Universally Composable Relaxed Password Authenticated Key Exchange

Michel Abdalla[1,2], Manuel Barbosa[3], Tatiana Bradley[4], Stanisław Jarecki[4], Jonathan Katz[5], Jiayu Xu[5]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@ens.fr
[2] INRIA, Paris, France
[3] FCUP and INESC TEC, Porto, Portugal
mbb@fc.up.pt
[4] University of California, Irvine, USA
tebradle@uci.edu, sjarecki@uci.edu
[5] Dept. of Computer Science, George Mason University, USA
jkatz2@gmail.com, jxu27@gmu.edu

Abstract. Protocols for *password authenticated key exchange* (PAKE) allow two parties who share only a weak password to agree on a cryptographic key. We revisit the notion of PAKE in the universal composability (UC) framework, and propose a relaxation of the PAKE functionality of Canetti et al. that we call *lazy-extraction PAKE* (lePAKE). Our relaxation allows the ideal-world adversary to *postpone* its password guess until after a session is complete. We argue that this relaxed notion still provides meaningful security in the password-only setting.

As our main result, we show that several PAKE protocols that were previously only proven secure with respect to a "game-based" definition of security can be shown to UC-realize the lePAKE functionality in the random-oracle model. These include SPEKE, SPAKE2, and TBPEKE, the most efficient PAKE schemes currently known.

1 Introduction

Protocols for *password authenticated key exchange* (PAKE) allow two parties who share only a weak password to agree on a cryptographically strong key by communicating over an insecure network. PAKE protocols have been studied extensively in the cryptographic literature [9,8,10,16,26,13,14], and are compelling given the widespread use of passwords for authentication. Even though the current practice is to implement password-based authentication by using TLS to set up a secure channel over which the password is sent, there are many arguments in favor of using PAKE protocols in conjunction with TLS [23]. Continued interest in PAKE is indicated by the fact that several PAKE protocols are currently under active consideration for standardization by the IETF [29].

Defining security for PAKE protocols is made challenging by the fact that a password shared by the parties may have low entropy, and so can be guessed by an adversary with noticeable probability. This must somehow be accounted for in any security definition. Roughly speaking, the security guaranteed by a PAKE protocol is that an attacker who initiates Q *online* attacks—i.e., actively interferes in Q sessions of the protocol—can make at most Q password guesses (i.e., at most one per session in which it interferes) and can succeed in impersonating a party only if one of those guesses was correct. In particular, this means that *offline* attacks, in which an adversary merely eavesdrops on executions of the protocol, should not help the adversary in any way.

Two paradigms of PAKE security. In the cryptographic literature there are two leading paradigms for defining the above intuition. The first is the so-called "game-based" definition introduced by Bellare et al. [8]. Here, a password is chosen from a distribution with min-entropy κ, and the security experiment considers an interaction of an adversary with multiple instances of the PAKE protocol using that password. A PAKE protocol is considered secure if no probabilistic polynomial-time (PPT) attacker can distinguish a real session key from a random session key with advantage better than $Q \cdot 2^{-\kappa}$ plus a negligible quantity.

A second approach uses a "simulation-based" definition [10,13]. The most popular choice here is to work in the universal composability (UC) framework [12], and this is what we assume here. This approach works by first defining an appropriate ideal functionality for PAKE; a PAKE protocol is then considered secure if it realizes that functionality in the appropriate sense. Canetti et al. [13] pursued this approach, and defined a PAKE functionality that explicitly allows an adversary to make password guesses; a random session key is generated unless the adversary's password guess is correct. As argued by Canetti et al. [13], this approach has a number of advantages. A definition in the UC framework is better suited for handling general correlations between passwords, e.g., when a client uses unequal but related passwords with different servers, or when an honest party uses different but closely related passwords due to mistyping. It also ensures security under arbitrary protocol composition, which is useful for arguing security of protocols that use PAKE as a subroutine, e.g., for converting symmetric PAKE to asymmetric PAKE [15,21] or strong asymmetric PAKE [23]. This is especially important in the context of PAKE standardization, because strong asymmetric PAKE protocols can strengthen the current practice of password-over-TLS authentication while achieving optimal security against server compromise.

Is there an inherent price for simulation-based security? Simulation-based security for PAKE is a desirable target. Unfortunately, the current state-of-the-art [13,28,25,11] suggests that this notion is more difficult to satisfy than the game-based definition. In particular, the most efficient UC PAKE protocol [11] is roughly a factor of two less efficient than the most efficient game-based PAKEs[1] such as SPEKE [22,31,20], SPAKE2 [7], or TBPEKE [33].

[1] Variants of EKE [9] shown to be universally composable [4,11] may appear to be exceptions, but EKE requires an ideal cipher defined over a cryptographic group, and it is not clear how that can be realized efficiently.

Perhaps surprisingly, we show here that this "gap" can be overcome; in particular, we show that *the SPEKE, SPAKE2, and TBPEKE protocols—which were previously only known to be secure with respect to the game-based notion of security—can be shown to be universally composable* (in the random-oracle model). The caveat is that we prove universal composability with respect to a relaxed version of the PAKE functionality originally considered by Canetti et al. At a high level, the main distinction is that the UC PAKE functionality of Canetti et al. [13] requires an attacker conducting an online attack against a session to make its password guess *before* that session is completed, whereas the relaxed functionality we consider—which we call *lazy-extraction PAKE* (lePAKE)—allows the attacker to delay its password guess until *after* the session completes. (However, the attacker is still limited to making a single password guess per actively attacked session.) On a technical level, this relaxed functionality is easier to realize because it allows the simulator to defer extraction of an attacker's password guess until a later point in the attacker's execution (see further discussion below). Nevertheless, the lazy-extraction PAKE functionality continues to capture the core properties expected from a PAKE protocol. In particular, as a sanity check on the proposed notion, we show that lePAKE plus key confirmation satisfies the game-based notion of PAKE with *perfect forward secrecy* (PFS) [8,5,6].

Implications for PAKE standardization. Recently, the Crypto Forum Research Group (CFRG), an IRTF (Internet Research Task Force) research group focused on applications of cryptographic mechanisms, initiated a PAKE selection process with the goal of providing recommendations for password-based authenticated key establishment for the IETF. Originally, four candidates were under consideration by the CRFG in the symmetric PAKE category; the final decision was between SPAKE2 and CPace, and the latter was ultimately selected. Our results validate the security of SPAKE2 and the proof we provide for TBPEKE will be adapted to cover CPace and included in the full version [2].

1.1 Technical Overview

The fundamental reason for an efficiency gap between known protocols achieving game-based PAKE and simulation-based PAKE is that the UC PAKE functionality, as defined by Canetti et al. [13] and used in all subsequent work, requires the adversary's password guesses to be (straight-line) *extractable* from the adversary's messages to the honest parties. Recall that for a PAKE to be UC secure there must exist an efficient simulator which simulates PAKE protocol instances given access to the ideal PAKE functionality, which in particular requires the simulator to specify a unique explicit password guess for each PAKE instance which the real-world adversary actively attacks. (The ideal PAKE functionality then allows the simulator, and hence the real-world adversary, to control the session key output by this instance *if* the provided password guess matched the password used by that PAKE instance, and otherwise the session key is random and thus secure.) The fact that the simulator must specify this explicit password before the attacked PAKE instance terminates, requires the simulator to online extract the password guess committed to in adversary's messages. Moreover, this extraction must be performed straight-line, because universal composibility prohibits rewinding the adversary.

Unfortunately, online extraction cannot be done for many efficient game-based PAKE's listed above, because in these protocols each party sends a single protocol message which forms a *perfectly hiding* commitment to the password. Specifically, if g generates group of prime order p then in SPAKE2 each party sends a message of the form $X = g^z \cdot (P_i)^{pw}$ where $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, P_1, P_2 are random group elements in the CRS, and $i = 1$ or 2 depending on the party's role. In TBPEKE this message has the form $X = (P_1 \cdot (P_2)^{H(pw)})^z$, and in SPEKE it is $X = H(pw)^z$ where H is a hash onto the group. These commitments are binding under the discrete logarithm hardness assumption (the first two are variants of Pedersen commitment [32], the third one requires ROM), and they are equivocable, i.e. the simulator can "cheat" on messages sent on behalf of the honest parties, but they are perfectly hiding and thus not extractable. These commitments can be replaced with extractable ones, but it is not clear how to do so without increasing protocol costs (or resorting to ideal ciphers over a group).

PAKE with post-execution input extraction. However, in all the above schemes the final session key is computed by hashing the protocol transcript and the Diffie-Hellman key established by this PAKE interaction, e.g. $Z = g^{z_1 \cdot z_2}$ in SPAKE2 or $Z = (H(pw))^{z_1 \cdot z_2}$ in SPEKE. Since this final hash is modeled as a Random Oracle, an adversary who learns any information on the session key must query this RO hash on the proper input. If the information in this hash query suffices for the simulator to identify the unique password to which this query corresponds, then a protocol message *together with the final hash inputs* do form an extractable commitment to the unique password guess the adversary makes on the attacked session.

However, the hash used in the final session key derivation is a local computation each party does in a "post-processing" stage which can be executed after the counterpart terminates the protocol. Therefore a simulator which extracts a password guess from the adversary's protocol message(s) and this local hash computation might extract it after the attacked session terminates. By the rules of the PAKE functionality of Canetti et al. [13], such extraction would happen too late, because the PAKE functionality allows the simulator to test a password guess against a session but does so only when this session is still active (and has not been attacked previously e.g. on a different password guess). Indeed, it would seem counter-intuitive to allow the ideal-world adversary, i.e. the simulator, to provide the unique effective password guess *after* the attacked session completes. Nevertheless, this is exactly how we propose to relax the UC PAKE functionality in order to accommodate protocols where input-extraction is possible, but succeeds only from the adversary's post-processing computation.

The relaxation we propose, the *lazy-extraction* PAKE, will require the ideal-world adversary to "interrupt" a fresh session while it is active in order to then perform the post-execution password test (we will call such tests "late" password tests). This models the UC PAKE requirement that an adversary can use an honest PAKE session as a password-testing oracle only if it actively attacks that session, and in particular it still holds that passively observed sessions do not provide any avenue for an attack. (To keep the new elements of the lazy-extraction PAKE model clear we use separate terms, resp. RegisterTest and LateTestPwd, for this new type of online session interruption and for the late password test, see Section 2.) Moreover, even if the adversary chooses this "lazy-extraction attack" route, the functionality still allows for only a *single* password test on an actively attacked session. This requirement effectively commits a (computationally bounded) real-world adversary to a unique password guess on each actively attacked session, because an adversary who performs the local computation related to more than one password test would not be simulatable in the model where the ideal-world adversary can submit at most one such test to the lazy-extraction PAKE functionality.

