers. Here, we allow multiple OPRF keys to be associated with one server. Each such key is identified with a tag *sid* and a server can be associated with multiple such tags. In the context of our application to aPAKE protocols, each aPAKE session is associated with a unique OPRF key used by the server for a particular user, so the session-id *sid* corresponds to a user account at that server. Any *sid* can include *sub-sessions*, denoted by *ssid*, corresponding to different runs of the OPRF protocol between a user and a server.

(4) We add an Initialization phase to the functionality, which models a server picking an OPRF key and, in addition, computing the OPRF value on any input. This interface simplifies the usage of OPRF in our applications to aPAKE, where the server will pick an OPRF key for a new user and evaluate the OPRF on the user's password (for generating an encryption key). This modeling differs from [27] who framed OPRF initialization as an interactive procedure through an Eval call while here it is performed locally by the server.

4 A Compiler from aPAKE to Strong aPAKE via OPRF

In Fig. 3 we specify a compiler that transforms any OPRF and any aPAKE into a Strong aPAKE protocol. In UC terms the Strong aPAKE protocol is defined in the $(\mathcal{F}_{\mathsf{OPRF}}, \mathcal{F}_{\mathsf{aPAKE}})$-hybrid world, for $\mathcal{F}_{\mathsf{OPRF}}$ with the output length parameter $\ell = 2\tau$. The compiler is simple. First, the user transforms its password pw into a randomized value rw by interacting with the server in an OPRF protocol where the user inputs pw and the server inputs the OPRF key. Nothing is learned at the server about pw (i.e., rw is indistinguishable from random as long as the input pw is not queried as input to the OPRF). Next, the user sets rw as its password in the given aPAKE protocol. Note that since the password rw is taken from a pseudorandom set, then even if the size of this set is the same as the original dictionary D from which pw was taken, the pseudorandom set is unknown to the attacker (the attacker can only learn this set via OPRF queries which require an online dictionary attack). Thus, any previous ability to run a pre-computation attack against the aPAKE protocol based on dictionary D is now lost.

We assume that $\mathcal{A}$ always simultaneously sends queries (Compromise, sid) and (StealPwdFile, sid) for the same sid, resp. to $\mathcal{F}_{\mathsf{OPRF}}$ to $\mathcal{F}_{\mathsf{aPAKE}}$, because in any instantiation of this scheme the server's OPRF-related state and aPAKE-related state would be part of the same $\mathsf{file}[sid]$. Consequently, for a single sid, S's status (compromised or not) in $\mathcal{F}_{\mathsf{OPRF}}$ and $\mathcal{F}_{\mathsf{aPAKE}}$ is always the same.

Password Registration

1. On input (StorePwdFile, $sid, \mathsf{U}, \mathsf{pw}$), S sends (Init, sid, pw) to $\mathcal{F}_{\mathsf{OPRF}}$.
   On $\mathcal{F}_{\mathsf{OPRF}}$'s response (Init, sid, rw), S sends (StorePwdFile, $sid, \mathsf{U}, \mathsf{rw}$) to $\mathcal{F}_{\mathsf{aPAKE}}$.

Password Authentication and Key Generation

1. On input (UsrSession, $sid, ssid, \mathsf{S}, \mathsf{pw}'$), U sends (Eval, $sid, ssid, \mathsf{S}, \mathsf{pw}'$) to $\mathcal{F}_{\mathsf{OPRF}}$.
   On $\mathcal{F}_{\mathsf{OPRF}}$'s response (Eval, $sid, ssid, \mathsf{rw}'$), U sends (UsrSession, $sid, ssid, \mathsf{S}, \mathsf{rw}'$) to $\mathcal{F}_{\mathsf{aPAKE}}$.
2. On input (SvrSession, $sid, ssid$), S sends (SndrComplete, $sid, ssid$) to $\mathcal{F}_{\mathsf{OPRF}}$ and (SvrSession, $sid, ssid$) to $\mathcal{F}_{\mathsf{aPAKE}}$.
3. On $(sid, ssid, SK)$ or (abort, $sid, ssid$) from $\mathcal{F}_{\mathsf{aPAKE}}$, the recipient, either U or S, outputs this message.

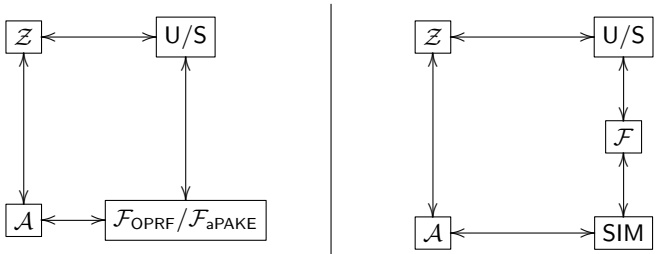
Fig. 3: Strong aPAKE Protocol in the $(\mathcal{F}_{\mathsf{OPRF}}, \mathcal{F}_{\mathsf{aPAKE}})$-Hybrid World

4.1 Proof of Security

Theorem 1. *The protocol in Fig. 3 UC-realizes the* $\mathcal{F}_{\mathsf{SaPAKE}}$ *functionality assuming access to the OPRF functionality* $\mathcal{F}_{\mathsf{OPRF}}$ *and aPAKE functionality* $\mathcal{F}_{\mathsf{aPAKE}}$.

Concretely, for any adversary $\mathcal{A}$ *against the protocol, there is a simulator* SIM *that produces a view in the simulated ideal world (henceforth simulated world) such that the advantage that an environment has in distinguishing between this view and the view in the* $(\mathcal{F}_{\mathsf{OPRF}}, \mathcal{F}_{\mathsf{aPAKE}})$*-hybrid real world (henceforth real world) is at most* $(q_{\mathrm{F}}^2 + 2q_{\mathrm{O}} + 4q_{\mathrm{U}} + 2q_{\mathrm{S}})/2^{2\tau+1}$*, where* q_{F} *is the number of* Eval *and* OfflineEval *messages aimed at* $\mathcal{F}_{\mathsf{OPRF}}$ *from* $\mathcal{A}$*;* q_{O} *is the number of* OfflineTestPwd *messages aimed at* $\mathcal{F}_{\mathsf{aPAKE}}$ *from* $\mathcal{A}$*; and* q_{U} *and* q_{S} *are the numbers of* U *and* S *sub-sessions which* $\mathcal{Z}$ *starts via a* UsrSession *and a* SvrSession *message, respectively. (In the real world,* $\mathcal{A}$ *sends the messages to* $\mathcal{F}_{\mathsf{OPRF}}$ *and* $\mathcal{F}_{\mathsf{aPAKE}}$*. In the simulated world,* $\mathcal{A}$ *sends the messages to* SIM *acting as both* $\mathcal{F}_{\mathsf{OPRF}}$ *and* $\mathcal{F}_{\mathsf{aPAKE}}$*.)*

Proof. For any adversary $\mathcal{A}$, we construct a simulator SIM as in Fig. 4 and Fig. 5. Following [11], without loss of generality, we may assume that $\mathcal{A}$ is a "dummy" adversary that merely passes all its messages and computations to the environment $\mathcal{Z}$. We omit all interactions with corrupted U and S where SIM acts as $\mathcal{F}_{\mathsf{OPRF}}$ and $\mathcal{F}_{\mathsf{aPAKE}}$, since the simulation is trivial (SIM gains all information needed and simply follows the code of $\mathcal{F}_{\mathsf{OPRF}}/\mathcal{F}_{\mathsf{aPAKE}}$). To keep notation brief we denote functionality $\mathcal{F}_{\mathsf{SaPAKE}}$ as $\mathcal{F}$.



(a) real world

(b) simulated world

In order to account for the advantage of the environment $\mathcal{Z}$ in distinguishing between its real world view and its simulated world view, we compare how these two settings respond to environment's commands and derive the distinguishing advantages in cases where the simulation is not perfect. Below we assume that $\mathcal{Z}$ issues the (StorePwdFile, *sid*, U, pw) command to S for some pw; otherwise any subsequent server-side commands of $\mathcal{Z}$ will not have any effect.

For every pair (sid, S), initialize $\mathsf{tx}(sid, \mathsf{S})$ to 0; initialize tested to $\varnothing$.

Stealing Password Data and Offline Queries

1. On $(\textsc{Compromise}, sid)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$ and $(\textsc{StealPwdFile}, sid)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, send $(\textsc{StealPwdFile}, sid)$ to $\mathcal{F}$.
   If $\mathcal{F}$ returns "no password file," pass it to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.
   If $\mathcal{F}$ returns "password file stolen," mark S and $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \cdot \rangle$ $\textsc{compromised}$ (record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \bot \rangle$ and mark it $\textsc{compromised}$ if there is no such record). Furthermore, if $\cdot$ is a string rw and $\mathsf{rw} \in \mathsf{tested}$, then send rw to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$; otherwise send "password file stolen" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.
2. On $(\textsc{OfflineEval}, sid, \mathsf{S}, x)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$, if S is corrupted or marked $\textsc{compromised}$, send $(\textsc{OfflineEval}, sid, F_{sid,\mathsf{S}}(x))$ to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{OPRF}}$ (pick $F_{sid,\mathsf{S}}(x) \leftarrow_{\mathrm{R}} \{0,1\}^{2\tau}$ if it is undefined) and $(\textsc{OfflineTestPwd}, sid, x)$ to $\mathcal{F}$. If $\mathcal{F}$ returns "correct guess," retrieve $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \cdot \rangle$; if the last item is $\bot$, replace it with $F_{sid,\mathsf{S}}(x)$ (or record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, F_{sid,\mathsf{S}}(x) \rangle$ if there is no such record).
3. On message $(\textsc{OfflineTestPwd}, sid, \mathsf{rw}^*)$ from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, add rw^* to tested. Furthermore, if there is a record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \mathsf{rw} \rangle$ marked $\textsc{compromised}$, do:
   - If $\mathsf{rw} = \mathsf{rw}^*$, send "correct guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.
   - Else send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.

Fig. 4: The Simulator SIM in the Stealing Password Data Phase

– "no password file" or rw or "password file stolen" (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends $(\textsc{Compromise}, sid)$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$ and $(\textsc{StealPwdFile}, sid)$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$. $\mathcal{Z}$ receives "no password file" if and only if $\mathcal{Z}$ did not sent a $(\textsc{StorePwdFile}, sid, \mathsf{U}, \mathsf{pw})$ message to S. $\mathcal{Z}$ receives rw if and only if $\mathcal{Z}$ sent a $(\textsc{StorePwdFile}, sid, \mathsf{U}, \mathsf{pw})$ message to S previously, an $(\textsc{OfflineTestPwd}, sid, \mathsf{rw})$ message aimed at $\mathcal{F}_{\mathsf{aPAKE}}$ was sent from $\mathcal{A}$, and $\mathsf{rw} = F_{sid,\mathsf{S}}(\mathsf{pw})$.

– "correct/wrong guess" on $(\textsc{OfflineTestPwd}, sid, \mathsf{rw}^*)$ (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends $(\textsc{OfflineTestPwd}, sid, \mathsf{rw}^*)$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, provided that there is a record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \mathsf{pw} \rangle$ marked $\textsc{compromised}$ in $\mathcal{F}_{\mathsf{aPAKE}}$ (or a record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \cdot \rangle$ marked $\textsc{compromised}$ in SIM). For each such $(\textsc{OfflineTestPwd}, sid, \mathsf{rw}^*)$ message, we have:
  In the real world, $\mathcal{Z}$ receives "correct guess" if and only if $\mathsf{rw}^* = F_{sid,\mathsf{S}}(\mathsf{pw})$.
  In the simulated world, $\mathcal{Z}$ receives "correct guess" if and only if (1) SIM's record contains $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, F_{sid,\mathsf{S}}(\mathsf{pw}) \rangle$, and (2) $\mathsf{rw}^* = F_{sid,\mathsf{S}}(\mathsf{pw})$.
  The message sent to $\mathcal{Z}$ in the two worlds is the same unless in the simulated world, the condition (2) above holds, while (1) does not hold (thus it receives

Password Authentication

1. On (UsrSession, sid, $ssid$, U, S) from $\mathcal{F}$, send (Eval, sid, $ssid$, U, S) to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{OPRF}}$. Also, if this is the first UsrSession message for $ssid$, record $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$ and mark it fresh.
2. On (SvrSession, sid, $ssid$, U, S) from $\mathcal{F}$, if there is no record $\langle \text{file}, \mathsf{U}, \mathsf{S}, \cdot\rangle$, record $\langle \text{file}, \mathsf{U}, \mathsf{S}, \perp\rangle$ and mark it uncompromised; send (SndrComplete, sid, $ssid$, S) and (SvrSession, sid, $ssid$, U, S) to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{OPRF}}$ and $\mathcal{F}_{\mathsf{aPAKE}}$, respectively. Also, if this is the first SvrSession message for $ssid$, set $\mathsf{tx}(sid, \mathsf{S})$++, record $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ and mark it fresh.
3. On (RcvComplete, sid, $ssid$, S^*) from $\mathcal{A}$ as a message to $\mathcal{F}_{\mathsf{OPRF}}$, retrieve $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$; ignore this message if (i) such record does not exist, or (ii) S is honest and not marked compromised and $\mathsf{S}^* \neq \mathsf{S}$, or (iii) $\mathsf{tx}(sid, \mathsf{S}^*) = 0$. Else set $\mathsf{tx}(sid, \mathsf{S})$−−, augment the record to $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*\rangle$, mark it fresh and send (UsrSession, sid, $ssid$, U, S) to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.