Explicit authentication and perfect forward security. To test the proposed lazy-extraction UC PAKE notion we show two further things. First, we show that any lazy-extraction UC PAKE followed by a key confirmation round upgrades lazy-extraction UC PAKE to PAKE with explicit (mutual) authentication (PAKE-EA) [17], but it also realizes a stronger variant of lazy-extraction PAKE functionality which we call the *relaxed* UC PAKE. In the relaxed PAKE model, the adversary can still make a (single) late password test on an actively attacked session but such sessions are guaranteed to terminate with an abort. Hence, the attacker cannot use a late password test to compromise a session. Intuitively, if a lazy-extraction PAKE is followed by a key confirmation and the attacker delays its late password test until after the key confirmation is sent, then the key confirmation must fail and its counterpart will abort on such session. Hence, the "late password test" reveals if the tested passworded was correct but it cannot reveal a key of an active session.

Secondly, we show that any relaxed UC PAKE satisfies the game-based notion of PAKE with perfect forward secrecy (PFS) [8,5,6]. (A similar test was done by Canetti et al. with regard to the original UC PAKE notion [13].) Intuitively, since the lazy-extraction attack avenue against a relaxed PAKE cannot be used to compromise keys of any active session, it follows that all active sessions, i.e. all sessions which terminate with a session key as opposed to an abort, are as secure in the relaxed UC PAKE model as they are in the original UC PAKE model of Canetti et al. In particular, they are secure against future password compromise.

Related and concurrent work. Jarecki et al. [24] recently introduced the relaxed UC PAKE model in the context of the *asymmetric* PAKE (aPAKE) functionality [15], and showed that this relaxation is necessary to prove security of the OPAQUE protocol proposed in [23]. As discussed above, the lazy-extraction PAKE model goes further than the relaxed PAKE model, and this further relaxation appears to be necessary in order to model protocols like SPEKE, SPAKE2, and TBPEKE as universally composable PAKEs. (See Section 2 for the precise specifications of the lazy-extraction PAKE and the relaxed PAKE models.)

Hasse and Labrique [18] have recently argued that CPace [19] realizes a variant of the lazy-extraction UC PAKE functionality, but the variant of this notion they consider seems unsatisfactory, e.g. it appears not to imply security of passively observed sessions, and it appears to be not realizable as stated (see Section 2 for discussion). They also argue that adding a key-confirmation step suffices to convert such protocol into a standard UC PAKE, while we show that the result is still only a *relaxed* UC PAKE.[2]

In concurrent work, Shoup [34] analyzes the UC security of two variants of SPAKE2 in the symmetric (PAKE) and asymmetric (aPAKE) settings. Both variants include built-in key confirmation and the protocol flows are simplified so that only the initiator uses the password for *blinding* its first message. Shoup shows these protocols UC secure with respect to revised ideal functionalities for PAKE and aPAKE, under a slightly weaker assumption than the one required by our modular proof, namely strong Diffie-Hellman [3] instead of Gap CDH. (Strong DH is a variant of Gap CDH where the DDH oracle can be queried on triples whose first element is fixed.) The revised UC PAKE functionality considered in [34] appears equivalent to the relaxed UC PAKE functionality which we show is realized by SPAKE2 with key confirmation.

1.2 Paper Overview

In Section 2, we introduce the two relaxations of the UC PAKE functionality, namely the lazy-extraction UC PAKE and relaxed UC PAKE functionalities, respectively abbreviated as lePAKE and rPAKE, together with the extension of the latter to explicit (mutual) authentication. In Section 3, we show that SPAKE2 scheme of [7] is a secure lePAKE under the Gap CDH assumption. In Section 4, we show that any lePAKE protocol followed by a key confirmation round is a secure rPAKE-EA, i.e. rPAKE with explicit authentication. In Section 5, we show that every rPAKE-EA protocol satisfies the game-based notion of PAKE with perfect forward secrecy, and that every lePAKE protocol by itself already satisfies weak forward secrecy. In Section 6, we also include the proof that TBPEKE [33] is a secure lePAKE protocol under appropriate assumptions, and we explain that this proof extends to similar results regarding SPEKE [22,31,20].

2 Relaxations of UC PAKE

In Fig. 1, we present the PAKE functionality as defined by Canetti et al. [13], and compare it with two relaxations that we refer to as *relaxed PAKE* (rPAKE) and *lazy-extraction PAKE* (lePAKE). We explain at a high level the differences between these various formulations. In the original PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$, after a party initiates a session (but before the party generates a key) the attacker may try to guess the password used in that session by making a single

[2] In [18] this is explicitly claimed not for CPace itself but for its asymmetric (aPAKE) version called AuCPace.

TestPwd query. If the attacker's password guess is correct, the session is marked compromised; if not, the session is marked interrupted. When a session key is later generated for that session, the attacker is given the ability to choose the key if the session is marked compromised, but a random key is chosen otherwise. Importantly, the attacker is only allowed to make a password guess for a session *before* the key is generated and the session terminates.

Session initiation

On $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role})$ from $\mathcal{P}$, ignore this query if record $\langle sid, \mathcal{P}, \cdot, \cdot, \cdot \rangle$ already exists. Otherwise record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$ marked fresh and send $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', \mathsf{role})$ to $\mathcal{A}$.

Active attack

– On $(\mathsf{TestPwd}, sid, \mathcal{P}, pw^*)$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \cdot \rangle$ then:
  - If $pw^* = pw$ then mark it compromised and return "correct guess";
  - If $pw^* \neq pw$ then mark it interrupted and return "wrong guess".

– On $(\mathsf{RegisterTest}, sid, \mathcal{P})$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', \cdot, \cdot \rangle$ then mark it interrupted and flag it tested.

– On $(\mathsf{LateTestPwd}, sid, \mathcal{P}, pw^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \cdot, K \rangle$ marked completed with flag tested then remove this flag and do:
  - If $pw^* = pw$ then return $\boxed{K}$ "correct guess" to $\mathcal{A}$;
  - If $pw^* \neq pw$ then return $\boxed{K^{\$} \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}}$ "wrong guess" to $\mathcal{A}$.

Key generation

On $(\mathsf{NewKey}, sid, \mathcal{P}, K^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$ not marked completed then do:

– If the record is compromised, or either $\mathcal{P}$ or $\mathcal{P}'$ is corrupted, then set $K := K^*$.
– If the record is fresh and $\exists$ a completed record $\langle sid, \mathcal{P}', \mathcal{P}, pw, \mathsf{role}', K' \rangle$ with $\mathsf{role}' \neq \mathsf{role}$ that was fresh when $\mathcal{P}'$ output (sid, K'), then set $K := K'$.
– In all other cases pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.

Finally, append K to record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$, mark it completed, and output (sid, K) to $\mathcal{P}$.

Fig. 1. UC PAKE variants: The original PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$ of Canetti et al. [13] is the version with all gray text omitted. The relaxed PAKE functionality $\mathcal{F}_{\mathsf{rPAKE}}$ includes the gray text but omits the boxed portions; the lazy-extraction PAKE functionality $\mathcal{F}_{\mathsf{lePAKE}}$ includes the gray text but omits the dashed portions.

In both the relaxed PAKE functionality $\mathcal{F}_{\mathsf{rPAKE}}$ and the lazy-extraction PAKE functionality $\mathcal{F}_{\mathsf{lePAKE}}$, the attacker is given the ability to make a password guess for a session even *after* a session key is generated and that session has completed. Formally, this is allowed only if the attacker makes a RegisterTest query before the session key is generated; this indicates the attacker's intention to (possibly) make a password guess later, and models active interference with a real-world protocol execution. (Of course, the attacker also has the option of making a password guess before a key is generated as in the original $\mathcal{F}_{\mathsf{PAKE}}$.) Having made a $\mathsf{RegisterTest}$ query for a session, the attacker may then make a $\mathsf{LateTestPwd}$ query to that session after the session key K is generated. $\mathcal{F}_{\mathsf{rPAKE}}$ and $\mathcal{F}_{\mathsf{lePAKE}}$ differ in what happens next:

First, in $\mathcal{F}_{\mathsf{rPAKE}}$, the attacker is only told whether or not its password guess is correct, but learns nothing about K in either case. Secondly, in $\mathcal{F}_{\mathsf{lePAKE}}$, the attacker is given K if its password guess is correct, and given a random key otherwise.[3]

It is easy to see that both $\mathcal{F}_{\mathsf{rPAKE}}$ and $\mathcal{F}_{\mathsf{lePAKE}}$ are relaxations of $\mathcal{F}_{\mathsf{PAKE}}$, in the sense that any protocol realizing $\mathcal{F}_{\mathsf{PAKE}}$ also realizes $\mathcal{F}_{\mathsf{rPAKE}}$ and $\mathcal{F}_{\mathsf{lePAKE}}$. Although, as defined, $\mathcal{F}_{\mathsf{lePAKE}}$ and $\mathcal{F}_{\mathsf{rPAKE}}$ are incomparable, the version of $\mathcal{F}_{\mathsf{lePAKE}}$ in which the attacker is additionally notified whether its password guess is correct (cf. Footnote 3) is a strict relaxation of $\mathcal{F}_{\mathsf{rPAKE}}$.

Following the work of Groce and Katz [17], we also consider PAKE functionalities that incorporate explicit (mutual) authentication, which we refer to as PAKE-EA.[4] Intuitively, in a PAKE-EA protocol a party should abort if it did not establish a matching session key with its intended partner. As in the case of PAKE, the original PAKE-EA functionality introduced by Groce and Katz required the attacker to make its password guess before the session key is generated, while we introduce a relaxed version of the PAKE-EA functionality, denoted $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$ and shown in Fig. 2, that allows the attacker to delay its password guess until after the session has completed.[5] If the attacker's guess is correct, it is notified of that fact; our relaxation thus parallels that of $\mathcal{F}_{\mathsf{rPAKE}}$. Note that such late password guess can only be performed on aborted sessions, since the attacker must send a $\mathsf{RegisterTest}$ query before the session completes, which marks the session $\mathsf{interrupted}$, and by the rule of explicit authentication, an $\mathsf{interrupted}$ session must result in aborting.

Besides the intuitive appeal of our relaxed definitions, we justify these relaxations by showing that it is easy to realize $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$ in the $\mathcal{F}_{\mathsf{lePAKE}}$-hybrid world (Section 4), that any protocol realizing $\mathcal{F}_{\mathsf{rPAKE}}$ satisfies perfect forward se-

[3] Note that here the attacker is not explicitly notified whether its password guess is correct. While it is arguably more natural to notify the attacker, we obtain a slightly stronger functionality by omitting this notification.

[4] Although Canetti et al. [13] informally suggest a way of modeling explicit authentication in PAKE, the functionality they propose seems unacceptably weak in the sense that it does not require a party to abort even when an attacker successfully interferes with its partner's session.

[5] While relaxing the Groce-Katz functionality, we also make some minor changes to their original: (1) we make the parties symmetric, and do not require the server to generate a session key first, and (2) we allow the adversary to force a party to abort by sending a $(\mathsf{NewKey}, sid, \mathcal{P}, \perp)$ message. (This second modification is required, and its omission appears to be an oversight of Groce and Katz.)

Session initiation

On $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role})$ from $\mathcal{P}$, ignore this query if record $\langle sid, \mathcal{P}, \cdot, \cdot, \cdot \rangle$ already exists. Otherwise record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$ marked fresh and send $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', \mathsf{role})$ to $\mathcal{A}$.

Active attack

- On $(\mathsf{TestPwd}, sid, \mathcal{P}, pw^*)$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \cdot \rangle$ then:
  - If $pw^* = pw$ then mark it compromised and return "correct guess";
  - If $pw^* \neq pw$ then mark it interrupted and return "wrong guess".
- On $(\mathsf{RegisterTest}, sid, \mathcal{P})$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', \cdot, \cdot \rangle$ then mark it interrupted and flag it tested.
- On $(\mathsf{LateTestPwd}, sid, \mathcal{P}, pw^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \cdot, K \rangle$ marked completed with flag tested then remove this flag and do:
  - If $pw^* = pw$ then return "correct guess" to $\mathcal{A}$.
  - If $pw^* \neq pw$ then return "wrong guess" to $\mathcal{A}$.

Key generation and explicit authentication

- On $(\mathsf{GetReady}, sid, \mathcal{P})$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$ marked fresh then re-label it ready.
- On $(\mathsf{NewKey}, sid, \mathcal{P}, K^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$ not marked completed then do:
  - If the record is compromised, or $\mathcal{P}$ or $\mathcal{P}'$ is corrupted, or $K^* = \bot$, then set $K{:=}K^*$.
  - Else, if the record is fresh or ready, and $\exists$ a record $\langle sid, \mathcal{P}', \mathcal{P}, pw, \mathsf{role}' \rangle$ marked ready s.t. $\mathsf{role}' \neq \mathsf{role}$ then pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.
  - Else, if the record is ready and $\exists$ a completed record $\langle sid, \mathcal{P}', \mathcal{P}, pw, \mathsf{role}', K' \rangle$ with $\mathsf{role}' \neq \mathsf{role}$ that was fresh when $\mathcal{P}'$ output (sid, K'), then set $K := K'$.
  - In all other cases, set $K := \bot$.

  Finally, append K to record $\langle sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{role} \rangle$, mark it completed, and output (sid, K) to $\mathcal{P}$.

Fig. 2. The $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$ functionality for relaxed PAKE-EA. The original PAKE-EA of Groce and Katz [17] corresponds to the version with gray text omitted. The boxed text highlights the differences from $\mathcal{F}_{\mathsf{rPAKE}}$.

crecy (Section 5), and that any protocol realizing $\mathcal{F}_{\mathsf{lePAKE}}$ satisfies weak forward secrecy (Section 5.3).

Note on the relaxed PAKE functionality used in [18]. A preliminary version of the lazy-extraction PAKE functionality, referred as "relaxed PAKE" therein, appeared in an early version of [24] and was adopted by [18] as a model for the CPace protocol. This version was imprecise in several respects: First, it does not require the adversary to explicitly attack an online session via a RegisterTest query before issuing a LateTestPwd query on a completed session. This appears too weak, e.g. because it allows an adversary to issue LateTestPwd queries even on passively observed sessions. On the other hand, it restricts the adversary from making a LateTestPwd query upon completion of the matching counterpart's session (with a matching *sid* but not necessarily a matching password), which appears too strong, because a man-in-the-middle attacker can make $\mathcal{P}'$ complete with a random key or an abort, and this does not affect its capabilities regarding party $\mathcal{P}$. Our lazy-extraction PAKE functionality makes this notion more precise, and in Section 6 we show that TBPEKE [33] and SPEKE [22] realize the lePAKE fuctionality under (Gap) CDH and/or SDH assumptions. Since CPace [18,19] is a close variant of SPEKE, these results can be extended to cover CPace as well. [6]

3 Security of SPAKE2

We consider SPAKE2 as a motivating example for our work. SPAKE2 was proposed by Abdalla and Pointcheval [7] and shown secure in the game-based PAKE model [8] under the CDH assumption in the ROM. SPAKE2 is, to the best of our knowledge, the most efficient PAKE protocol which does not assume ideal cipher over a group. Its costs are 2 fixed-base and 1 variable-base exponentiations per party, and it is round-minimal because it can be executed in a single simultaneous round of bi-directional communication.

We show that SPAKE2 realizes the lazy-extraction UC PAKE functionality under the *Gap* CDH assumption, and the result is tight in the sense that any environment which distinguishes between the real-world execution of SPAKE2 and the ideal-world interaction with a simulator and the lazy-extraction PAKE functionality, and does so in time T with advantage ϵ, implies an attack on Gap CDH which achieves roughly the same (T, ϵ) advantage, where "roughly" means that both T and ϵ are modified by only additive factors. This UC security proof complements the result that SPAKE2 meets the game-based PFS definition [1], which was not considered in [7]. Interestingly, the game-based PFS result of [1] is not tight: The proof relies on a special assumption introduced in [7] for which

[6] In CPace [18], the key derivation hash includes only the session ID and the Diffie-Hellman key, while our proof of TBPEKE security assumes that it also includes party IDs, the password-dependent base, and the transcript. The final version of CPace selected by the CFRG has been updated to include all these elements except the password. In the full version [2], we will analyze the security of this version of CPace.

a reduction to Gap CDH is known, but it is not a tight reduction. Still, since we do not know that lazy-extraction UC PAKE security implies PFS security by itself, this result is the only one we currently know for PFS security of (raw) SPAKE2.[7]

We recall the two-flow, simultaneous round SPAKE2 protocol of [7] in Fig. 3, with some notational choices adjusted to the UC setting.

Public parameters: generator g and random elements M, N of group $\mathbb{G}$ of prime order p; hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$

$\mathcal{P}$ on $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{client})$:		$\mathcal{P}'$ on $(\mathsf{NewSession}, sid, \mathcal{P}', \mathcal{P}, pw', \mathsf{server})$:
$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$		$y \leftarrow_{\mathrm{R}} \mathbb{Z}_p$
$X \leftarrow g^x \cdot M^{pw}$		$Y \leftarrow g^y \cdot N^{pw'}$
	$\xrightarrow{X}$ $\xleftarrow{Y}$	
$\hat{Y} \leftarrow Y/N^{pw}$		$\hat{X} \leftarrow X/M^{pw'}$
$K \leftarrow \mathsf{H}(sid, \mathcal{P}, \mathcal{P}', X, Y, pw, \hat{Y}^x)$		$K' \leftarrow \mathsf{H}(sid, \mathcal{P}, \mathcal{P}', X, Y, pw', \hat{X}^y)$
output K		output K'

Fig. 3. SPAKE2 protocol of [7]

Theorem 1. $\mathsf{SPAKE2}$ *realizes the Lazy-Extraction PAKE functionality* $\mathcal{F}_{\mathsf{lePAKE}}$ *in ROM under the Gap-CDH assumption.*

Gap CDH and Gap DL assumptions. Recall that the Computational Diffie-Hellman (CDH) assumption states that, given generator g and two random elements $A = g^a$, and $B = g^b$ in a cyclic group of prime order, it is hard to find $C = \mathsf{DH}_g(A, B) = g^{ab}$, while the Discrete Logarithm (DL) assumption states that it is hard to find $a = \mathsf{DL}_g(A)$ given (g, A), for random A. In the gap version of either assumption, the respective problem must remain hard even if the adversary has access to a Decisional Diffie-Hellman oracle, which on any triple of group elements (A, B, C) returns 1 if $C = \mathsf{DH}_g(A, B)$ and 0 otherwise. The Gap DL assumption follows via a trivial (and tight) reduction from the Gap CDH assumption, but we introduce it to highlight the fact that certain forms of adversarial behavior in SPAKE2 imply solving the harder problem of Gap DL.

Simulator for SPAKE2. The UC simulator SIM for SPAKE2, given in full in Fig. 4, acts as the ideal adversary, with access to the ideal functionality $\mathcal{F}_{\mathsf{lePAKE}}$ (shortened to $\mathcal{F}$ in the subsequent discussion). The simulator's goal is to emulate, except for at most negligible probability, the real-world interaction between the environment $\mathcal{Z}$, a real-world adversary $\mathcal{A}$, and honest parties running the

[7] Note that the combined results of Sections 3 to 5 show that SPAKE2 *followed by a key confirmation round* is PFS secure, with tight security with respect to Gap CDH.

SPAKE2 protocol. Technically, SIM must simulate messages that appear to come from the real players, respond appropriately to adversarial messages, and answer random oracle queries consistently, and to do so with access to the $\mathcal{F}$ interface, but not the secret inputs, i.e. the passwords, of the honest players.