Active Session Attacks

1. On (TestPwd, sid, $ssid$, $\mathsf{P}, \mathsf{rw}^*$) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, if there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*\rangle$ (if $\mathsf{P} = \mathsf{U}$) or $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ (if $\mathsf{P} = \mathsf{S}$) marked fresh, check whether there is an x such that $\mathsf{rw}^* = F_{sid,\mathsf{S}^*}(x)$ (if $\mathsf{P} = \mathsf{U}$) or $\mathsf{rw}^* = F_{sid,\mathsf{S}}(x)$ (if $\mathsf{P} = \mathsf{S}$).
   - If there are more than one such x's, output halt and abort.
   - If there is a unique such x, send (TestPwd, sid, $ssid$, P, x) to $\mathcal{F}$.
     If $\mathcal{F}$ returns "correct guess," mark the record compromised and send "correct guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$; if $\mathsf{P} = \mathsf{S}$, also retrieve $\langle \text{file}, \mathsf{U}, \mathsf{S}, \cdot\rangle$, and if the last item is $\perp$, replace it with rw^*.
     If $\mathcal{F}$ returns "wrong guess," mark the record interrupted and send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.
   - If there is no such x, mark the record interrupted and send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.
2. On (Impersonate, sid, $ssid$) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, if there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*\rangle$ marked fresh, do:
   - If $\mathsf{S}^* = \mathsf{S}$, send (Impersonate, sid, $ssid$) to $\mathcal{F}$, and pass $\mathcal{F}$'s response ("correct guess" or "wrong guess") to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.
   - Else send "wrong guess" to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$.

   Also, if the response is "correct guess," mark the record compromised; else mark the record interrupted.

Key Generation and Authentication

1. On (NewKey, sid, $ssid$, P, SK) or (TestAbort, sid, $ssid$, P) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, if there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*\rangle$ (if $\mathsf{P} = \mathsf{U}$) or $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ (if $\mathsf{P} = \mathsf{S}$) not marked completed, pass the message from $\mathcal{A}$ to $\mathcal{F}$. In the case of (TestAbort, sid, $ssid$, P), also pass $\mathcal{F}$'s response (Succ or Fail) to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{aPAKE}}$. Finally, mark the record above completed.

Fig. 5: The Simulator SIM in the Login Phase

"correct guess" in the real world and "wrong guess" in the simulated world). (1) is equivalent to the combination of the following two conditions:

- $\mathcal{A}$ sent (OFFLINEEVAL, sid, pw) aimed at $\mathcal{F}_{\mathsf{OPRF}}$ previously;
- $\mathcal{A}$ sent (TESTPWD, sid, $ssid$, S, rw^*) aimed at $\mathcal{F}_{\mathsf{aPAKE}}$ previously, and $\mathsf{rw}^* = F_{sid,\mathsf{S}}(\mathsf{pw})$.

These are exactly the only two ways in which $\mathcal{Z}$ may learn $F_{sid,\mathsf{S}}(\mathsf{pw})$: the first is to query it, and the second is to guess it correctly. Since (1) does not hold, $F_{sid,\mathsf{S}}(\mathsf{pw})$ is random to $\mathcal{Z}$. Therefore, the probability of (2) is $1/2^{2\tau}$; since there are q_{O} possible (OFFLINETESTPWD, sid, rw^*) messages, the distinguishing advantage of $\mathcal{Z}$ is at most $q_{\mathrm{O}}/2^{2\tau}$.

– (EVAL, sid, $ssid$, U, S) (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after inputting (USRSESSION, sid, $ssid$, S, pw') to U.

– (SNDRCOMPLETE, sid, $ssid$, S) and (SVRSESSION, sid, $ssid$, U, S) (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives these after sending (SVRSESSION, sid, $ssid$) to S.

– (USRSESSION, sid, $ssid$, U, S) (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends (RCVCOMPLETE, sid, $ssid$, S^*) aimed at $\mathcal{F}_{\mathsf{OPRF}}$, provided that (i) there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{pw}' \rangle$ in $\mathcal{F}_{\mathsf{OPRF}}$ (or a record $\langle ssid, \mathsf{U}, \mathsf{S} \rangle$ in SIM), (ii) if S is honest not marked COMPROMISED, then $\mathsf{S}^* = \mathsf{S}$, and (iii) $\mathsf{tx}(sid, \mathsf{S}^*) > 0$.

– "correct/wrong guess" on (TESTPWD, sid, $ssid$, P, rw^*) (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends (TESTPWD, sid, $ssid$, P, rw^*) aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, provided that there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{rw}' \rangle$ (if $\mathsf{P} = \mathsf{U}$) or $\langle ssid, \mathsf{S}, \mathsf{U}, \mathsf{rw}' \rangle$ (if $\mathsf{P} = \mathsf{S}$) marked FRESH in $\mathcal{F}_{\mathsf{aPAKE}}$ (or a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^* \rangle$ or $\langle sid, ssid, \mathsf{S}, \mathsf{U} \rangle$ marked FRESH in SIM), and HALT does not occur in the simulated world.
HALT occurs if and only if there are more than one x's such that $\mathsf{rw}^* = F_{sid,\mathsf{S}^*}(x)$ (if $\mathsf{P} = \mathsf{U}$) or $\mathsf{rw}^* = F_{sid,\mathsf{S}}(x)$ (if $\mathsf{P} = \mathsf{S}$). This means that there are x_1 and x_2 such that $F_{sid,\mathsf{S}^*}(x_1) = F_{sid,\mathsf{S}^*}(x_2)$ or $F_{sid,\mathsf{S}}(x_1) = F_{sid,\mathsf{S}}(x_2)$. Since $F_{sid,\mathsf{S}}(\cdot)$ and $F_{sid,\mathsf{S}^*}(\cdot)$ are random functions onto $\{0,1\}^{2\tau}$, we have $\Pr[\text{HALT}] \leq q_{\mathrm{F}}^2/2^{2\tau+1}$.
Next we assume that HALT does not occur. If $\mathsf{P} = \mathsf{U}$, then consider the first message of type (TESTPWD, sid, $ssid$, U, rw^*) (note that $\mathcal{A}$ receives a reply for the first such message only, since $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{rw}' \rangle$ becomes either COMPROMISED or INTERRUPTED after the first message):
In the real world, $\mathcal{Z}$ receives "correct guess" if and only if $\mathsf{rw}^* = F_{sid,\mathsf{S}^*}(\mathsf{pw}')$.
In the simulated world, $\mathcal{Z}$ receives "correct guess" if and only if the followings hold: (1) $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ is defined in SIM, and (2) $\mathsf{rw}^* = F_{sid,\mathsf{S}^*}(\mathsf{pw}')$.
The message sent to $\mathcal{Z}$ in the two worlds is the same unless in the simulated world, the condition (2) above holds, while (1) does not hold (thus it receives "correct guess" in the real world and "wrong guess" in the simulated world). Since $\mathcal{A}$ does not query $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ (otherwise $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ would be defined in SIM), this value is random to $\mathcal{Z}$. Therefore, the probability of (2) for a single U session is $1/2^{2\tau}$; since there are q_{U} U sub-sessions, the distinguishing

advantage of $\mathcal{Z}$ is at most $q_{\mathrm{U}}/2^{2\tau}$.
Similarly, if $\mathsf{P} = \mathsf{S}$, then the probability that $\mathcal{Z}$ can distinguish between the two worlds is at most $q_{\mathrm{S}}/2^{2\tau}$.
We conclude that the distinguishing advantage of $\mathcal{Z}$ after $\mathcal{A}$ sends query $(\textsc{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{rw}^*)$ is at most $(q_{\mathrm{F}}^2 + 2q_{\mathrm{U}} + 2q_{\mathrm{S}})/2^{2\tau+1}$.

– "correct/wrong guess" on $(\textsc{Impersonate}, sid, ssid)$ (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends $(\textsc{Impersonate}, sid, ssid)$ aimed at $\mathcal{F}_{\mathsf{aPAKE}}$, provided that there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{rw}'\rangle$ marked FRESH in $\mathcal{F}_{\mathsf{aPAKE}}$ (or a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*\rangle$ marked FRESH in SIM). Similar with above, $\mathcal{A}$ receives a reply for the first such message only, so we only consider the first such message:
In the real world, $\mathcal{Z}$ receives "correct guess" if and only if there is a record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \mathsf{rw}\rangle$ marked COMPROMISED in $\mathcal{F}_{\mathsf{aPAKE}}$ and $\mathsf{rw}' = \mathsf{rw}$. Note that $\mathsf{rw}' = F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ and $\mathsf{rw} = F_{sid,\mathsf{S}}(\mathsf{pw})$.
In the simulated world, $\mathcal{Z}$ receives "correct guess" if and only if there is a record $\langle \textsc{file}, \mathsf{U}, \mathsf{S}, \mathsf{pw}\rangle$ marked COMPROMISED in $\mathcal{F}_{\mathsf{SaPAKE}}$, $\mathsf{S}^* = \mathsf{S}$ and $\mathsf{pw}' = \mathsf{pw}$.
The message sent to $\mathcal{Z}$ in the two worlds is the same unless $(\mathsf{S}^* \neq \mathsf{S} \vee \mathsf{pw}' \neq \mathsf{pw}) \wedge F_{sid,\mathsf{S}^*}(\mathsf{pw}') = F_{sid,\mathsf{S}}(\mathsf{pw})$ (thus it receives "correct guess" in the real world and "wrong guess" in the simulated world). Since $\mathsf{S}^* \neq \mathsf{S}$ or $\mathsf{pw}' \neq \mathsf{pw}$, $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ and $F_{sid,\mathsf{S}}(\mathsf{pw})$ are two independently random strings in $\{0,1\}^{2\tau}$; therefore, for a single U session, the probability of $F_{sid,\mathsf{S}^*}(\mathsf{pw}') = F_{sid,\mathsf{S}}(\mathsf{pw})$ is at most $1/2^{2\tau}$. Since there are q_{U} U sub-sessions, the distinguishing advantage of $\mathcal{Z}$ is at most $q_{\mathrm{U}}/2^{2\tau}$.

– $(sid, ssid, SK)$ (from U or S); SUCC or FAIL (from $\mathcal{A}$): In both worlds, these messages depend on the state of the corresponding record, which is the same in the two worlds.

We conclude that $\mathcal{Z}$'s view in the real world and the simulated world is the same, except for (1) "correct/wrong guess" after $\mathcal{A}$ sends $(\textsc{OfflineTestPwd}, sid, \mathsf{rw}^*)$, (2) "correct/wrong guess" or HALT after $\mathcal{A}$ sends $(\textsc{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{rw}^*)$, and (3) "correct/wrong guess" after $\mathcal{A}$ sends $(\textsc{Impersonate}, sid, ssid)$. The probabilities that (1), (2) and (3) are different in the two worlds are no more than $q_{\mathrm{O}}/2^{2\tau}$, $(q_{\mathrm{F}}^2 + 2q_{\mathrm{U}} + 2q_{\mathrm{S}})/2^{2\tau+1}$ and $q_{\mathrm{U}}/2^{2\tau}$, respectively. Using a hybrid argument, we can see that $\mathcal{Z}$'s advantage is at most $(q_{\mathrm{F}}^2 + 2q_{\mathrm{O}} + 4q_{\mathrm{U}} + 2q_{\mathrm{S}})/2^{2\tau+1}$.