We briefly describe how SIM simulates client $\mathcal{P}$'s interaction with an arbitrary environment $\mathcal{Z}$ and an adversary **Adv**. (The server case is similar since the protocol is symmetric.) SIM first embeds trapdoors into the CRS, i.e., it picks $m, n \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and sets $M = g^m$ and $N = g^n$. To simulate the protocol message X, SIM picks $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and sets $X = g^z$. Since X is also uniformly distributed in the real protocol, the environment cannot tell the difference. When $\mathcal{A}$ queries the random oracle $\mathsf{H}(sid, \mathcal{P}, \mathcal{P}', X, Y', pw, W)$, SIM decides whether it corresponds to a valid password guess: SIM first computes the exponent $\hat{x}$ such that $X = g^{\hat{x}} \cdot M^{pw}$, using the CRS trapdoor m, and then checks if $W = (Y'/N^{pw})^{\hat{x}}$. If so, then SIM stores (Y', pw). If $\mathcal{A}$ later sends a protocol message Y' aimed at $\mathcal{P}$, then this is an online attack: when $\mathcal{A}$ makes the RO query, SIM picks a random string K as the output, and stores K together with (Y', pw). Then, when $\mathcal{A}$ sends Y', SIM sends $(\mathsf{TestPwd}, sid, \mathcal{P}, pw)$ to $\mathcal{F}$, and if $\mathcal{F}$ replies "correct guess," then SIM sets $\mathcal{P}$'s key to K by sending $(\mathsf{NewKey}, sid, \mathcal{P}, K)$ to $\mathcal{F}$. (Otherwise, i.e. if $\mathcal{F}$ replies "wrong guess", SIM sends $(\mathsf{NewKey}, sid, \mathcal{P}, 0^\kappa)$ and $\mathcal{F}$ sets $\mathcal{P}$'s key to a random string.) On the other hand, if $\mathcal{A}$ makes an above query to H *after* sending Y' to the client, then this is a postponed attack, so SIM sends $(\mathsf{RegisterTest}, sid, \mathcal{P})$ to $\mathcal{F}$ when $\mathcal{A}$ sends Y' for which no such query has been made yet, and later sends $(\mathsf{LateTestPwd}, sid, \mathcal{P}, pw)$ to $\mathcal{F}$ when $\mathcal{A}$ makes the above query to H. If $\mathcal{F}$ then replies with a key K (which is either correct or random, depending on whether pw is correct or not), SIM "programs" H output as K. An adversary could distinguish this emulation from a real interaction by querying H on $(sid, \mathcal{P}, \mathcal{P}', X, Y', pw_i, W_i)$ tuples for $W_i = (Y'/N^{pw_i})^{\hat{x}_i}$ and $\hat{x}_i = \mathsf{DL}(X/M^{pw_i})$ for *two* different passwords pw_i, but we show that this attack, as well as all others, can be reduced to Gap CDH.

Record keeping. For each party $\mathcal{P}$ and session sid, simulator SIM stores a state $\pi_{\mathcal{P}}^{sid} = (\mathsf{role}, \mathsf{exp}, \mathcal{C}, \mathcal{S}, X, Y, X^*, Y^*, \mathsf{pw}, \mathsf{guesses}, \mathsf{waiting})$ whose components are used as follows:

- Variable $\mathsf{role} \in \{\mathsf{client}, \mathsf{server}\}$ is the role of $\mathcal{P}$ in this session. (Note that in the protocol of Fig. 3 variable role is used only in the ordering of the identities in the hash H query.)
- exp is the private exponent used in the network messages, x for $\mathcal{C}$ and y for $\mathcal{S}$. In the first few games, it has the same meaning as in the protocol, but for SIM it is the discrete log of the simulated network message.
- $\mathcal{C}, S$ are the the party party identifiers for client and server respectively, X, Y are the simulated messages sent by resp. $\mathcal{C}$ and $\mathcal{S}$ (on sessions identified by sid), and X^*, Y^* are the messages the adversary sends to resp. $\mathcal{S}$ and $\mathcal{C}$. Simulator SIM stores messages X and Y^* for the client (copying Y from the server if it exists), and symmetrically it stores Y and X^* for the server (copying X from the client if it exists). Unknown values are set to $\perp$.

- pw is a password used by party $\mathcal{P}$ on session sid. It is used only in intermediate games, while simulator SIM always sets it to $\perp$.
- $\mathsf{guesses}$ is a table mapping group elements Z^* to pairs (pw, K^*), representing potential password guesses and corresponding keys, which the simulator constructs from adversary's queries to oracle H of the form $(sid, \mathcal{P}, \mathcal{P}', X, Z^*, \mathsf{pw}, \cdot)$ if $\mathcal{P}$ plays the client role, and $(sid, \mathcal{P}', \mathcal{P}, Y, Z^*, \mathsf{pw}, \cdot)$ if $\mathcal{P}$ plays the server role. If the adversary sends Z^* to party $\mathcal{P}$, the simulator looks up the corresponding password pw, which it sends to $\mathcal{F}$ as a tested password.
- $\mathsf{waiting}$ is a flag which is set to $\mathtt{T}$ for the session which has not received an adversarial message Z^*, and $\mathtt{F}$ otherwise. This flag is used to ensure that only the first message an adversary sends to a session is processed, and all others are ignored.

Let $\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{SPAKE2}}$ be the probability of the event that environment $\mathcal{Z}$ with adversary $\mathcal{A}$ outputs 1 in the real world, and $\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathsf{SPAKE2}}$ be the corresponding probability in the ideal world. The goal is to show that $|\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{SPAKE2}} - \mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathsf{SPAKE2}}|$ is negligible. We use the well-known sequence-of-games proof strategy to show that we may move from the real game to the simulator in a manner indistinguishable to the environment, except for negligible probability. We begin with Game 0, the real game, and move through a series of steps, each of which we show to be indistinguishable from the previous, to the final simulator. Throughout the proof, **Gi** denotes the probability that $\mathcal{Z}$ outputs 1 while interacting with Game i.

Proof of Theorem 1.

Game 0. This is the real world, in which $\mathcal{A}$ interacts with real players, and may view, modify, and/or drop network messages.

$$\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{SPAKE2}} = \Pr\left[\mathbf{G0}\right].$$

Game 1. (Simulate real world with trapdoors) We now simulate the behavior of the real players and the random oracle. The simulation is exactly as the real game, except for the inclusion of record keeping, and embedding of trapdoors in M and N, i.e, setting $M = g^m; N = g^n$ for known m and n. The embedding of trapdoors is not noticeable to the environment as M, N are still drawn from from the same distribution as before, thus:

$$\Pr\left[\mathbf{G1}\right] = \Pr\left[\mathbf{G0}\right].$$

Game 2. (Random key if adversary is passive) If the adversary passes a simulated Z message sent to $(sid, \mathcal{P})$ without modification, output a random key for $\mathcal{P}$ instead of the true random oracle output. The environment notices this change only if $\mathcal{A}$ makes a hash query that would result in an inconsistency, namely $\mathsf{H}(sid, \mathcal{C}, \mathcal{S}, X', Y', pw, W = g^{xy})$, where $X' = g^x M^{pw}, Y' = g^y N^{pw}$ are the simulated messages. We check for such queries, and abort if any occur.

We may reduce this event to Gap-CDH as follows. Let q_s be the maximum number of sessions invoked by the environment. Consider an adversary

```
generate CRS
$M \leftarrow g^m; N \leftarrow g^n$ for $(m, n) \leftarrow_{\mathrm{R}} \mathbb{Z}_p$
return $M, N$

on (NewSession, $sid, \mathcal{P}, \mathcal{P}'$, role) from $\mathcal{F}$
if $\pi_{\mathcal{P}}^{sid} \neq \bot$: return $\bot$
$(X, Y) \leftarrow (\bot, \bot)$
if role = client :
    $(\mathcal{C}, \mathcal{S}) \leftarrow (\mathcal{P}, \mathcal{P}')$
    $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$; $X \leftarrow M^z$; $Z \leftarrow X$
    if $\pi_{\mathcal{P}'}^{sid} \neq \bot$ and $\pi_{\mathcal{P}'}^{sid}$.role $\neq$ client: $Y \leftarrow \pi_{\mathcal{P}'}^{sid}.Y$; $\pi_{\mathcal{P}'}^{sid}.X \leftarrow X$
else if role = server :
    $(\mathcal{C}, \mathcal{S}) \leftarrow (\mathcal{P}', \mathcal{P})$
    $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$; $Y \leftarrow N^z$; $Z \leftarrow Y$
    if $\pi_{\mathcal{P}'}^{sid} \neq \bot$ and $\pi_{\mathcal{P}'}^{sid}$.role $\neq$ server: $X \leftarrow \pi_{\mathcal{P}'}^{sid}.X$; $\pi_{\mathcal{P}'}^{sid}.Y \leftarrow Y$
$\pi_{\mathcal{P}}^{sid} \leftarrow$ (role, $z, \mathcal{C}, \mathcal{S}, X, Y, \bot, \bot, \bot, \bot$, T)
send $Z$ from $\mathcal{P}$ to $\mathcal{A}$

on $Z^*$ from $\mathcal{A}$ as msg to $(sid, \mathcal{P})$
if $\pi_{\mathcal{P}}^{sid} = \bot$ or $\pi_{\mathcal{P}}^{sid}$.waiting = F: return $\bot$
(role, $\cdot, \mathcal{C}, \mathcal{S}, X, Y, \cdot, \cdot, \cdot$, guesses, $\cdot$) $\leftarrow \pi_{\mathcal{P}}^{sid}$
$K \leftarrow 0^\kappa$
if role = client :
    $\pi_{\mathcal{P}}^{sid}.Y^* \leftarrow Z^*$
    if $Z^* = Y$: jump to end
else if role = server :
    $\pi_{\mathcal{P}}^{sid}.X^* \leftarrow Z^*$
    if $Z^* = X$: jump to end
if $\pi_{\mathcal{P}}^{sid}$.guesses$[Z^*] = (pw, K^*)$:
    reply $\leftarrow$ (TestPwd, $sid, \mathcal{P}, pw$) to $\mathcal{F}$
    if reply = "correct": $K \leftarrow K^*$
else: send (RegisterTest, $sid, \mathcal{P}$) to $\mathcal{F}$
end: $\pi_{\mathcal{P}}^{sid}$.waiting $\leftarrow$ F
     send (NewKey, $sid, \mathcal{P}, K$) to $\mathcal{F}$

on H$(sid, \mathcal{C}, \mathcal{S}, X', Y', pw, W)$ from $\mathcal{A}$:
if $\mathsf{T}_{\mathsf{H}}[sid, \mathcal{C}, \mathcal{S}, X', Y', pw, W] = \bot$:
    $K \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$; $(\hat{x}, \hat{y}) \leftarrow (\bot, \bot)$
    if $\pi_{\mathcal{C}}^{sid} \neq \bot$: $\hat{x} \leftarrow m \cdot \pi_{\mathcal{C}}^{sid}$.exp $- m \cdot pw$
    if $\pi_{\mathcal{S}}^{sid} \neq \bot$: $\hat{y} \leftarrow n \cdot \pi_{\mathcal{S}}^{sid}$.exp $- n \cdot pw$
    if $\pi_{\mathcal{C}}^{sid}.X = X'$ and $\pi_{\mathcal{S}}^{sid}.Y = Y'$ and $W = g^{\hat{x}\hat{y}}$: abort
    else if $\pi_{\mathcal{C}}^{sid}.X = X'$ and $W = (Y'/N^{pw})^{\hat{x}}$ :
        if $Y' = \pi_{\mathcal{C}}^{sid}.Y^*$: $\mathcal{P} \leftarrow \mathcal{C}$; jump to late_test_pw
        else: $\pi_{\mathcal{C}}^{sid}$.guesses$[Y'] \leftarrow (pw, K)$
    else if $\pi_{\mathcal{S}}^{sid}.Y = Y'$ and $W = (X'/M^{pw})^{\hat{y}}$:
        if $X' = \pi_{\mathcal{S}}^{sid}.X^*$: $\mathcal{P} \leftarrow \mathcal{S}$; jump to late_test_pw
        else: $\pi_{\mathcal{S}}^{sid}$.guesses$[X'] \leftarrow (pw, K)$
    jump to end
    late_test_pw: reply $\leftarrow$ (LateTestPwd, $sid, \mathcal{P}, pw$) to $\mathcal{F}$
                  $K \leftarrow$ reply; if no reply, abort
    end: $\mathsf{T}_{\mathsf{H}}[sid, \mathcal{C}, \mathcal{S}, X', Y', pw, W] \leftarrow K$
send $\mathsf{T}_{\mathsf{H}}[sid, \mathcal{C}, \mathcal{S}, X', Y', pw, W]$ to $\mathcal{A}$
```

Fig. 4. Simulator algorithm SIM for SPAKE2 security proof

$\mathcal{B}_2$ against Gap-CDH. On generalized CDH challenge[8] $(A_1 = g^{a_1}, \cdots, A_{q_s} = g^{a_{q_s}}, B_1 = g^{b_1}, \cdots, B_{q_s} = g^{b_{q_s}})$, the reduction indexes sessions $(sid, \mathcal{P}, \mathcal{P}')$, and embeds $X_i = A_i \cdot M^{pw_i}$, $Y_i = B_i \cdot N^{pw_i}$ when generating the simulated messages for the ith session. The reduction can re-use the code of **G2**, except for the cases where it requires the secret exponents a_i and b_i: (1) to generate $K \leftarrow \mathsf{H}(sid, \mathcal{C}, \mathcal{S}, X, Y, pw, \hat{Y}^{a_i}\{\text{or } \hat{X}^{b_i}\})$ and (2) to check for the "bad event" of an inconsistency in the hash response. To handle case (1), the reduction stores an additional value K for each session $(sid, \mathcal{C}, \mathcal{S})$ which is set randomly when either the reduction must handle case (1), or when $\mathcal{A}$ queries $\mathsf{H}(sid, \mathcal{C}, \mathcal{S}, X', Y', pw, W)$ such that the password is correct and $\mathsf{DDH}(X', Y', W)$ holds (checked via the DDH oracle): if either of these events happens again the same value of K is used. The check of case (2) can be done via the DDH oracle, i.e., by querying $\mathsf{DDH}(A_i, B_i, W)$: if the bad event occurs, $\mathcal{B}_2$ solves the CDH challenge with answer W. Thus:

$$|\Pr[\mathbf{G2}] - \Pr[\mathbf{G1}]| \leq \mathbf{Adv}^{\mathsf{GCDH}}_{\mathcal{B}_2}.$$

Game 3. (Random simulated messages) On $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', \mathsf{role})$, if this is the first $\mathsf{NewSession}$ for $(sid, \mathcal{P})$, set $Z \leftarrow g^z$ for $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, and send Z to $\mathcal{A}$ as a message from $\mathcal{P}$ to $(\mathcal{P}', sid)$. Note that we may now compute the original exponents via: $\hat{x} = m \cdot \pi^{sid}_{\mathcal{C}}.\mathsf{exp} - m \cdot pw$ and $\hat{y} = n \cdot \pi^{sid}_{\mathcal{S}}.\mathsf{exp} - n \cdot pw$. This change is not observable to the environment, as it is merely a syntactic change in the calculation of the exponents:

$$\Pr[\mathbf{G3}] = \Pr[\mathbf{G2}].$$

Game 4. (Random keys if adversary does not correctly guess password) We now detect when an adversarial hash query corresponds to a password guess. We can detect this event by inspecting the X', Y', pw and W values provided to the hash oracle. Let us assume the adversary is guessing the client's password (the server case is symmetric). To make a password guess against the client, the adversary must set $X' = \pi^{sid}_{\mathcal{C}}.X$, i.e., use the simulated message sent by the client. The adversary can use any choice of Y', but to correspond with a specific password guess, the following must hold: $W = (Y'/N^{pw})^{\hat{x}}$ (where $\hat{x}$ is the exponent such that $\pi^{sid}_{\mathcal{C}}.X = g^{\hat{x}} M^{pw}$). In other words, W must be the value that would be used by a real client if Y' were sent as the server's message. If such a password guess query is detected, we check if Y' was previously sent as an adversarial message on behalf of the server: if so, and if the password guess is correct, we program the random oracle to match the previously sent key. If Y' was not previously sent, we note the values Y', pw queried by the adversary and the random key K output by the RO. If Y' is later sent as the adversarial message, and the password is correct, we output the stored key K. If the password is incorrect, we output a random key independent of the RO table. If at any point a second password guess (correct or incorrect) is detected for the same sid and party, we abort the game.

[8] The generalized CDH problem is tightly equivalent to the CDH problem by random self-reducibility.

This change is noticeable to the environment only in the abort case, i.e, the case where the adversary makes two password guesses with a single (X', Y') transcript, i.e.:

$$\mathsf{H}(sid, \mathcal{P}, \mathcal{P}', X', Y', pw, Z) \text{ and } \mathsf{H}(sid, \mathcal{P}, \mathcal{P}', X', Y', pw', Z'),$$

such that $pw \neq pw'$, and

$$\mathsf{CDH}(g^{mx}/M^{pw}, g^{ny}/N^{pw}) = Z \text{ and } \mathsf{CDH}(g^{mx}/M^{pw'}, g^{ny}/N^{pw'}) = Z'.$$

Call this event bad_4. It can be split into two cases: 1) for one of the passwords $pw^* \in \{pw, pw'\}$ it holds that $X' = M^{pw^*}$ or $Y' = N^{pw^*}$, i.e., there is a collision between the guessed password pw^* and the secret exponent x or y or 2) there is no such collision. Case 2, which we denote bad_4^1, can be reduced to Gap-CDH, as shown in Lemma 1. Case 1 can be reduced to Gap-DL as follows: Adversary $\mathcal{B}_{4.2}$ on Gap DL challenge $A = g^a$, sets simulated messages as: $X_i = A \cdot g^{\Delta_{i,x}}$ and $Y_i = A \cdot g^{\Delta_{i,y}}$, picking a fresh random $\Delta_{i,x}$ and $\Delta_{i,y}$ for each session $(sid, \mathcal{P}, \mathcal{P}')$. In the ith session (for every i), $\mathcal{B}_{4.2}$ uses the DDH oracle to check for bad_4. If true, $\mathcal{B}_{4.2}$ further checks if $Y' = N^{pw^*}$ or $X' = M^{pw^*}$ for one of the passwords: in the former case this means that $Y' = N^{pw} = A \cdot g^{\Delta_{i,y}}$, so $npw = a + \Delta_{i,y}$, and $\mathcal{B}_{4.2}$ can output the DL solution is $a = npw/\Delta_{i,y}$, and the latter case is symmetric. We have that

$$|\Pr[\mathbf{G4}] - \Pr[\mathbf{G3}]| \leq \mathbf{Adv}^{\mathsf{GDL}}_{\mathcal{B}_{4.1}} + \mathbf{Adv}^{\mathsf{GCDH}}_{\mathcal{B}_{4.2}}.$$

Game 5. (Use $\mathcal{F}_{\mathsf{lePAKE}}$ interface) In the final game, we modify the challenger so that it uses the $\mathsf{RegisterTest}$, $\mathsf{TestPwd}$ and $\mathsf{LateTestPwd}$ interfaces to check passwords, and the NewKey interface to set keys. This is an internal change that is not noticeable to the environment, thus

$$\Pr[\mathbf{G5}] = \Pr[\mathbf{G4}].$$

In addition, this simulator perfectly mimics the ideal world except for the cases where it aborts, which we have already shown to happen with negligible probability, so:

$$\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathsf{SPAKE2}} = \Pr[\mathbf{G5}].$$

Thus the distinguishing advantage of $\mathcal{Z}$ between the real world and the ideal world is:

$$|\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathsf{SPAKE2}} - \mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{SPAKE2}}| \leq \mathbf{Adv}^{\mathsf{GCDH}}_{\mathcal{B}_2} + \mathbf{Adv}^{\mathsf{GDL}}_{\mathcal{B}_{4.1}} + \mathbf{Adv}^{\mathsf{GCDH}}_{\mathcal{B}_{4.2}},$$

which is negligible if Gap-CDH is hard. □

Lemma 1. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{4.1}$ (whose running time is linear in the running time of $\mathcal{A}$) such that:*

$$\Pr[\mathbf{G4} \to bad_4^1] \leq \mathbf{Adv}^{\mathsf{GCDH}}_{\mathcal{B}_{4.1}}$$

Proof. Consider an attacker $\mathcal{B}_{4.1}$ against GCDH. It receives a challenge $(M = g^m, N = g^n)$ and wants to find $\mathsf{CDH}(M, N) = g^{mn}$. The attacker emulates **G4**, except for setting the CRS values as M, N from the challenge instead of randomly. It uses the DDH oracle to carry out the three checks in the if/else if/else if structure of the hash response, and for detecting the bad event.

In particular, it detects the bad event bad_4^1 when it sees two hash queries (sid, X', Y', pw, Z) and (sid, X', Y', pw', Z') such that $pw \neq pw'$, and both of the following hold, where either x or y is known (i.e, chosen by the attacker as the exponent for a simulated message):

$$\mathsf{CDH}(g^{mx}/M^{pw}, g^{ny}/N^{pw}) = Z \tag{1}$$

$$\mathsf{CDH}(g^{mx}/M^{pw'}, g^{ny}/N^{pw'}) = Z' \tag{2}$$

The attacker can then solve for the CDH response, g^{nm}, as follows.