5 A Compiler from AKE-KCI to Strong aPAKE via OPRF

Our second transformation for building a Strong aPAKE protocol composes an OPRF with an Authenticated Key Exchange (AKE) protocol, "glued" together using authenticated encryption. We require the AKE to be secure in the UC model, namely, to realize the UC KE functionality of [14], and to also be "KCI secure." The latter notion was defined in [33] under a game-based formulation

and formalized in Section 5.1 below in the UC setting. We first prove the OPRF-AKE composition to be SaPAKE secure for protocols where the the last-to-complete (LTC) party in the AKE protocol is the server. The case where the client is the LTC (e.g., using a 2-message AKE protocol without explicit client authentication) is proven in Section 5.4 under a slightly relaxed variant of the SaPAKE functionality.

5.1 UC Definition of AKE-KCI

The KCI notion for KE protocols, which stands for "key-compromise impersonation," captures the property we call "security against reverse impersonation," which concerns an attacker $\mathcal{A}$ who learns party P's long-term keys but otherwise does not actively control P. Resistance to KCI attacks, or "KCI security" for short, postulates that even though $\mathcal{A}$ can impersonate P to other parties, sessions which P itself runs with honest peers need to remain secure. A game-based definition of this notion appears in [33], and here we formalize it in the UC model through functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$ presented in Fig. 6. We specialize functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$ to our user-server setting where only servers can be compromised, but it can be extended to allow for compromise of any protocol party.

In the description below, we assume $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$.

- On (UsrSession, sid, $ssid$, S) from U, send (UsrSession, sid, $ssid$, U, S) to $\mathcal{A}^*$. If there is no record $(ssid, \mathsf{U}, \cdot)$, record $(ssid, \mathsf{U}, \mathsf{S})$ and mark it FRESH.
- On (SvrSession, sid, $ssid$, U) from S, send (SvrSession, sid, $ssid$, U, S) to $\mathcal{A}^*$. If there is no record $(ssid, \mathsf{S}, \cdot)$, record $(ssid, \mathsf{S}, \mathsf{U})$ and mark it FRESH.
- On (Compromise, sid) from $\mathcal{A}^*$, mark S COMPROMISED.
- On (Impersonate, sid, $ssid$) from $\mathcal{A}^*$, if S is marked COMPROMISED and there is a record $(ssid, \mathsf{U}, \mathsf{S})$ marked FRESH, mark the record COMPROMISED.
- On (NewKey, sid, $ssid$, P, SK) from $\mathcal{A}^*$ where $|SK| = \tau$, if there is a record $(ssid, \mathsf{P}, \mathsf{P}')$ not marked COMPLETED, do:
  - If the record is marked COMPROMISED, or either P or P' is corrupted, send $(sid, ssid, SK)$ to P.
  - If the record is marked FRESH, a $(sid, ssid, SK')$ tuple was sent to P', and at that time there was a record $(ssid, \mathsf{P}', \mathsf{P})$ marked FRESH, send $(sid, ssid, SK')$ to P.
  - Else pick $SK'' \leftarrow_{\mathrm{R}} \{0,1\}^\tau$ and send $(sid, ssid, SK'')$ to P.

  Finally, mark $(ssid, \mathsf{P}, \mathsf{P}')$ COMPLETED.

Fig. 6: Functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$

Functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$ extends the standard KE functionality of [14] with two adversarial actions. The first, COMPROMISE, is targeted at a server and captures the compromise of the server's keys. The second is IMPERSONATE which is borrowed from the aPAKE functionality of [23] shown in Fig. 1. This action can only be targeted at users' sessions, and only for sessions with servers compromised via the COMPROMISE action, and it marks such session as COMPROMISED, which implies that the attacker can determine the session key this session outputs, via the NEWKEY action. This models the fact that user's sessions with a compromised S as a peer cannot be assumed to be secure since they could have been run with the adversary who has stolen S's keys. However, sessions at S itself must not be affected by the IMPERSONATE action, and they remain secure. All other elements in $\mathcal{F}_{\mathsf{AKE-KCI}}$ are the same as in the basic UC KE functionality, except of some syntactic specialization to the user-server setting.

AKE-KCI security of HMQV. Our concrete instantiation of a Strong aPAKE protocol, OPAQUE (Fig. 9 in Section 6), is illustrated with HMQV [33] as the AKE-KCI component. The KCI property of HMQV was proved in [33] in the game-based Canetti-Krawczyk model [13] extended to include KCI security. Here we require UC security, namely, a protocol that realizes functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$. Fortunately, [14] proves the equivalence of the game-based definition of [13] and their UC AKE formulation. Thanks to this equivalence, HMQV, as a basic KE, is secure in the UC model. More precisely, this applies to the three-message HMQV with explicit client authentication (which satisfies the "ACK" property required for the equivalence in [14]). For the 2-message version of HMQV, the equivalence still holds using the notion of *non-information oracle* [14] that holds for HMQV under Computational Diffie-Hellman (CDH) assumption in the RO model. For our purposes, however, we need HMQV to realize the extended AKE-KCI functionality of Fig. 6. The equivalence with the game-based definition extends to this case. Indeed, since the original equivalence from [14] holds even in the case of adaptive party corruptions, the COMPROMISE and IMPERSONATE actions introduced here – which constitute a *limited* form of adaptive corruptions – follow as a special case. Finally, we note that the equivalence between the above models also preserves forward secrecy, so this property (proved in the game-based Canetti-Krawczyk model in [33]) holds in the UC too. We note that by the results in [33], the 3-message HMQV enjoys full PFS while the 2-message provides PFS only against passive attackers. The above security of HMQV (without including security against the leakage of ephemeral exponents) is based on the CDH assumption in the RO model [33].

5.2 Strong aPAKE Construction from OPRF and AKE-KCI

Our Strong aPAKE protocol based on OPRF and AKE-KCI is shown in Fig. 7. The protocol uses the same OPRF tool as the Strong aPAKE construction of Section 4, for length parameter $\ell = 2\tau$, which defines the "randomized password" value $\mathsf{rw} = F_k(\mathsf{pw})$ for user U's password pw and OPRF key k held by server S.

Public Components:

- KCI-secure AKE protocol Π with private/public keys denoted p_s, P_s, p_u, P_u;
- Random-key robust authenticated encryption $\mathsf{AE} = (\mathsf{AuthEnc}, \mathsf{AuthDec})$ with (2τ)-bit keys;
- Functionality $\mathcal{F}_{\mathsf{OPRF}}$ with output length parameter $\ell = 2\tau$.

Password Registration

1. On input $(\textsc{StorePwdFile}, sid, \mathsf{U}, \mathsf{pw})$, S generates pairs (p_s, P_s) and (p_u, P_u), and sends $(\textsc{Init}, sid, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{OPRF}}$.
   On $\mathcal{F}_{\mathsf{OPRF}}$'s response $(\textsc{Init}, sid, \mathsf{rw})$, S computes $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_u, P_u, P_s)$ and records $\mathsf{file}[sid] := \langle p_s, P_s, P_u, c\rangle$.

Server Compromise

1. On $(\textsc{StealPwdFile}, sid)$ from $\mathcal{A}$, S retrieves $\mathsf{file}[sid]$ and sends it to $\mathcal{A}$.

Login

1. On $(\textsc{UsrSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, U sends $(\textsc{Eval}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{OPRF}}$.
2. On $(\textsc{SvrSession}, sid, ssid)$, S retrieves $\mathsf{file}[sid] = \langle p_s, P_s, P_u, c\rangle$, sends c to U, sends $(\textsc{SndrComplete}, sid, ssid)$ to $\mathcal{F}_{\mathsf{OPRF}}$ and runs Π on input (p_s, P_s, P_u).
3. On $(\textsc{Eval}, sid, ssid, \mathsf{rw}')$ from $\mathcal{F}_{\mathsf{OPRF}}$ and c from S, U computes $\mathsf{AuthDec}_{\mathsf{rw}'}(c)$. If the result is $\perp$, U outputs $(\textsc{abort}, sid, ssid)$ and halts. Otherwise U parses $(p'_u, P'_u, P'_s) := \mathsf{AuthDec}_{\mathsf{rw}'}(c)$ and runs Π on input (p'_u, P'_u, P'_s).
4. Given Π's local output SK, the corresponding party, either U or S, outputs $(sid, ssid, SK)$.

Fig. 7: Strong aPAKE based on AKE-KCI in the $\mathcal{F}_{\mathsf{OPRF}}$-Hybrid World

We assume that in the AKE-KCI protocol Π each party holds a (private,public) key pair, and that the each party runs the Login subprotocol using its key pair and the public key of the counterparty as inputs. In Password Registration phase, server S generates the user U's keys, and S's password file contains S's key pair p_s, P_s; U's public key P_u; and a ciphertext c of U's private key p_u, and the public keys P_u and P_s created using an Authenticated Encryption scheme using $\mathsf{rw} = F_k(\mathsf{pw})$ as the key. After creating the password file, value p_u is erased at S. In Login phase, S runs OPRF with U, which lets U compute $\mathsf{rw} = F_k(\mathsf{pw})$, it sends c to U, who can decrypt it under rw and retrieves its key-pair p_u, P_u together with the server's key P_s, at which point both parties have appropriate inputs to the AKE-KCI protocol Π to compute the session key.

Role of Authenticated Encryption. The Strong aPAKE protocol of Fig. 7 utilizes an *Authenticated Encryption* scheme $\mathsf{AE} = (\mathsf{AuthEnc}, \mathsf{AuthDec})$ to encrypt and authenticate U's AKE "credential" $m = (p_u, P_u, P_s)$. We encrypt the whole payload m for simplicity, because unlike U's private key p_u, values

P_u, P_s could be public and need to be only authenticated, not encrypted. However, the authentication property of AE must apply to the whole payload. Intuitively, U must authenticate S's public key P_s, but if U derived even its key pair (p_u, P_u) using just the secrecy of $\mathsf{rw} = F_k(\mathsf{pw})$, e.g., using rw as randomness in a key generation, and U then executed AKE on such (p_u, P_u) pair, the resulting protocol would already be insecure. To see an example, if an AKE leaks U's public key input P_u (note that AKE does not guarantee privacy of the public key) then an adversary $\mathcal{A}$ who engages U in a single protocol instance can find U's password pw via an offline dictionary attack by running the OPRF with U on some key k^*, and then given P_u leaked in the subsequent AKE it finds pw s.t. the key generation outputs P_u as a public key on randomness $\mathsf{rw} = F_{k^*}(\mathsf{pw})$.

Thus the role of the authentication property in authenticated encryption is to commit $\mathcal{A}$ to a single guess of rw and consequently, given the OPRF key k^*, to a single guess pw. (Note that our UC OPRF notion implies that F is collision-resistant.) To that end we need the authenticated encryption to satisfy the following property which we call *random-key robustness*:[6] For any efficient algorithm $\mathcal{A}$ there is a negligible probability that $\mathcal{A}$ on input (k_1, k_2) for two random keys k_1, k_2 outputs c s.t. $\mathsf{AuthDec}_{k_1}(c) \neq \perp$ and $\mathsf{AuthDec}_{k_2}(c) \neq \perp$. In other words, it must be infeasible to create an authenticated ciphertext that successfully decrypts under two different randomly generated keys. This property can be achieved in the standard model using e.g. encrypt-then-MAC with a MAC that is collision resistant with respect to the message and key, a property enjoyed by HMAC with full hash output. In the RO model used by our aPAKE application one can also enforce it for any authenticated encryption scheme by attaching to its ciphertext c a hash $H(k, c)$ for a RO hash H with 2τ-bit outputs.

Note on not utilizing $\mathcal{F}_{\mathsf{AKE-KCI}}$. In Fig. 7 we abstract the OPRF protocol as functionality $\mathcal{F}_{\mathsf{OPRF}}$, but we use the real-world AKE-KCI protocol Π, rather than functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$. The reason for this presentation is that in the KE functionality of [14], of which $\mathcal{F}_{\mathsf{AKE-KCI}}$ is an extension, it is not clear how to support a usage of the KE protocol on keys which are computed via some other mechanism than the intended KE key generation. The KE functionality of [14] assumes that each entity keeps its private key as a permanent state, authenticates to a counterparty given its identity, and a KE party cannot specify any bitstring as one's own private key and a counterparty's public key. This is not how we use AKE in our Strong aPAKE of Fig. 7 precisely because U does not keep state and has to reconstruct its keys from a password (via OPRF). However, we can still use the real-world protocol Π, which UC-realizes $\mathcal{F}_{\mathsf{AKE-KCI}}$, giving it the OPRF-computed information as input. In the proof of security we utilize the simulator $\mathsf{SIM}_{\mathsf{AKE}}$, which shows that Π UC-realizes $\mathcal{F}_{\mathsf{AKE-KCI}}$, in our simulator construction, but we rely on its correctness only if U runs Π on the correctly

[6] This notion is a weakening of *full robustness (FROB)* from [20] where the attacker is allowed to choose k_1, k_2 (in our case these keys are random). An even weaker notion, Semi-FROB, is defined in [20] where k_1, k_2 are random but only k_1 is provided to $\mathcal{A}$.

reconstructed (p_u, P_s, P_s), and if the adversary causes U to reconstruct a different string we interpret this as a successful attack on U's login session.