First, write $Z = g^z; Z' = g^{z'}$ for unknown $z, z' \in \mathbb{Z}_p$. Considering only the exponents in Eqs. (1) and (2), we have that:

$$m(x - pw) \cdot n(y - pw) = z \tag{3}$$

$$m(x - pw') \cdot n(y - pw') = z' \tag{4}$$

Assume that the attacker knows the exponent x (the other case is symmetric). Scaling Eqs. (3) and (4) by resp. $(x - pw')$ and $(x - pw)$, gives:

$$m(x - pw')(x - pw) \cdot n(y - pw) = z \cdot (x - pw') \tag{5}$$

$$m(x - pw)(x - pw') \cdot n(y - pw') = z' \cdot (x - pw) \tag{6}$$

Subtracting Eq. (6) from Eq. (5) allows us to remove the unknown y term:

$$mn(x - pw)(x - pw')(pw' - pw) = z \cdot (x - pw') - z' \cdot (x - pw) \tag{7}$$

Finally, we may solve for the desired CDH value:

$$g^{mn} = \left(Z^{(x-pw')} \cdot Z'^{(pw-x)}\right)^{1/(x-pw)(x-pw')(pw-pw')}$$

This is possible as long as we are not dividing by zero, i.e., if $pw \neq x$ and $pw' \neq x$, which is explicitly excluded in the definition of event bad_4^1 (see Case 1 of **G4** for handling of this case).

□

Remark on Gap CDH. The proof relies on the gap version of CDH, and it seems hard to prove security from the standard CDH assumption, because the Decision Diffie-Hellman oracle is used by CDH reductions to maintain consistency of answers to RO queries, and it is not clear how to ensure this consistency otherwise. This is also the case in the all the other PAKE protocols we consider in Section 6.

4 Adding Explicit Authentication

We will show that any protocol that securely realizes the lazy-extraction UC PAKE functionality $\mathcal{F}_{\mathsf{lePAKE}}$, followed by a key confirmation round, is a secure realization of the relaxed UC PAKE-EA functionality $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$. (See Section 2 for the definition of these functionalities.) This protocol compiler construction is shown in Fig. 5.

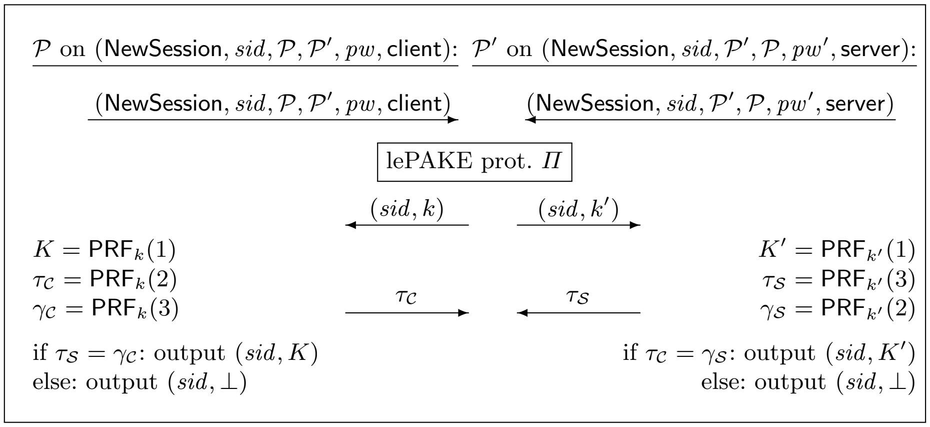

$\mathcal{P}$ on $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{client})$:
$\mathcal{P}'$ on $(\mathsf{NewSession}, sid, \mathcal{P}', \mathcal{P}, pw', \mathsf{server})$:
$(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{client})$
$(\mathsf{NewSession}, sid, \mathcal{P}', \mathcal{P}, pw', \mathsf{server})$
lePAKE prot. Π
(sid, k)
(sid, k')
$K = \mathsf{PRF}_k(1)$
$\tau_{\mathcal{C}} = \mathsf{PRF}_k(2)$
$\gamma_{\mathcal{C}} = \mathsf{PRF}_k(3)$
$\tau_{\mathcal{C}}$
$\tau_{\mathcal{S}}$
$K' = \mathsf{PRF}_{k'}(1)$
$\tau_{\mathcal{S}} = \mathsf{PRF}_{k'}(3)$
$\gamma_{\mathcal{S}} = \mathsf{PRF}_{k'}(2)$
if $\tau_{\mathcal{S}} = \gamma_{\mathcal{C}}$: output (sid, K)
else: output $(sid, \perp)$
if $\tau_{\mathcal{C}} = \gamma_{\mathcal{S}}$: output (sid, K')
else: output $(sid, \perp)$

Fig. 5. Compiler from lePAKE protocol Π to rPAKE-EA protocol Π'.

Theorem 2. *Protocol Π' shown in Fig. 5 realizes the Relaxed PAKE-EA functionality $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$ if Π realizes the Lazy-Extraction PAKE functionality $\mathcal{F}_{\mathsf{lePAKE}}$ and* PRF *is a secure PRF.*

Figure 6 shows the simulator used in the proof of Theorem 2. (For notational simplicity, we denote functionality $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$ as simply $\mathcal{F}$.) For the formal proof of this theorem we refer to the full version of this paper [2], but here we provide an informal overview.

The proof is essentially a case-by-case argument, where all possible scenarios are divided into several cases, according to whether $\mathcal{A}$ performs an online attack on party $\mathcal{P}$'s rPAKE session, and if so, whether it is an online attack or a postponed attack on $\mathcal{P}$'s lePAKE session.[9] Below we describe the cases, and how each case can be simulated:

Case 1 (online attack on lePAKE → online attack on rPAKE): $\mathcal{A}$ sends $(\mathsf{TestPwd}, sid, \mathcal{P}, pw^*)$ to $\mathcal{F}_{\mathsf{lePAKE}}$ when $\mathcal{P}$'s lePAKE session is fresh.

[9] Note that each rPAKE session runs a lePAKE session as a subprotocol, and these two sessions should not be confused. Similarly, each party has a lePAKE output (which is always a string) and an rPAKE output (which is either a string derived from its lePAKE output or $\perp$).

In this case SIM passes $(\mathsf{TestPwd}, sid, \mathcal{P}, pw^*)$ to $\mathcal{F}$. If $\mathcal{F}$ replies with "correct guess", i.e., $pw^* = pw$, then $\mathcal{P}$'s rPAKE session is $\mathsf{compromised}$, so SIM can set its rPAKE output. Now on $(\mathsf{NewKey}, sid, k^*)$ from $\mathcal{A}$, SIM can compute (K, τ, γ) from $k = k^*$ as the real-world session would, and send τ to $\mathcal{A}$; on a tag τ^* from $\mathcal{A}$, SIM lets $\mathcal{P}$ output K if $\tau^* = \gamma$ and $\perp$ otherwise, matching the real-world execution.

On the other hand, if $\mathcal{F}$ replies with "wrong guess", i.e., $pw^* \neq pw$, then $\mathcal{P}$'s rPAKE session is $\mathsf{interrupted}$, hence $\mathcal{P}$'s rPAKE output is $\perp$. SIM simply computes (K, τ, γ) from a random k and sends τ to $\mathcal{A}$. This again matches the real-world execution with overwhelming probability: $\mathcal{P}$'s rPAKE output is $\perp$ unless $\tau^* = \gamma$, which happens with negligible probability, and τ is computed in the exact same way.

Case 2 (postponed attack on lePAKE → online attack on rPAKE): $\mathcal{A}$ sends $(\mathsf{RegisterTest}, sid, \mathcal{P})$ when $\mathcal{P}$'s lePAKE session is fresh, followed by $(\mathsf{NewKey}, sid, \mathcal{P}, k^*)$; and then sends $(\mathsf{LateTestPwd}, sid, \mathcal{P}, pw^*)$ to $\mathcal{F}_{\mathsf{lePAKE}}$ (cauing $\mathcal{P}$'s rPAKE session to complete) *before* sending a tag τ^* to $\mathcal{P}$.

This is essentially the same with Case 1, except that the order of $\mathsf{TestPwd}/\mathsf{LateTestPwd}$ and NewKey to $\mathcal{F}_{\mathsf{lePAKE}}$ is reversed. SIM can reverse the order of these two queries on the rPAKE level too, by sending a $\mathsf{RegisterTest}$ message to $\mathcal{F}$ first.

Case 3 (postponed attack on lePAKE → postponed attack on rPAKE): This is the complementary case of Case 2, i.e., $\mathcal{A}$ sends $(\mathsf{RegisterTest}, sid, \mathcal{P})$ when $\mathcal{P}$'s lePAKE session is fresh, followed by $(\mathsf{NewKey}, sid, \mathcal{P}, \star)$; and then a tag τ^* to $\mathcal{P}$ *before* sending any $(\mathsf{LateTestPwd}, sid, \mathcal{P}, pw^*)$ message to $\mathcal{F}_{\mathsf{lePAKE}}$. (Eventually $\mathcal{A}$ may or may not send $\mathsf{LateTestPwd}$.)

Again, in the real world $\mathcal{P}$'s rPAKE output is $\perp$ unless $\tau^* = \gamma$, which happens with negligible probability, so SIM can simply let $\mathcal{P}$ output $\perp$. However, if $pw^* = pw$, $\mathcal{A}$ learns $\mathcal{P}$'s lePAKE output k (and thus can check if tag τ is the "correct" one, i.e., τ is derived from k using PRF). This can be simulated as follows: on NewKey from $\mathcal{A}$, SIM computes (K, τ, γ) from a random k and sends τ to $\mathcal{A}$; on τ^* from $\mathcal{A}$, SIM sends $\mathsf{RegisterTest}$ to $\mathcal{F}$; on $(\mathsf{LateTestPwd}, sid, \mathcal{P}, pw^*)$ from $\mathcal{A}$, SIM passes this message to $\mathcal{F}$, and if $\mathcal{F}$ replies with "correct guess", then SIM sends k to $\mathcal{A}$, making τ the "correct" tag. (If $\mathcal{F}$ replies with "wrong guess", then SIM sends a fresh random key to $\mathcal{A}$.)

Case 4 (no attack on lePAKE): $\mathcal{A}$ sends neither a $(\mathsf{TestPwd}, sid, \mathcal{P}, \star)$ nor a $(\mathsf{RegisterTest}, sid, \mathcal{P})$ query to $\mathcal{F}_{\mathsf{lePAKE}}$ when $\mathcal{P}$'s lePAKE session is fresh (thus $\mathcal{P}$'s lePAKE session remains fresh until it becomes $\mathsf{completed}$).

In this case, $\mathcal{A}$ never learns $\mathcal{P}$'s lePAKE output k, so SIM can send a random tag τ to $\mathcal{A}$. If $\mathcal{A}$ merely passes the tags between $\mathcal{P}$ and $\mathcal{P}'$, then SIM let $\mathcal{P}$ complete its rPAKE session by sending $(\mathsf{GetReady}, sid, \mathcal{P})$ and then $(\mathsf{NewKey}, sid, \mathcal{P}, 0^\kappa)$ to $\mathcal{F}$; if $\mathcal{P}$ and $\mathcal{P}'$'s passwords match, then $\mathcal{P}$ outputs a random K, otherwise $\mathcal{P}$ outputs $\perp$. On the other hand, if $\mathcal{A}$ modifies the tag from $\mathcal{P}'$ to $\mathcal{P}$, then it is not the "correct" tag of $\mathcal{P}$, so SIM lets $\mathcal{P}$ output $\perp$.

```
On (NewSession, $sid, \mathcal{P}, \mathcal{P}'$, role) from $\mathcal{F}$
If there is no record $\langle sid, \mathcal{P}, \ldots \rangle$ then:
    Send (NewSession, $sid, \mathcal{P}, \mathcal{P}'$, role) to $\mathcal{A}$ and store $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{role} \rangle$ marked fresh.
On (TestPwd, $sid, \mathcal{P}, pw^*$) from $\mathcal{A}$
If there is a fresh record $\langle sid, \mathcal{P}, \ldots \rangle$ then:
    Send (TestPwd, $sid, \mathcal{P}, pw^*$) to $\mathcal{F}$;
    If $\mathcal{F}$ replies "correct guess" then pass it to $\mathcal{A}$ and mark this record compromised.
    If $\mathcal{F}$ replies "wrong guess" then pass it to $\mathcal{A}$ and mark this record interrupted.
On (RegisterTest, $sid, \mathcal{P}$) from $\mathcal{A}$
If there is a fresh record $\langle sid, \mathcal{P}, \ldots \rangle$ then:
    Mark this record interrupted and flag it tested.
On (NewKey, $sid, \mathcal{P}, k^*$) from $\mathcal{A}$
If there is a record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{role} \rangle$ not completed then:
    Define $k$ as follows:
        If this record is compromised, or $\mathcal{P}$ or $\mathcal{P}'$ is corrupted, then set $k := k^*$.
        Else if this record is interrupted (tested or not), then set $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.
        Else set $k := \perp$.
    If $k \neq \perp$ then:
        If role = client then set $\tau := \mathsf{PRF}_k(2)$ and $\gamma := \mathsf{PRF}_k(3)$;
        If role = server then set $\tau := \mathsf{PRF}_k(3)$ and $\gamma := \mathsf{PRF}_k(2)$;
    Else, i.e., if $k = \perp$, pick $\tau \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$, set $\gamma := \perp$, and send (GetReady, $sid, \mathcal{P}$) to $\mathcal{F}$;
    Mark record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{role} \rangle$ "completed with key $k$ and tag $\gamma$";
    Send $\tau$ to $\mathcal{A}$ as the authenticator from $\mathcal{P}$ to $\mathcal{P}'$.
On delivery of an authenticator from $\mathcal{A}$
If record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{role} \rangle$ is marked "completed with key $k$ and tag $\gamma$", and $\mathcal{A}$ sends a purported authenticator, denoted $\tau^*$, to protocol instance $(sid, \mathcal{P})$ then:
    If $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{role} \rangle$ is flagged tested:
        Send (RegisterTest, $sid, \mathcal{P}$) to $\mathcal{F}$.
    If $k \neq \perp$ then:
        If $\tau^* = \gamma$ then send (NewKey, $sid, \mathcal{P}, \mathsf{PRF}_k(1)$) to $\mathcal{F}$.
        If $\tau^* \neq \gamma$ then send (NewKey, $sid, \mathcal{P}, \perp$) to $\mathcal{F}$.
    If $k = \perp$ (i.e., the record was fresh right before it became completed) then:
        If there is a completed record $\langle sid, \mathcal{P}', \mathcal{P}, \mathsf{role}' \rangle$ for $\mathsf{role}' \neq \mathsf{role}$, which was
            marked fresh right before it became completed, and which sent out
            authenticator $\tau'$ s.t. $\tau^* = \tau'$, then send (NewKey, $sid, \mathcal{P}, 0^{\kappa}$) to $\mathcal{F}$.
        Else send (NewKey, $sid, \mathcal{P}, \perp$) to $\mathcal{F}$.
On (LateTestPwd, $sid, \mathcal{P}, pw^*$) from $\mathcal{A}$
If there is a record $\langle sid, \mathcal{P}, \ldots \rangle$ marked "completed with key $k$", and flagged tested:
    Remove the tested flag from this record;
    If $\mathcal{A}$ did not send an authenticator to protocol instance $(sid, \mathcal{P})$ then send
        (TestPwd, $sid, \mathcal{P}, pw^*$) to $\mathcal{F}$;
    Else send (LateTestPwd, $sid, \mathcal{P}, pw^*$) to $\mathcal{F}$;
    If $\mathcal{F}$ replies "correct guess" then send $k$ to $\mathcal{A}$.
    If $\mathcal{F}$ replies "wrong guess" then send $k^{\$} \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$ to $\mathcal{A}$.
```

Fig. 6. Simulation for relaxed PAKE-EA protocol in Figure 5.

Compiler from PAKE to PAKE with entity authentication. If we replace the lazy-extraction PAKE functionality with the (standard) PAKE, then the same compiler construction realizes the (standard) PAKE with explicit authentication functionality. In other words, by dropping the "laziness" of the underlying PAKE protocol, we get a compiler from PAKE to PAKE with explicit authentication. While technically not a corollary of Theorem 2, it is clear that the proof of Theorem 2 can be slightly modified to prove this conclusion: In that proof, the simulator SIM sends a password test (i.e., send a $\mathsf{LateTestPwd}$ message to $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$) only if $\mathcal{A}$ does so (i.e., sends $\mathsf{LateTestPwd}$ message aimed at $\mathcal{F}_{\mathsf{lePAKE}}$ played by SIM); therefore, if both SIM and $\mathcal{A}$ are not allowed to do a late password test, the simulation will still succeed.

While it is well known that PAKE plus "key confirmation" yields PAKE with explicit authentication, to the best of our knowledge, there has been no proof of this fact in the UC setting.

SPAKE2 with key confirmation. An immediate corollary of Theorems 1 and 2 is that SPAKE2 with key confirmation realizes the relaxed UC PAKE-EA functionality $\mathcal{F}_{\mathsf{rPAKE\text{-}EA}}$ under the Gap-CDH assumption in ROM.

5 PAKE Relaxations and PFS

In this section we prove that any protocol that realizes the Relaxed PAKE functionality satisfies the standard game-based notion of security for PAKE protocols offering perfect forward secrecy (PFS). This is an important sanity check for the definition, as it shows that the extra power given to the ideal-world adversary by the late test feature does not weaken the security guarantee for PAKE sessions that are completed before passwords are corrupted. We show that a similar argument can be used to show that the weaker Lazy-Extraction PAKE definition implies a weak form of PFS, referred to as weak FS, where security in the presence of password leakage is only guaranteed with respect to passive attackers [27,30].

5.1 Defining PFS

We recall the standard game-based notion of security for PAKE protocols and which follows from a series of works [5,6] that refined the security notion proposed by Bellare, Pointcheval and Rogaway in [8]. Section 3 and [1] include the full details.

The definition is based on an experiment in which a challenger emulates a scenario where a set of parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, each running an arbitrary number of PAKE sessions, relies on a trusted setup procedure to establish pre-shared long-term (low-entropy) passwords for pairwise authentication. Passwords for each pair $(\mathcal{P}_i, \mathcal{P}_j)$ are sampled from a distribution over a dictionary $\mathcal{D}$; we assume here the case where $\mathcal{D}$ is any set of cardinality greater than one, and each password is sampled independently and uniformly at random from this set.[10] A ppt adversary $\mathcal{A}$ is challenged to distinguish established session keys from truly random ones with an advantage that is better than password guessing.

The security experiment goes as follows. The challenger first samples passwords for all pairs of parties, generates any global public parameters (CRS) that the protocol may rely on and samples a secret bit b. The challenger manages a set of instances π_i^j, each corresponding to the state of session instance j at party P_i, according to the protocol definition. The adversary is then executed with the CRS as input; it may interact with the following set of oracles, to which it may place multiple adaptive queries:

Execute: Given a pair of party identities $(\mathcal{P}_i, \mathcal{P}_j)$ this oracle animates an honest execution of a new PAKE session established between the two parties and returns the communications trace to the attacker. This gives rise to two new session instances π_i^k and π_j^l, which for correct protocols will have derived the same established session key.

Send: Given a party identity P_i, an instance j and a message m, this oracle processes m according to the state of instance π_i^j (or creates this state if the instance was not yet initialized) and returns any outgoing messages to the attacker.

Corrupt: Given a pair of party identities $(\mathcal{P}_i, \mathcal{P}_j)$, this oracle returns the corresponding pre-shared password.

Reveal: Given a party identity $\mathcal{P}_i$ and an instance j, this oracle checks π_i^j and, if this session instance has completed as defined by the protocol, the output of the session (usually either a secret key or an abort symbol) is returned to the attacker.

RoR: Given a party identity $\mathcal{P}_i$ and an instance j, this oracle checks π_i^j and, if this session instance has completed as defined by the protocol *and* this session instance is *fresh*, the adversary is challenged on guessing bit b: if $b = 0$ then the derived key is given to the attacker; otherwise a new random key is returned.[11]

[10] This assumption is standard for the corruption model captured by this game-based definition. If correlated passwords were allowed, then corrupting one password might reveal information that allows the attacker to trivially infer another one; preventing trivial attacks in this setting leads to a definition in the style of [8], where the corruption of a password must invalidate RoR queries associated with all correlated passwords; this means the whole dictionary if no restrictions are imposed on the distribution. The finer-grained definition of password corruption we adopt here does not easily extend to the case of arbitrary correlations between passwords. See [1] for a discussion. The UC definition covers arbitrary password sampling distributions and the results we prove in this section should extend to any reasonable game-based definition that deals with more complex password distributions. This is clearly the case for the concrete distributions discussed in [1].