5.3 Proof of Security

Here we state and prove the Strong aPAKE security of the generic OPRF-AKE composition protocol from Fig. 7 for the case where in the AKE protocol, the user completes its session and gets its session key before the server does. The complementary case is treated in Section 5.4.

Formally, we modify the UC functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$ to $\mathcal{F}_{\mathsf{AKE-KCI+}}$ by adding the following line:

- On (NewKey, *sid*, *ssid*, S, SK) from $\mathcal{A}^*$, ignore this message if there is no (*sid*, *ssid*, U) session marked completed.

Theorem 2. *If protocol Π UC-realizes functionality $\mathcal{F}_{\mathsf{AKE-KCI+}}$, then the protocol in Fig. 7 UC-realizes functionality $\mathcal{F}_{\mathsf{SaPAKE}}$ in the $\mathcal{F}_{\mathsf{OPRF}}$-hybrid model.*

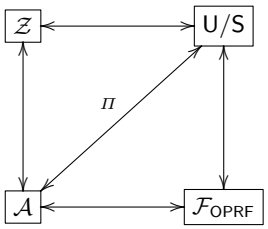
Concretely, fix an environment $\mathcal{Z}$. Suppose that there is a simulator $\mathsf{SIM}_{\mathsf{AKE}}$ such that the distinguishing advantage of an environment, whose computational resources are comparable to $\mathcal{Z}$, between the real execution of Π and $\mathcal{Z}$'s interaction with $\mathsf{SIM}_{\mathsf{AKE}}$ is at most ϵ_{AKE}. Then for any adversary $\mathcal{A}$ against the protocol, there is a simulator SIM that produces a view in the simulated world such that the advantage that $\mathcal{Z}$ has in distinguishing between this view and the view in the real world is no more than $q_{\mathrm{U}} \cdot (2\epsilon_{\mathrm{SEC}} + \epsilon_{\mathrm{AUTH}} + q_{\mathrm{F}}^2 \cdot \epsilon_{\mathrm{RBST}}) + \epsilon_{\mathrm{AKE}}$, where q_{F} is the number of Eval *and* OfflineEval *messages aimed at $\mathcal{F}_{\mathsf{OPRF}}$ from $\mathcal{A}$; ϵ_{SEC}, ϵ_{AUTH} and ϵ_{RBST} are the advantages of an adversary, whose computational resources are comparable to $\mathcal{A}$, in the security games of the semantic security, authenticity, and random-key robustness of AE, respectively; and q_{U} is the number of U sub-sessions which $\mathcal{Z}$ starts via a* UsrSession *message.*

Proof. For any adversary $\mathcal{A}$, we construct a simulator SIM as in Fig. 8. While interacting with $\mathsf{SIM}_{\mathsf{AKE}}$, SIM plays the role of both $\mathcal{F}_{\mathsf{AKE-KCI}}$ and $\mathcal{A}$.

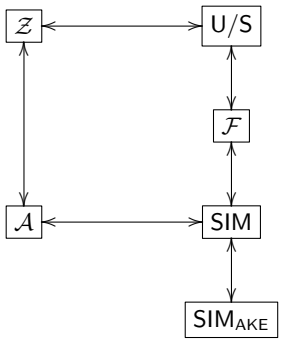
As in the proof of Theorem 1, we assume that $\mathcal{A}$ is a "dummy" adversary that merely passes all its messages and computations to the environment $\mathcal{Z}$, we omit all interactions with corrupted U and S where SIM acts as $\mathcal{F}_{\mathsf{OPRF}}$, and we denote functionality $\mathcal{F}_{\mathsf{SaPAKE}}$ as $\mathcal{F}$.

In order to account for the advantage of the environment $\mathcal{Z}$ in distinguishing between its views in the real world and the simulated world, we compare between these two settings in the different simulator actions and derive the distinguishing advantages in cases where the simulation is not perfect. Below we assume that $\mathcal{Z}$ issues the (StorePwdFile, *sid*, U, pw) command to S for some pw; otherwise any subsequent server-side commands of $\mathcal{Z}$ will not have any effect.

- $\mathsf{file}[sid] = \langle p_s, P_s, P_u, c \rangle$ (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends (Compromise, *sid*) aimed at $\mathcal{F}_{\mathsf{OPRF}}$ and (StealPwdFile, *sid*) to S, provided that $\mathcal{Z}$ input (StorePwdFile, *sid*, U, pw) to S



(a) real world



(b) simulated world

previously.

In both worlds, p_s, P_s and P_u are generated in the same way, and c is computed as $\mathsf{AuthEnc}_{\mathsf{rw}}(p_u, P_u, P_s)$. The only difference is that rw is $F_{sid,\mathsf{S}}(\mathsf{pw})$ in the real world, while it is chosen from random in the simulated world. In $\mathcal{Z}$'s view, rw in the two worlds is the same unless and until it queries $F_{sid,\mathsf{S}}(\mathsf{pw})$ via either of the following two ways: (1) letting $\mathcal{A}$ send $(\textsc{OfflineEval}, sid, \mathsf{S}, \mathsf{pw})$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$ after S is compromised, or (2) letting $\mathcal{A}$ (acting as a corrupted user) send $(\textsc{Eval}, sid, ssid, \mathsf{S}, \mathsf{pw})$ and $(\textsc{RcvComplete}, sid, ssid, \mathsf{S})$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$ in an online session. However, in both cases, once $\mathcal{A}$ sends such message(s), SIM sets $F_{sid,\mathsf{S}}(\mathsf{pw})$ to rw. [7] Therefore, in both worlds, $F_{sid,\mathsf{S}}(\mathsf{pw}) = \mathsf{rw}$ and $\mathcal{Z}$ cannot distinguish.

- $(\textsc{OfflineEval}, sid, \rho)$ (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after $\mathcal{A}$ sends $(\textsc{OfflineEval}, sid, \mathsf{S}, x)$ to $\mathcal{F}_{\mathsf{OPRF}}$, provided that S is corrupted or marked COMPROMISED. The selection of ρ is the same in the two worlds, except that in the simulated world, if $x = \mathsf{pw}$, ρ is set to rw which was chosen from random in advance, while in the real world, ρ is always chosen from random directly. The two cases are identical in $\mathcal{Z}$'s view.
- $(\textsc{Eval}, sid, ssid, \mathsf{U}, \mathsf{S})$ (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives this message after inputting $(\textsc{UsrSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ to U.
- c and $(\textsc{SndrComplete}, sid, ssid, \mathsf{S})$ (from $\mathcal{A}$): In both worlds, $\mathcal{Z}$ receives these two messages after inputting $(\textsc{SvrSession}, sid, ssid)$ to S. As argued above, $\mathcal{Z}$ cannot distinguish the two c's in the two worlds.

[7] Note that we assume that in Π S does not complete its session and output until U completes its session. However, U's message(s) to S are computed using its input (p_u, P_u, P_s), which in turn is computed via decrypting c using rw', and rw' is computed via an OPRF instance on pw'. Therefore, an OPRF instance on pw' must complete before S's session is completed, hence in case (2) above S's session is not completed and SIM will receive "correct guess."

For every sid and every server S, initialize $\mathsf{tx}(sid, \mathsf{S})$ to 0.
Generate two key pairs (p_s, P_s) and (p_u, P_u), pick $\mathsf{rw} \leftarrow_{\mathrm{R}} \{0,1\}^{2\tau}$, compute $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_u, P_u, P_s)$, and record $\mathsf{file}[sid] := \langle p_s, P_s, P_u, c\rangle$.
Let Π_u (resp. Π_s) be U's (resp. S's) algorithm in the execution of Π.

Stealing Password Data and Offline Queries

1. On (Compromise, sid) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$ and (StealPwdFile, sid) from $\mathcal{A}$ aimed at S, send (StealPwdFile, sid) to $\mathcal{F}$.
   If $\mathcal{F}$ returns "password file stolen," mark S compromised and send $\mathsf{file}[sid]$ to $\mathcal{A}$ as a message from S.
2. On (OfflineEval, sid, S, x) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$, if S is marked compromised or corrupted, send (OfflineTestPwd, sid, x) to $\mathcal{F}$. If $\mathcal{F}$ returns "correct guess," set $F_{sid,\mathsf{S}}(x) := \mathsf{rw}$. Regardless, send (OfflineEval, $sid, F_{sid,\mathsf{S}}(x)$) to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{OPRF}}$ (if $F_{sid,\mathsf{S}}(x)$ is undefined, pick $F_{sid,\mathsf{S}}(x) \leftarrow_{\mathrm{R}} \{0,1\}^{2\tau}$).

Password Authentication

1. On (UsrSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) from $\mathcal{F}$, send (Eval, $sid, ssid, \mathsf{U}, \mathsf{S}$) to $\mathcal{A}$ as a message from $\mathcal{F}_{\mathsf{OPRF}}$. Also, if this is the first UsrSession message for $ssid$, record $\langle ssid, \mathsf{U}, \mathsf{S}, \cdot\rangle$.
2. On (SvrSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) from $\mathcal{F}$, retrieve $\mathsf{file}[sid] = \langle p_s, P_s, P_u, c\rangle$, send c and (SndrComplete, $sid, ssid, \mathsf{S}$) to $\mathcal{A}$ as a message from S to U and from $\mathcal{F}_{\mathsf{OPRF}}$, respectively, and send (SvrSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) to $\mathsf{SIM}_{\mathsf{AKE}}$ as a message from $\mathcal{F}_{\mathsf{AKE-KCI}}$. Also, if this is the first SvrSession message for $ssid$, set $\mathsf{tx}(sid, \mathsf{S})$++.
3. On (RcvComplete, $sid, ssid, \mathsf{S}^*$) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$, retrieve $\langle ssid, \mathsf{U}, \mathsf{S}, \cdot\rangle$; ignore this message if (i) such record does not exist, or (ii) S is honest and not marked compromised and $\mathsf{S}^* \neq \mathsf{S}$, or (iii) $\mathsf{tx}(sid, \mathsf{S}^*) = 0$. Else set $\mathsf{tx}(sid, \mathsf{S}^*)$−−, augment $\langle ssid, \mathsf{U}, \mathsf{S}, \cdot\rangle$ to $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*, \cdot\rangle$ and mark $(ssid, \mathsf{U})$ completed.

Key Generation and Authentication

1. On (Eval, $sid, ssid, \mathsf{S}, x$) and (RcvComplete, $sid, ssid, \mathsf{S}$) from $\mathcal{A}$ aimed at $\mathcal{F}_{\mathsf{OPRF}}$, in addition to simulating $\mathcal{F}_{\mathsf{OPRF}}$ by running its code, send (TestPwd, $sid, ssid, \mathsf{S}, x$) to $\mathcal{F}$. If $\mathcal{F}$ returns "correct guess," set $F_{sid,\mathsf{S}}(x) := \mathsf{rw}$.
2. As soon as $(ssid, \mathsf{U})$ is marked completed and a c' is sent from $\mathcal{A}$ aimed at U, retrieve $\mathsf{file}[sid] = \langle p_s, P_s, P_u, c\rangle$ and $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{S}^*, \cdot\rangle$, and proceed as follows:
   - If $c' = c$ and $\mathsf{S}^* = \mathsf{S}$, send (TestAbort, $sid, ssid, \mathsf{U}$) to $\mathcal{F}$.
     If $\mathcal{F}$ returns Succ (which means that $\mathsf{pw}' = \mathsf{pw}$), send (UsrSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) to $\mathsf{SIM}_{\mathsf{AKE}}$ as a message from $\mathcal{F}_{\mathsf{AKE-KCI}}$. Also, if S is compromised, and there was an (OfflineTestPwd, sid, x) message to $\mathcal{F}$ with response "correct guess," send (Impersonate, $sid, ssid$) to $\mathcal{F}$ (the response must be "correct guess" since $\mathsf{pw}' = \mathsf{pw}$), record (p_u, P_u, P_s) and mark this case $(**)$-(a); else mark this case $(*)$.
   - Else for every x such that $F_{sid,\mathsf{S}^*}(x)$ is defined (denote it y), check whether $\mathsf{AuthDec}_y(c') \neq \perp$.
     - If there are more than one such x's, output halt and abort.
     - If there is a unique such x, send (TestPwd, $sid, ssid, \mathsf{U}, x$) to $\mathcal{F}$.
       If $\mathcal{F}$ returns "correct guess," parse and record $(p'_u, P'_u, P'_s) := \mathsf{AuthDec}_y(c')$. Mark this case $(**)$-(b).
       If $\mathcal{F}$ returns "wrong guess," send (TestAbort, $sid, ssid, \mathsf{U}$) to $\mathcal{F}$.
     - If there is no such x, send (TestAbort, $sid, ssid, \mathsf{U}$) to $\mathcal{F}$.
3. In case $(*)$: (i) On (Impersonate, $sid, ssid$) from $\mathsf{SIM}_{\mathsf{AKE}}$, pass this message to $\mathcal{F}$; (ii) While $\mathsf{SIM}_{\mathsf{AKE}}$ simulates the execution of Π, pass messages between it and $\mathcal{A}$; (iii) On (NewKey, $sid, ssid, \mathsf{P}, SK$) from $\mathsf{SIM}_{\mathsf{AKE}}$, pass this message to $\mathcal{F}$.
4. In case $(**)$: (i) retrieve record (p'_u, P'_u, P'_s); (ii) On $\mathcal{A}$'s message as from S to U in the execution of Π, run Π_u on (p'_u, P'_u, P'_s); (iii) When Π_u is completed with output SK, send (NewKey, $sid, ssid, \mathsf{U}, SK$) to $\mathcal{F}$.
5. If case $(*)$ does not occur, no matter case $(**)$ occurs or not: (i) On $\mathcal{A}$'s message as from U to S in the execution of Π, run Π_s on (p_s, P_s, P_u); (ii) When Π_s is completed with output SK, send (NewKey, $sid, ssid, \mathsf{U}, SK$) to $\mathcal{F}$.

Fig. 8: The Simulator SIM

– (ABORT, sid, $ssid$) (from U): In both worlds, $\mathcal{Z}$ may receive this message after $\mathcal{A}$ sends (RCVCOMPLETE, sid, $ssid$, S^*) aimed at $\mathcal{F}_{\mathsf{OPRF}}$ and c' aimed at U, provided that (i) there is a record $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{pw}' \rangle$ in $\mathcal{F}_{\mathsf{OPRF}}$ (or a record $\langle ssid, \mathsf{U}, \mathsf{S}, \cdot \rangle$ in SIM), (ii) if S is honest and not marked COMPROMISED, then $\mathsf{S}^* = \mathsf{S}$, and (iii) $\mathsf{tx}(sid, \mathsf{S}^*) > 0$.
Note that in the simulated world $\mathcal{Z}$ may see a HALT message from SIM at this time. HALT occurs when there exist $x_1 \neq x_2$ such that $\mathsf{AuthDec}_{y_1}(c'), \mathsf{AuthDec}_{y_2}(c') \neq \perp$, where $y_1 = F_{sid,\mathsf{S}^*}(x_1)$ and $y_2 = F_{sid,\mathsf{S}^*}(x_2)$. Since $F_{sid,\mathsf{S}^*}(\cdot)$ is a random function onto $\{0,1\}^{2\tau}$, y_1 and y_2 are independent random strings in $\{0,1\}^{2\tau}$; thus, for fixed y_1 and y_2, the probability that $\mathcal{A}$ finds c' such that $\mathsf{AuthDec}_{y_1}(c') \neq \perp$ and $\mathsf{AuthDec}_{y_2}(c') \neq \perp$ is at most ϵ_{RBST} due to the random-key robustness of AE. Since $\mathcal{A}$ queries F q_{F} times, there are q_{F} independent y's; using a polynomial reduction, we derive that the probability of HALT for a single U sub-session is at most $q_{\text{F}}^2 \cdot \epsilon_{\text{RBST}}$. Since there are q_{U} U sub-sessions, we have that $\Pr[\text{HALT}] \leq q_{\text{U}} q_{\text{F}}^2 \cdot \epsilon_{\text{RBST}}$.
Next suppose that HALT does not occur. In the real world, $\mathcal{Z}$ receives (ABORT, sid, $ssid$) from U if and only if $\mathsf{AuthDec}_{\mathsf{rw}'}(c') = \perp$; that is, $\mathcal{Z}$ does not receive this message if and only if $\mathsf{AuthDec}_{\mathsf{rw}'}(c') \neq \perp$. There are only three possibilities:
(1) $(\mathsf{pw}', \mathsf{S}^*, c') = (\mathsf{pw}, \mathsf{S}, c)$: Then $\mathsf{rw}' = \mathsf{rw} = F_{sid,\mathsf{S}}(\mathsf{pw})$, thus $\mathsf{AuthDec}_{\mathsf{rw}'}(c') = \mathsf{AuthDec}_{\mathsf{rw}}(c) = (p_u, P_u, P_s)$.
(2) $\mathcal{A}$ queries $\mathsf{rw}' = F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ previously, and $\mathsf{AuthDec}_{\mathsf{rw}'}(c') \neq \perp$: If $\mathcal{A}$ learns rw', then it can compute an $\mathsf{AuthEnc}$ instance on rw' and any message to find a c' such that $\mathsf{AuthDec}_{\mathsf{rw}'}(c') \neq \perp$.
(3) Other cases where $\mathcal{A}$ finds a c' such that $\mathsf{AuthDec}_{\mathsf{rw}'}(c') \neq \perp$, while rw' is independently random of everything else in $\mathcal{Z}$'s view (since $\mathcal{A}$ does not query $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$), and $\mathcal{Z}$ does not query $\mathsf{AuthEnc}_{\mathsf{rw}'}(p'_u, P'_u, P'_s)$ ($\mathcal{Z}$ queries $\mathsf{AuthEnc}_{\mathsf{rw}'}(p'_u, P'_u, P'_s)$ by setting $\mathsf{pw}' = \mathsf{pw}$ [thus making $\mathsf{rw}' = \mathsf{rw}$] and receiving $c = \mathsf{AuthEnc}_{\mathsf{rw}}(p_u, P_u, P_s)$ from S). Since AE is an authenticated encryption, the probability of (3) is at most ϵ_{AUTH} for a single U sub-session; since there are q_{U} U sub-sessions, the probability of (3) is at most $q_{\text{U}} \cdot \epsilon_{\text{AUTH}}$.
In the simulated world, $\mathcal{Z}$ does not receive this message if and only if either of the following two conditions holds:
(1) $c' = c$, $\mathsf{S}^* = \mathsf{S}$ and $\mathcal{F}$ returns SUCC on (TESTABORT, sid, $ssid$, U) from SIM. The last condition holds if and only if there are two records $\langle ssid, \mathsf{U}, \mathsf{S}, \mathsf{pw}' \rangle$ and $\langle ssid, \mathsf{S}, \mathsf{U}, \mathsf{pw}'' \rangle$, the former marked FRESH and $\mathsf{pw}' = \mathsf{pw}''$. Note that no TESTPWD, IMPERSONATE or NEWKEY message has been issued yet, so the record must be FRESH. According to the syntax of SVRSESSION, we have $\mathsf{pw}'' = \mathsf{pw}$. Therefore, the last condition is equivalent to $\mathsf{pw}' = \mathsf{pw}$, thus this case is equivalent to case (1) in the real world.
(2) There exists x such that $y = F_{sid,\mathsf{S}^*}(x)$ is defined in SIM, $\mathsf{AuthDec}_y(c') \neq \perp$ and $\mathcal{F}$ returns "correct guess" on (TESTPWD, sid, $ssid$, x) from SIM. The last condition is equivalent to

$x = \mathsf{pw}'$; thus, the three conditions combined are equivalent to $\mathsf{rw}' = F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ is defined in SIM and $\mathsf{AuthDec}_{\mathsf{rw}'}(c') \neq \perp$. SIM defines $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$ only when receiving $(\textsc{OfflineEval}, sid, \mathsf{S}^*, \mathsf{pw}')$ from $\mathcal{A}$. Therefore, this case is equivalent to case (2) in the real world.

Hence, $\mathcal{Z}$ receives this message in the two worlds under the same conditions, except for case (3) in the real world.

– Messages sent between U and S while executing Π (from $\mathcal{A}$), and $(sid, ssid, SK')$ (from U and S): In both worlds, the messages between U and S are simulated during the execution of Π, and $\mathcal{Z}$ receives $(sid, ssid, SK')$ when Π is completed and sends output to U or S. We use the notation of cases as shown in the description of SIM:
  - In case $(*)$: In the real world, U and S execute Π on (p_u, P_u, P_s) and (p_s, P_s, P_u), respectively (i.e., U and S's inputs match each other); $\mathcal{A}$'s view consists of $\textsc{file}[sid] = \langle p_s, P_s, P_u, c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_u, P_u, P_s)\rangle$ at most (note that rw is random in $\mathcal{A}$'s view). We argue that in this case $\mathcal{Z}$'s distinguishing advantage is at most $q_{\mathrm{U}} \cdot \epsilon_{\mathrm{SEC}} + \epsilon_{\mathrm{AKE}}$. In the first step we replace c with $\mathsf{AuthEnc}_{\mathsf{rw}}(\tilde{p}_u, \tilde{P}_u, P_s)$, where $(\tilde{p}_u, \tilde{P}_u)$ is a pair of (independently) random (private,public) keys; according to the semantic security of AE, the distinguishing advantage between these two games is at most ϵ_{SEC} for a single c. Since there are q_{U} c's in total, the distinguishing advantage is at most $q_{\mathrm{U}} \cdot \epsilon_{\mathrm{SEC}}$. After this step c becomes independent of p_u, hence $\mathcal{Z}$'s view is identical to the environment's view in a normal execution of Π (plus some strings independently random of everything else), which is simulated by $\mathsf{SIM}_{\mathsf{AKE}}$ in the simulated world. Therefore, the distinguishing advantage of $\mathcal{Z}$ between the intermediate game and the simulated world is at most ϵ_{AKE}.
  - In case $(**)$, when $\mathcal{A}$ sends messages aimed at U: In the real world, either (a) U and S execute Π on (p_u, P_u, P_s) and (p_s, P_s, P_u), respectively, but p_u, P_u, p_s, P_s are known to $\mathcal{A}$, or (b) U and S execute Π on (p'_u, P'_u, P'_s) and (p_s, P_s, P_u), respectively (i.e., U and S's inputs do not match). This is simulated by SIM running Π_s on (p_u, P_u, P_s) in subcase (a), or on (p'_u, P'_u, P'_s) in subcase (b) directly. In both subcases, in the simulated world, U's session is compromised (thus SIM can set U's output session key), hence the simulation is perfect.
  - When $\mathcal{A}$ sends messages aimed at S (other than in case $(*)$): Similar to above, this is simulated by SIM running Π_s on (p_s, P_s, P_u) directly. If $\mathcal{A}$ queries $F_{sid,\mathsf{S}}(\mathsf{pw})$, in the simulated world S's session is compromised, hence the simulation is perfect. Otherwise rw is random in $\mathcal{Z}$'s view, so similar to the argument in case $(*)$, we replace c with $\mathsf{AuthEnc}_{\mathsf{rw}}(\tilde{p}_u, \tilde{P}_u, P_s)$ for random $(\tilde{p}_u, \tilde{P}_u)$, and then c becomes independent of p_u. After that, S's session key is independently random in $\mathcal{Z}$'s view, which is simulated perfectly.

Based on the analysis above, we construct a sequence of games below, starting from the real world and ending at the simulated world. Let $\Delta(\mathbf{G}_i, \mathbf{G}_{i+1})$ be $\mathcal{Z}$'s distinguishing advantage between games $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$.

$\mathbf{G}_0$ is the real world.

In $\mathbf{G}_1$, we let the challenger output HALT and abort if there exist $x_1 \neq x_2$ such that $\mathcal{A}$ queries both $y_1 = F_{sid,\mathsf{S}^*}(x_1)$ and $y_2 = F_{sid,\mathsf{S}^*}(x_2)$ (via interacting with $\mathcal{F}_{\mathsf{OPRF}}$ offline or online), and $\mathsf{AuthDec}_{y_1}(c'), \mathsf{AuthDec}_{y_2}(c') \neq \perp$. We have $\Delta(\mathbf{G}_0, \mathbf{G}_1) \leq q_{\mathrm{U}} \cdot q_{\mathrm{F}}^2 \cdot \epsilon_{\mathrm{RBST}}$.

In $\mathbf{G}_2$, we let U output (ABORT, sid, $ssid$) if $(\mathsf{pw}', \mathsf{S}^*, c') \neq (\mathsf{pw}, \mathsf{S}, c)$, $\mathcal{A}$ does not query $F_{sid,\mathsf{S}^*}(\mathsf{pw}')$, and $\mathsf{AuthDec}_{\mathsf{rw}'}(c') \neq \perp$. We have $\Delta(\mathbf{G}_1, \mathbf{G}_2) \leq q_{\mathrm{U}} \cdot \epsilon_{\mathrm{AUTH}}$.

In $\mathbf{G}_3$, in the case that $(\mathsf{pw}', \mathsf{S}^*, c') = (\mathsf{pw}, \mathsf{S}, c)$, and $\mathcal{A}$ does not query $F_{sid,\mathsf{S}}(\mathsf{pw})$ offline (denote this case $(*)$), we replace the real execution of Π with the simulation by $\mathsf{SIM}_{\mathsf{AKE}}$, as shown in case $(*)$ in the description of SIM. We have $\Delta(\mathbf{G}_2, \mathbf{G}_3) \leq q_{\mathrm{U}} \cdot \epsilon_{\mathrm{SEC}} + \epsilon_{\mathrm{AKE}}$.

In $\mathbf{G}_4$, in the case that $(*)$ does not occur, we replace the real execution of Π_s with $\mathcal{A}$ with the simulation by $\mathsf{SIM}_{\mathsf{AKE}}$, as shown in the description of SIM. We have $\Delta(\mathbf{G}_3, \mathbf{G}_4) \leq q_{\mathrm{U}} \cdot \epsilon_{\mathrm{SEC}}$.

As shown in the analysis above, $\mathcal{Z}$'s view in $\mathbf{G}_4$ is identical to $\mathcal{Z}$'s view in the simulated world, where the game challenger of $\mathbf{G}_4$ is split into the simulator SIM and the SaPAKE functionality $\mathcal{F}$.

Summing up the results above, we derive that $\mathcal{Z}$'s distinguishing advantage between the real world and the simulated world is at most $q_{\mathrm{U}} \cdot (2\epsilon_{\mathrm{SEC}} + \epsilon_{\mathrm{AUTH}} + q_{\mathrm{F}}^2 \cdot \epsilon_{\mathrm{RBST}}) + \epsilon_{\mathrm{AKE}}$, which completes the proof.

5.4 Client as last-to-complete and relaxed SaPAKE functionality

In the previous subsection we proved the Strong aPAKE security of the OPRF-AKE composed protocol from Fig. 7 in the case that the user completes its session first in the AKE protocol. Here we extend the proof to the case that the server completes its session first. This requires a relaxation of the SaPAKE functionality.

The relaxed SaPAKE functionality. The UC AKE-KCI protocol (which realizes $\mathcal{F}_{\mathsf{AKE-KCI}}$) above, combined with the UC OPRF protocol, does not necessarily yield a UC Strong aPAKE. Indeed, suppose that the AKE-KCI allows S to output its key and end its session before U computes the OPRF output. Then when an adversarial user U^* tests a candidate password pw^* (i.e., upon receiving S's ciphertext c, U^* computes the OPRF on pw^* and uses the result to decrypt S's ciphertext), at the time when the simulator is able to extract pw^* from U^*'s OPRF computation, S's session is already completed. Hence the simulator fails to complete its simulation as it cannot send a TestPwd message to the SaPAKE functionality (recall that in the Strong aPAKE functionality, TestPwd queries on a completed session do not have any effect).

To overcome this technical difficulty in the proof, we slightly relax the SaPAKE functionality $\mathcal{F}_{\mathsf{SaPAKE}}$ in a way that allows for the completion of simulation when the server completes its session first in the AKE protocol. The modified functionality, denoted $\mathcal{F}_{\mathsf{SaPAKE}^*}$, captures security in the latter case

without seemingly weakening the practical security of the model. The modification to $\mathcal{F}_{\mathsf{SaPAKE}}$ from Fig. 1 is as follows: The (ideal) attacker is allowed to send a *single* TestPwd message on S to the functionality after S's session is completed, and if the password guess is correct, the functionality returns S's session key. That is, $\mathcal{F}_{\mathsf{SaPAKE}^*}$ is obtained from $\mathcal{F}_{\mathsf{SaPAKE}}$ by adding the following line to the description of the functionality:

– On (TestPwd, sid, $ssid$, S, pw^*) from $\mathcal{A}^*$, if there is a record $\langle ssid, \mathsf{S}, \mathsf{U}, \mathsf{pw}\rangle$ marked completed with session key SK, and this is the first TestPwd message for $ssid$ and S, do: if $\mathsf{pw}^* = \mathsf{pw}$, return SK to $\mathcal{A}^*$; else return "wrong guess."

Theorem 3. *If protocol Π UC-realizes functionality $\mathcal{F}_{\mathsf{AKE-KCI}}$, then protocol in Fig. 7 UC-realizes functionality $\mathcal{F}_{\mathsf{SaPAKE}^*}$ in the $\mathcal{F}_{\mathsf{OPRF}}$-hybrid model.*

Proof. The proof is the same as in the case of Theorem 2 except for the following difference. When in Π, S is not the LTC party, and $\mathcal{A}$ (acting as a corrupted user) sends (Eval, sid, $ssid$, S, pw) and (RcvComplete, sid, $ssid$, S) aimed at $\mathcal{F}_{\mathsf{OPRF}}$ after S's session is completed (denote its session key SK), SIM acts as follows. It sends (TestPwd, sid, $ssid$, S, pw^*) to $\mathcal{F}$, and since S's session is already completed, SIM receives SK. Then SIM only needs to simulate $\mathcal{Z}$'s view when $\mathcal{A}$ sends messages in Π aimed at S. This is done by executing Π_s on (p_s, P_s, P_u), and when it is completed, sending (NewKey, sid, $ssid$, U, SK) to $\mathcal{F}$. The resultant simulation is perfect.

6 OPAQUE: A Strong Asymmetric PAKE Instantiation

Figure 9 shows OPAQUE, a concrete instantiation of the generic OPRF+AKE protocol from Fig. 7. An illustration is presented in Figure 10.

The OPRF is instantiated with the DH-OPRF scheme from [27] recalled in Appendix A, while the AKE protocol can be instantiated with any UC-secure 2-message implicitly-authenticated AKE-KCI; in Fig. 9 this is illustrated with HMQV [33]. Note that the two messages of DH-OPRF and the two messages from HMQV (or a similar protocol) run "in parallel" hence obtaining a 2-message SaPAKE.

By Theorem 2 on the security of the generic OPRF+AKE construction, by Lemma 1 in Appendix A on the security of DH-OPRF, and by security of HMQV (see below), we get that protocol OPAQUE realizes functionality $\mathcal{F}_{\mathsf{SaPAKE}}$, hence it is a provably-secure Strong aPAKE, under the One-More Diffie-Hellman assumption [3, 27] in ROM.

6.1 Protocol Details and Properties

We expand on the specification of OPAQUE and the protocol's properties.

• *Password registration.* Password registration is the only part of the protocol assumed to run over secure channels where parties can authenticate each other.

Public Parameters and Components

- Security parameter τ
- Group G of prime order $q, |q| = 2\tau$ and generator g (G^* denotes $G \setminus \{1\}$).
- Hash functions $H(\cdot,\cdot)$, $H'(\cdot)$ with ranges $\{0,1\}^{2\tau}$ and G, respectively.
- Pseudorandom function (PRF) $f(\cdot)$ with range $\{0,1\}^{2\tau}$.
- OPRF function defined as $F_k(x) = H(x, (H'(x))^k)$ for key $k \in \mathbb{Z}_q$.
- Key-committing authenticated encryption scheme $(\mathsf{AuthEnc}, \mathsf{AuthDec})$.
- Key exchange formula KE defined below.

Password Registration

1. (StorePwdFile, $sid, \mathsf{U}, \mathsf{pw}$): S computes $k_s \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $\mathsf{rw} := F_{k_s}(\mathsf{pw})$, $p_s \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $p_u \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, $P_s := g^{p_s}$, $P_u := g^{p_u}$, $c \leftarrow \mathsf{AuthEnc}_{\mathsf{rw}}(p_u, P_u, P_s)$; it records $\mathsf{file}[sid] := \langle k_s, p_s, P_s, P_u, c\rangle$.

Login

1. (UsrSession, $sid, ssid, \mathsf{S}, \mathsf{pw}$): U picks $r, x_u \leftarrow_{\mathrm{R}} \mathbb{Z}_q$; sets $\alpha := (H'(\mathsf{pw}))^r$ and $X_u := g^{x_u}$; sends α and X_u to S.
2. (SvrSession, $sid, ssid$): On input α from U, S proceeds as follows:
   (a) Checks that $\alpha \in G^*$. If not, outputs (abort, $sid, ssid$) and halts;
   (b) Retrieves $\mathsf{file}[sid] = \langle k_s, p_s, P_s, P_u, c\rangle$;
   (c) Picks $x_s \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ and computes $\beta := \alpha^{k_s}$ and $X_s := g^{x_s}$;
   (d) Computes $K := \mathsf{KE}(p_s, x_s, P_u, X_u)$ and $SK := f_K(0)$;
   (e) Sends β, X_s and c to U;
   (f) Outputs $(sid, ssid, SK)$.
3. On β, X_s and c from S, U proceeds as follows:
   (a) Checks that $\beta \in G^*$. If not, outputs (abort, $sid, ssid$) and halts;
   (b) Computes $\mathsf{rw} := H(\mathsf{pw}, \beta^{1/r})$;
   (c) Computes $\mathsf{AuthDec}_{\mathsf{rw}}(c)$. If the result is $\perp$, outputs (abort, $sid, ssid$) and halts. Otherwise sets $(p_u, P_u, P_s) := \mathsf{AuthDec}_{\mathsf{rw}}(c)$;
   (d) Computes $K := \mathsf{KE}(p_u, x_u, P_s, X_s)$ and $SK := f_K(0)$;
   (e) Outputs $(sid, ssid, SK)$.

Key exchange formula KE with HMQV instantiation (if any of $X_u, P_u, X_s, P_s \notin G^*$ the receiving party outputs (abort, $sid, ssid$) and halts)

$$\text{For } \mathsf{S}\text{: } \mathsf{KE}(p_s, x_s, P_u, X_u) = H\left((X_u P_u^{e_u})^{x_s + e_s p_s}\right)$$
$$\text{For } \mathsf{U}\text{: } \mathsf{KE}(p_u, x_u, P_S, X_S) = H\left((X_s P_s^{e_s})^{x_u + e_u p_u}\right)$$

where $e_u = H(X_u, \mathsf{S}) \bmod q, e_s = H(X_s, \mathsf{U}) \bmod q$.

Fig. 9: Protocol OPAQUE

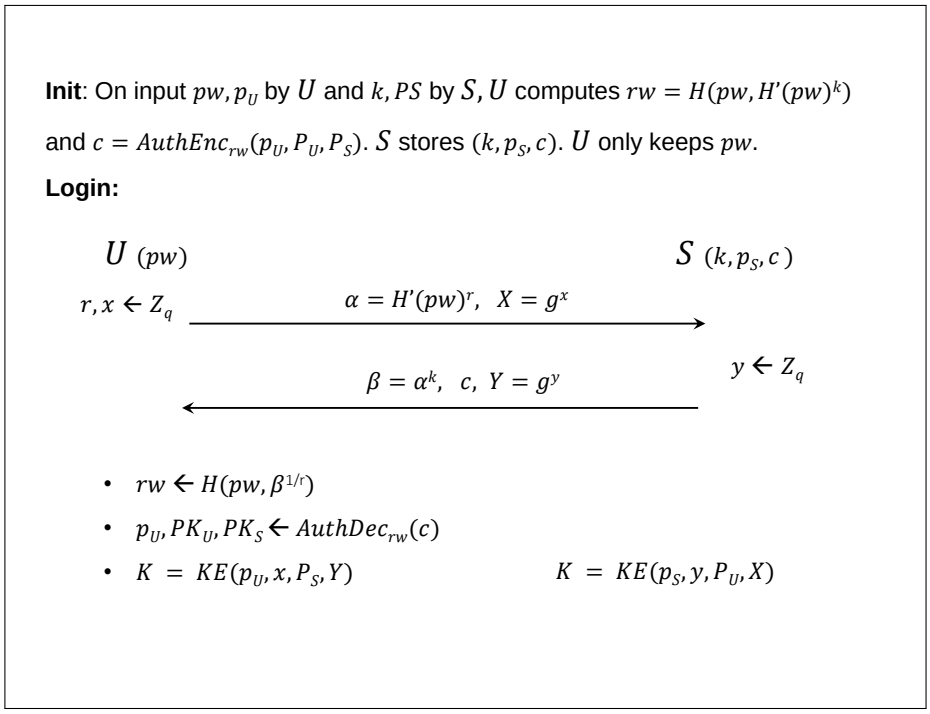


Fig. 10: Schematic Representation of OPAQUE (see Fig. 9 for the details)

We note that while OPAQUE is presented with S doing all the registration operations, in practice one may want to avoid that. Instead, we can let S choose an OPRF key k_s and U choose pw, and then run the OPRF protocol between U and S so only U learns its secrets $(\mathsf{pw}, \mathsf{rw}, p_u)$ and only S learns p_s. A problem arises with this approach if S's policy is to check the user's password for compliance with some rules. A possible workaround is to adapt techniques from [32] that present zero-knowledge proofs for proving compliance without disclosing the password.

- *Authenticated encryption.* As specified in Section 5.2, the scheme AuthEnc used in the protocol needs to satisfy the key-committing property defined there. In practice, using an encrypt-then-mac scheme with HMAC-256 (or larger) as the MAC provides this property (if a scheme does not have this property then adding on top of it such a HMAC computed on the scheme's ciphertext will ensure this property).
- *Key exchange.* The generic AKE representation via the KE formula applies to any protocol whose session key is computed as a function of the long-term private-public key pair of each party and ephemeral session-specific private-public values. These values are represented as (p_s, P_s, x_s, X_s) for the server and (p_u, P_u, x_u, X_u) for the user. We note that while more general

key-exchange protocols can be used with OPAQUE, this representation applies to many such protocols and, in particular, to HMQV [33] which we use here as our main instantiation.

• *Explicit mutual authentication.* The protocol as illustrated takes just two messages but does not provide explicit user authentication. With a third message the protocol achieves mutual authentication by simply adding the value $f_K(1)$ to the server's message and adding a third message where U sends $f_K(2)$ to S. Each party verifies that the value received from the other is computed correctly and if not it aborts.

• *Use of HMQV.* Recall that the security of OPAQUE depends on the KE protocol being AKE-secure in the UC model with the additional KCI property; namely, it should realize the AKE-KCI UC functionality from Fig. 6. As argued in Section 5.1, HMQV indeed realizes this functionality (under the CDH assumption in the RO model), hence it is appropriate for use in OPAQUE. Moreover, HMQV enjoys forward secrecy. Specifically, the 2-message protocol provides weak forward secrecy (i.e., forward secrecy is guaranteed for sessions where the user's message delivered to the server came from the real U) while the 3-message variant with explicit client authentication provides full forward secrecy, namely, against arbitrary active attacks [33].

• *Forward secrecy.* This property (or lack of it) is inherited by OPAQUE from the key exchange component KE. In the case of HMQV, forward secrecy is achieved as stated above. *One cannot overstate the importance of forward secrecy in password protocols: it guarantees that past session keys remain secure upon the compromise of a user's password (or server's information).*

• *User iterated hashing.* OPAQUE can be strengthened by increasing the cost of a dictionary attack in case of server compromise. This is done by changing the computation of rw to $\mathsf{rw} = H^n(F_k(\mathsf{pw}))$, that is, the client applies n iterations of the function H on top of the result of the OPRF value $F_k(\mathsf{pw})$. In practice, the iterations H^n would be replaced with one of the standard password-based KDFs, such as PBKDF2 [31] or bcrypt [40]. This forces an attacker that compromises the password file at the server to compute for *each* candidate password pw' the function $F_k(\mathsf{pw}')$ as well as the additional n hash iterations. Note that n needs not be remembered by the user; it can be sent from S to U in the server's message. Furthermore, one can follow Boyen's design and apply the probabilistic Halting KDF function [8] as used in [9] so that the iterations count is hidden from the attacker and even from the server.

• *Performance.* OPAQUE takes two messages (three with explicit mutual authentication); one exponentiation for S, two and a hashing-into-G for U, plus the cost of KE. With HMQV, the latter cost is one offline fixed-base exponentiation and one multi-exponentiation (at the cost of 1.16 regular exponentiations) per party (about three exponentiations in total for the server and four for the user). All exponentiations are in regular DH groups, hence accommodating the fastest elliptic curves (e.g., no pairings). It is common in PAKE protocols to count number of group elements transmitted between the

parties. In OPAQUE, U sends two while S sends three (one, P_u, can be omitted at the cost of one fixed-based exponentiation at the client).

• *Performance comparison.* The introduction presents background on OPAQUE and other password protocols. Here we provide a comparison with the more efficient among these protocols, particularly those that are being, or have been, considered for standardization. Clearly, OPAQUE is superior security-wise as the only one not subject to pre-computation attacks, but it also fares well in terms of performance.

AugPAKE [43, 44], is computationally very efficient with only 2.17 exponentiations per party; however, it uses 4 messages and does not provide forward secrecy. In addition, the protocol has only been analyzed as a PAKE protocol, not aPAKE [44]. Another proposed aPAKE protocol, SPAKE2+ [2, 15], uses two messages only and 3 multi-exponentiations (or about 3.5 exponentiations) per party which is similar to OPAQUE cost. The security of the protocol has only been informally argued in [15] and to the best of our knowledge no formal analysis has appeared. We also mention SRP which has been included in TLS ciphersuites in the past but is considered outdated as it does not have an instantiation that works over elliptic curves (the protocol is defined over rings and uses both addition and multiplication). Its implementations over RSA moduli is therefore less efficient than those over elliptic curve; it also takes 4 messages.

We also mention two very recent schemes that have been formally analyzed as aPAKE protocols but, as the rest, are vulnerable to pre-computation. The protocol VTBPEKE in [39] uses 3 messages and 4 exponentiations per party and was proven secure in the non-UC aPAKE model of [7], while [29] shows a *simultaneous* one-round scheme that they prove secure in the UC aPAKE model of [23] augmented with adaptive security. The protocol works over bilinear groups and its computational cost includes 4 exponentiations and 3 pairing per party. We note that all of the above protocols require an initial message from server to user in order to transmit salt, which may result in one or two added messages to the above message counts (except for VTBPEKE which already includes the salt transmission in its 3 messages). Also, all these protocols, like OPAQUE, work in the RO model.

• *Threshold implementation.* We comment on a simple extension of OPAQUE that can be very valuable in large deployments, namely, the ability to implement the OPRF phase as a Threshold OPRF [28]. In this case, an attacker needs to break into a threshold of servers to be able to impersonate the servers to the user or to run an offline dictionary attack. Such an implementation requires no user-side changes, i.e., the user does not need to know if the system is implemented with one or multiple servers.

• *OPAQUE as a general secret retrieval mechanism.* An important feature of OPAQUE is that it can serve not only as an aPAKE protocol but more generally as a means for retrieving a secret or credential from a server (such a secret is protected under ciphertext c stored at the server). In this functionality, OPAQUE acts as a 1-out-of-1 implementation of the PPSS scheme from [28]. The retrieved

secret can be used to protect information such as a bitcoin wallet, serve as a user-controlled encryption key for a backup or other information repository (e.g., a password manager), used as an authentication or signing key, and more. This offers a far more secure alternative to the practice of deriving low-entropy secrets directly from a user's password.

6.2 OPAQUE and TLS: Client authentication and hedging against PKI failures

As discussed earlier, OPAQUE offers a much more secure alternative to password-authenticated key exchange than the current practice of transmitting passwords over TLS. Yet, OPAQUE (as any other aPAKE) still requires additional mechanisms for negotiating cryptographic parameters (such as crypto algorithms) and for establishing the means needed to encrypt and authenticate communications using the keys generated by OPAQUE. Thus, it is natural to compose OPAQUE with the TLS protocol to offer strong password security while leveraging the standardized negotiation and record-layer security of TLS. Moreover, TLS can offer an initial server-authenticated channel to protect the privacy of account information, such as user name, transmitted between client and server. Here we discuss possible schemes for composing OPAQUE and TLS. We consider TLS 1.3 [41] as the upcoming and more secure version of TLS although some of the mechanisms can be implemented via prior versions of TLS.

The simplest TLS-OPAQUE combination is one where U's private key p_U stored by OPAQUE at S is used as a signature key for TLS client authentication. In this case, the OPAQUE-extended handshake protocol includes the following sequential steps (for a total of 5 messages): (i) a 1-RTT run of TLS 1.3 handshake protocol that produces a session key authenticated by S's TLS certificate; (ii) the two OPAQUE messages exchanged between client and server excluding the KE values g^x, g^y (these were already exchanged as part of the TLS 1-RTT run); (iii) TLS 1.3 client authentication using U's private signature key p_U retrieved from S in step (ii).

These steps result in mutual authentication where server's authentication is accomplished based on a TLS certificate. The client can either trust such a certificate or it can verify equality of the certificate's public key against P_S as retrieved by OPAQUE. In case of a mismatch the client can request a signature of S using P_S which is computed on the TLS transcript[8]. In the latter case, the protocol does not rely on PKI certificates except for protecting account information. *In all cases, the security of passwords and password authentication does not rely on PKI but on OPAQUE only.*

Variants of the above scheme include the use of a TLS 1.3 0-RTT exchange for sending the first OPAQUE message (including protected account information)

[8] Such additional server authentication and the client authentication in step (iii) can be implemented using TLS exported authenticators as defined in [45] (client authentication in this case corresponds to post-handshake authentication in [41]).

in which case steps (i) and (ii) are executed concurrently for a total of two messages (two flights in TLS jargon). This variant, while more efficient, relies on 0-RTT which is available only to clients and servers that have previously shared a key (negotiated in a previous handshake). A 0-RTT variant independent of pre-shared keys and based instead on a server's public key is possible (e.g., [35]) but it is not standardized by TLS 1.3. Finally, if protecting the secrecy of user's account information is not considered necessary then steps (i) and (ii) can run concurrently (without using the 0-RTT scheme); in this case server's authentication is based on the server's stored key P_S. This setting also allows for a maximally efficient scheme using HMQV as illustrated in Figure 9 (with additional key derivation and record layer processing based on TLS).

We note that the security of the above variants and composition rely on the modularity of OPAQUE that can compose the OPRF steps with arbitrary key-exchange protocols (with KCI security). We remark that the security of TLS 1.3 has been analyzed in multiple works (cf. [17–19, 21, 30, 36]) with client authentication via exported authentication (or "post-handshake authentication") studied in [34].

6.3 An OPAQUE variant: Multiplicative blinding

A variant of OPAQUE is obtained by replacing the user's exponential blinding operation $\alpha := (H'(\mathsf{pw}))^r$ with $\alpha := (H'(\mathsf{pw})) \cdot g^r$. The server responds as before with $\beta = \alpha^{k_s}$. Assuming that U knows the value $y = g^{k_s}$ (previously stored or received from S), it can compute the same "hashed Diffie-Hellman" value $(H'(\mathsf{pw}))^{k_s}$ as β / y^r. The advantage of this variant is that while the number of client exponentiations remains the same, one is fixed-base (g^r) and the other (y^r) can also be fixed-base if U caches y, a realistic possibility for accounts where the user logs in frequently (e.g., a personal email or social network). Computing y^r can also be done while waiting for the server's response to reduce latency. Moreover, both exponentiations can be done offline although only short-term storage is recommended as the leakage of r exposes $H'(\mathsf{pw})$ (hence opens pw to a dictionary attack). If U does not store y, it needs to be transmitted to U by S together with the response β. This still allows for fixed-base optimization for computing g^r but not for y^r. Note that the two OPAQUE variants compute the same value rw, hence an implementation can support both and leave it up to the client to choose one (and request y from S if needed).

However, it turns out that this multiplicative mechanism results in an OPRF protocol that does *not* realize our OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$. Thus, our analysis here does not imply the security of the multiplicative OPAQUE variant in general. If rw is redefined as $\mathsf{rw} := H(\mathsf{pw}, y, H'(\mathsf{pw})^{k_s})$, i.e. if y is included under the hash, then the resulting OPRF does realize our functionality, and OPAQUE remains secure as SaPAKE under both blinding variants. This change, however, introduces a (slight) overhead of having to transmit y even if the client implements the exponential blinding operation. An alternative approach would be to replace the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ with a weaker form $\mathcal{F}'_{\mathsf{OPRF}}$ and to show that (i) $\mathcal{F}'_{\mathsf{OPRF}}$ is realized by the multiplicative

variant (even without hashing y) and (ii) $\mathcal{F}'_{\mathsf{OPRF}}$ is sufficient for proving Theorem 2 hence implying the security of OPAQUE as SaPAKE. We intend to investigate this weakening of $\mathcal{F}_{\mathsf{OPRF}}$.

Acknowledgments. We thank Clemens Hlauschek for helpful comments and suggestions.

References

1. CFRG, Crypto Forum Research Group, `https://datatracker.ietf.org/rg/cfrg/documents/`.
2. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Topics in Cryptology – CT-RSA 2005*, pages 191–208. Springer, 2005.
3. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3), 2003.
4. 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.
5. 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.
6. S. M. Bellovin and M. Merritt. 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*, pages 244–250. ACM, 1993.
7. F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. *IACR Cryptology ePrint Archive*, 2013:833, 2013.
8. X. Boyen. Halting password puzzles. In *Usenix Security Symposium – SECURITY 2007*, pages 119–134. The USENIX Association, 2007.
9. X. Boyen. HPAKE: Password authentication secure against cross-site user impersonation. In *Cryptology and Network Security – CANS 2009*, pages 279–298. Springer, 2009.
10. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology – EUROCRYPT 2000*, pages 156–171. Springer, 2000.
11. 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.
12. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology – EUROCRYPT 2005*, pages 404–421. Springer, 2005.
13. 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.
14. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology – EUROCRYPT 2002*, pages 337–351. Springer, 2002.

15. D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In *Advances in Cryptology – EUROCRYPT 2008*, pages 127–145. Springer, 2008.
16. S. Contini. Method to protect passwords in databases for web applications, 2015. Cryptology ePrint Archive, Report 2015/387.
17. C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of tls 1.3. In *ACM CCS 17*, 2017.
18. C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *IEEE S&P 2016.*, 2016.
19. B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM CCS*, 2015. Also, Cryptology ePrint Archive, Report 2015/914.
20. P. Farshim, C. Orlandi, and R. Rosie. Security of symmetric primitives under incorrect usage of keys. *IACR Transactions on Symmetric Cryptology*, 2017(1):449–473, 2017.
21. M. Fischlin and F. Gnther. Replay attacks on zero round-trip time: The case of the tls 1.3 handshake candidates. In *IEEE S&P 2017*, 2017.
22. M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography – TCC 2005*, pages 303–324. Springer, 2005.
23. 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.
24. L. Gong, M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.
25. S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):230–268, 1999.
26. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *Advances in Cryptology – ASIACRYPT 2014*, pages 233–253. Springer, 2014.
27. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. 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*, pages 276–291. IEEE, 2016.
28. 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.
29. C. S. Jutla and A. Roy. Smooth NIZK arguments with applications to asymmetric UC-PAKE. *IACR Cryptology ePrint Archive*, 2016:233, 2016.
30. B. B. K. Bhargavan and N. Kobeissi. Verified models and reference implementations for the tls 1.3 standard candidate. In *IEEE S&P 2017*, 2017.
31. B. Kaliski. PKCS# 5: Password-based cryptography specification version 2.0. 2000.
32. F. Kiefer and M. Manulis. Zero-knowledge password policy checks and verifier-based PAKE. In *Computer Security – ESORICS 2014*, pages 295–312. Springer, 2014.
33. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol (extended abstract). In *Advances in Cryptology – CRYPTO 2005*, page 546. Springer, 2005.

34. H. Krawczyk. Unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in tls 1.3). In *ACM CCS 2016*, 2016.
35. H. Krawczyk and H. Wee. The OPTLS protocol and TLS 1.3. In *EuroS&P*, 2016.
36. X. Li, J. Xu, Z. Zhang, D. Feng, , and H. Hu. Multiple handshakes security of tls 1.3 candidates. In *IEEE S&P 2016.*, 2016.
37. P. Mackenzie. More efficient password-authenticated key exchange. In *Topics in Cryptology – CT-RSA 2001*, pages 361–377. Springer, 2001.
38. P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In *Advances in Cryptology – ASIACRYPT 2000*, pages 599–613. Springer, 2000.
39. D. Pointcheval and G. Wang. VTB-peke: Verifier-based two-basis password exponential key exchange. In *ACM Asia Conference on Computer and Communications Security – AsiaCCS 2017*, pages 301–312. ACM, 2017.
40. N. Provos and D. Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.
41. E. Rescorla. The transport layer security (TLS) protocol version 1.3 (draft 24), Feb. 2018.
42. J. Schmidt. Requirements for password-authenticated key agreement (PAKE) schemes. Technical report, 2017.
43. S. Shin and K. Kobara. Augmented password-authenticated key exchange (AugPAKE). *draft-irtf-cfrg-augpake-08*.
44. S. Shin, K. Kobara, and H. Imai. Security proof of AugPAKE. *IACR Cryptology ePrint Archive*, 2010:334, 2010.
45. N. Sullivan. Exported authenticators in tls (draft 4), Oct. 2018.

A The DH-OPRF Protocol Realizing Revised $\mathcal{F}_{\mathsf{OPRF}}$

Figure 11 shows the DH-OPRF protocol of [27] (who calls it 2HashDH), syntactically modified to realize functionality $\mathcal{F}_{\mathsf{OPRF}}$, see Fig. 2 in Section 3. Recall that the $\mathcal{F}_{\mathsf{OPRF}}$ functionality we show in Section 3 is a revision of the OPRF functionality defined in [27], with the most important difference being modeling adaptive corruptions. The protocol shown below is essentially the same as in [27], and requires the same One-More Diffie-Hellman assumption [3, 27] for security.

We defer the proof of the following Lemma 1 to the full version because it is very similar to the proof of security given in [27].

Lemma 1. *The DH-OPRF protocol shown in Fig. 11 UC-realizes the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ under the One-More Diffie Hellman assumption in ROM.*

Modifications in the Proof of [27]. We briefly discuss how our modifications to $\mathcal{F}_{\mathsf{OPRF}}$ influence the security proof, and leave the detailed proof to the full version of this paper.

Since no message is sent to $\mathcal{A}^*$ in the Initialization phase, adding Initialization has no impact on simulation. Allowing for sub-sessions (identified by *ssid*) results in adding *ssid* in the simulator whenever appropriate. The impact of changing SndrComplete messages as sent from $\mathcal{Z}$, instead of from

Components: Hash functions $H(\cdot,\cdot)$, $H'(\cdot)$ with ranges $\{0,1\}^\ell$ and G, respectively.

Initialization

- On input (INIT, sid, x), S picks $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ and outputs (INIT, $sid, H(x, H'(x)^k)$).

Evaluation

- On input (EVAL, $sid, ssid, \mathsf{S}, x$), U proceeds as follows:
  - If there is a record $\langle \mathsf{S}, x, r, y\rangle$, outputs (EVAL, $sid, ssid, y$) to $\mathcal{Z}$.
  - Else if there is a record $\langle \mathsf{S}', x, r, y\rangle$ (where $\mathsf{S}' \neq \mathsf{S}$), sends $a := H'(x)^r$ to S.
  - Else picks $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, records $\langle \mathsf{S}, x, r, \cdot\rangle$ and sends $a := H'(x)^r$ to S.
- On input (SNDRCOMPLETE, $sid, ssid$) and a from U, S sends $b := a^k$ to U.
- On b from S, if this is the first such message for $ssid$, U retrieves record $\langle \mathsf{S}, x, r, \cdot\rangle$, replaces $\cdot$ with $y := H_2(x, b^{1/r})$ and outputs (EVAL, $sid, ssid, y$).

Fig. 11: Protocol DH-OPRF (for PRF output length ℓ)

$\mathcal{A}^*$, is that no such messages are sent from SIM any more in steps 6 and 7; however, this does not influence the reduction that Pr[HALT] is negligible, since the only SNDRCOMPLETE messages which count are those in step 5, which are still there (the only difference is that their issuers become $\mathcal{Z}$ instead of SIM, but they still have the effect of increasing the tx value).

The remaining change is that $\mathcal{A}$ may compromise a server (for a specific sid) at any time; after that, $\mathcal{A}$ can compute the server's function value on any valid input. SIM is able to simulate this by sending OFFLINEEVAL messages to $\mathcal{F}$. Furthermore, note that HALT may only occur on servers who is not marked COMPROMISED at that time; therefore, the argument upper-bounding Pr[HALT] (in the setting where a server cannot be compromised) is not influenced.

1. Pick $r_1, \ldots, r_N \leftarrow_{\mathrm{R}} \mathbb{Z}_m$, and compute $g_1 := g^{r_1}, \ldots, g_N := g^{r_N}$. Record $(r_1, g_1), \ldots, (r_N, g_N)$. Set counter $J := 1$.
2. Every time when there is a fresh query x to $H'(\cdot)$, answer it with g_J and record (x, r_J, g_J). Finally, set $J{+}{+}$.
3. On (COMPROMISE, *sid*) from $\mathcal{A}$ as a message to S, mark S COMPROMISED, send (COMPROMISE, *sid*) to $\mathcal{F}$ and do:
   - If there is no record $\langle \mathsf{S}, k, z \rangle$, pick $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, compute $z := g^k$, record $\langle \mathsf{S}, k, z \rangle$ and send k to $\mathcal{A}$ as S's response.
   - Else retrieve $\langle \mathsf{S}, k, z \rangle$ and send k to $\mathcal{A}$ as S's response.
4. On (EVAL, *sid*, *ssid*, U, S) from $\mathcal{F}$, send g_J to $\mathcal{A}$ as U's message to S and record $\langle ssid, \mathsf{U}, \mathsf{S}, r_J, g_J \rangle$. Finally, set $J{+}{+}$.
5. On (SNDRCOMPLETE, *sid*, *ssid*, S) from $\mathcal{F}$ and a from $\mathcal{A}$ as some user U's message to S, do:
   - If there is no record $\langle \mathsf{S}, k, z \rangle$, pick $k \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, compute $z := g^k$, record $\langle \mathsf{S}, k, z \rangle$ and send a^k to $\mathcal{A}$ as S's response to U.
   - Else retrieve $\langle \mathsf{S}, k, z \rangle$ and send a^k to $\mathcal{A}$ as S's response to U.
6. On b from $\mathcal{A}$ as some server S's message to a user U, retrieve record $\langle ssid, \mathsf{U}, \cdot, r_j, g_j \rangle$ and do:
   - If there is a record $\langle \mathsf{S}', \cdot, z \rangle$ such that $b^{1/r_j} = z$, send (RCVCOMPLETE, *sid*, *ssid*, S') to $\mathcal{F}$.
   - Else create a new server S', record $\left\langle \mathsf{S}', \cdot, b^{1/r_j} \right\rangle$ and send (RCVCOMPLETE, *sid*, *ssid*, S') to $\mathcal{F}$.
7. Every time when there is a fresh query (x, u) to $H(\cdot, \cdot)$, do:
   (a) If there is a record (x, r_j, g_j), do:
      (1) If there is a record $\langle \mathsf{S}, k, z \rangle$ such that $u^{1/r_j} = z$ and S is marked COMPROMISED, send (OFFLINEEVAL, *sid*, S, x) to $\mathcal{F}$.
      On $\mathcal{F}$'s response (OFFLINEEVAL, *sid*, y), set $H(x, u) := y$.
      (2) Else if there is a record $\langle \mathsf{S}, \cdot, z \rangle$ such that $u = z^{r_j}$ and S is not marked COMPROMISED, send (EVAL, *sid*, *ssid*, S, x) and then (RCVCOMPLETE, *sid*, *ssid*, S) to $\mathcal{F}$.
      If $\mathcal{F}$ ignores this message, output HALT and abort.
      Otherwise on $\mathcal{F}$'s response (EVAL, *sid*, *ssid*, y), set $H(x, u) := y$.
   (b) In any other case, set $H(x, u) \leftarrow_{\mathrm{R}} \{0,1\}^l$.

Fig. 12: The Simulator SIM for the DH-OPRF Protocol ($\mathcal{F}_{\mathsf{OPRF}}$ Denoted $\mathcal{F}$)