[11] We use RoR (Real-or-Random) for this oracle rather than the standard Test oracle designation to avoid confusion with the test and late test requests that are included in the UC PAKE ideal functionality definitions.

Eventually the adversary terminates and outputs a guess bit b'. The definition of advantage excludes trivial attacks via the notion of session freshness used in the RoR oracle. Formal definitions are given in [1], here we give an informal description. Two session instances are *partnered* if their views match with respect to the identity of the peer, exchanged messages and derived secret keys—the first two are usually interpreted as a session identifier. A session is fresh if: a) the instance completed; b) the instance was not queried to Ror or Reveal before; c) at least one of the following four conditions holds: i. the instance accepted during a query to Execute; ii. there exists more than one partner instance; iii. no partner instance exists and the associated password was not corrupted prior to completion; iv. a unique fresh partner instance exists (implies not revealed).

A PAKE protocol is secure if, for any ppt attacker interacting with the above experiment and placing at most q_s queries to the Send oracle, we have that

$$|\Pr[b' = b] - 1/2| \leq q_s/|\mathcal{D}| + \epsilon\,,$$

where ϵ is a negligible term.

The original definition proposed by Bellare, Pointcheval and Rogaway [8] allows for stronger corruption models—fixing the corrupt password maliciously and revealing the internal state of session instances—which we do not consider. We also do not deal with the asymmetry between client and server (also known as augmented PAKE).

Known results for UC PAKE. Canetti et al. [13] introduced the notion of UC-secure PAKE and proved that this definition implies game-based security of the protocol as defined in [8]. Our proof that Relaxed PAKE implies game-based PAKE with PFS follows along the same lines and relies on two auxiliary results from that original proof that we recover here; the first result concerns a generic mechanism for the handling of session identifiers called *SID-enhancement* and the second one is a general result for security against eavesdroppers.

Given a two-party protocol Π, its SID-enhancement Π' is defined as the protocol that has the parties exchange nonces and then uses the concatenation of these nonces as SID. This transformation converts any protocol Π that assumes SIDs provided by an external environment as the means to define matching sessions, into another one that generates the SID on-the-fly as required by the syntax of the game-based security definition. Both the original proof and the one we give here show that the UC security of Π implies the game-based security of Π'. Intuitively, an environment simulating the PFS-game above can wait until the SID for the enhanced protocol is defined before calling NewSession to initiate the session of the parties in the UC setting.

For security against eavesdroppers, Canetti et al. show that no successful ideal world adversary can place TestPwd queries on sessions for which the environment $\mathcal{Z}$ instructed the adversary to pass messages between the players unmodified (i.e., to only eavesdrop on the session). We give here the intuition on why this is the case and refer the interested reader to [13] for a detailed proof.

The crucial observation is that, for eavesdropped sessions, the ideal-world adversary generates all the trace and hence has no side information on the password; this means that for every environment $\mathcal{Z}$ for which the ideal-world attacker might place such a query, there exists an environment $\mathcal{Z}'$ that can *catch* the simulator; $\mathcal{Z}'$ operates as $\mathcal{Z}$, but it uses a high-entropy password for the problematic session: in the real-world a session the two honest parties will end-up with matching keys with probability 1—one assumes perfect correctness here for simplicity—whereas an ideal-world adversary placing a $\mathsf{TestPwd}$ can never match the same behaviour. Indeed, querying a wrong password to $\mathsf{TestPwd}$ leads to mismatching keys with overwhelming probability in the ideal world and the ideal-world adversary cannot guess the password correctly except with small probability.

This argument extends trivially to $\mathsf{LateTestPw}$ queries, as these must be preceded by a $\mathsf{RegisterTest}$ query prior to session completion that also leads to mismatching keys with overwhelming probability. Furthermore, the above reasoning also applies when the ideal-world adversary may have the extra power of simulating an ideal object, i.e., the UC-secure PAKE protocol is defined in an $\mathcal{F}$-hybrid model. Indeed, whatever environment $\mathcal{Z}$ may have leaked to the ideal-world adversary via calls to $\mathcal{F}$, there exists an environment $\mathcal{Z}'$ that catches $\mathcal{S}$ as above.

5.2 Relaxed PAKE Implies PFS

Theorem 3. *Let $\mathcal{F}$ be an ideal object such as a random-oracle or an ideal-cipher. If Π securely realizes $\mathcal{F}_{\mathsf{rPAKE}}$ without explicit authentication, in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F})$-hybrid model, then its SID-enhanced version Π' is PFS-secure according to the game-based definition given above.*

We refer to the full version of this paper [2] for the proof of the above theorem, but this proof is quite similar to the proof of the corresponding theorem given in [13], which showed that the original notion of UC-secure PAKE implies game-based security (with PFS) of a PAKE protocol.

5.3 Lazy-Extraction PAKE Implies Weak FS

The proof of the above theorem can be adapted to show that any protocol that realizes the Lazy-Extraction UC PAKE functionality is secure under a weak form of game-based security: the attacker is not allowed to corrupt the passwords of sessions against which it launches an active attack. This notion of game-based security for PAKE is sometimes called weak FS. The proof of the following theorem follows similar lines as the proof of Theorem 3, and we include it in the full version of this paper [2].

Theorem 4. *Let $\mathcal{F}$ be an ideal object such as a random-oracle or an ideal-cipher. If Π securely realizes $\mathcal{F}_{\mathsf{lePAKE}}$ without explicit authentication, in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F})$-hybrid model, then its SID-enhanced version Π' is weak FS-secure.*

5.4 Practical implications

Putting together the results in Section 3 and Section 4 we obtain positive results for rPAKE secure protocols in the Universal Composability framework, namely SPAKE2, TBPEKE, CPace and SPEKE, followed by a round of key confirmation (although we did not give a detailed proof for the latter). The result in this section shows that all such protocols are also PFS secure in the game-based setting. The caveat here is that this proof involves modifying the protocol to deal with session identifiers: The UC PAKE model requires a unique session identifier as an input of the protocol, while in practice agreeing on such identifier before the protocol starts can add extra communication rounds to the protocol. For completeness, we include a direct proof that SPAKE2 with key confirmation provides game-based PFS with a tight reduction to Gap CDH in ROM in [1].

6 Security of TBPEKE, SPEKE, and their variants

In this section, we prove that the TBPEKE protocol proposed by Pointcheval and Wang [33] also realizes the lazy-extraction PAKE functionality under the same assumptions which were used to prove its game-based security. Moreover, since TBPEKE is a representative example of a class of protocols which includes SPEKE [22,31,20] and CPace [18,19], the same likely holds for these other protocols as well, or for their close variants. For example, it is straightforward to adapt our security proof for TBPEKE to show that lazy-extraction UC PAKE functionality is realized under the same assumptions also by SPEKE [22,31,20].[12] Due to its recent selection by the CFRG, we will analyze the specific case of CPace in the full version [2].[6]

We now recall the two-flow, simultaneous round TBPEKE protocol [33] in Fig. 7, with some notational choices adjusted to the UC setting.

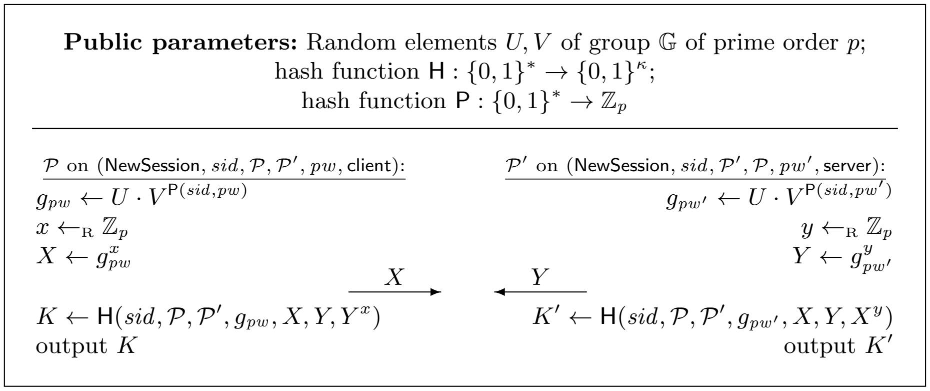
For the security proof of TBPEKE we require the following computational assumptions [33].

SDH and Gap SDH assumptions. The Simultaneous Diffie-Hellman (SDH) assumption states that, given three random elements X, $g = X^a$, and $h = X^b$ in a cyclic group of prime order, it is hard to find $Y \neq 1$ and R, S that simultaneously satisfy $R = \mathsf{CDH}_g(X, Y) = Y^{1/a}$ and $S = \mathsf{CDH}_h(X, Y) = Y^{1/b}$. In the gap version, the problem must remain hard even if the adversary has access to a Decisional Diffie-Hellman oracle.

Theorem 5. TBPEKE *realizes the Lazy-Extraction PAKE functionality* $\mathcal{F}_{\mathsf{lePAKE}}$ *in ROM under the Gap-CDH and Gap Simultaneous Diffie-Hellman (Gap-SDH) assumptions.*

For the formal proof of Theorem 5 we refer to the full version of this paper [2], and here we present only an informal overview of this proof and the simulator used in the proof, shown in Fig. 8.

[12] For instance, in the case of SPEKE (in which $g = G(sid, pw)$), one can adapt the proof for TBPEKE by simulating the random oracle G as $U \cdot V^{\mathsf{P}(sid,pw)}$.



Public parameters: Random elements U, V of group $\mathbb{G}$ of prime order p;
hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$;
hash function $\mathsf{P} : \{0,1\}^* \to \mathbb{Z}_p$

$\mathcal{P}$ on $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', pw, \mathsf{client})$:

$g_{pw} \leftarrow U \cdot V^{\mathsf{P}(sid,pw)}$
$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$
$X \leftarrow g_{pw}^x$

$\xrightarrow{X}$

$K \leftarrow \mathsf{H}(sid, \mathcal{P}, \mathcal{P}', g_{pw}, X, Y, Y^x)$
output K

$\mathcal{P}'$ on $(\mathsf{NewSession}, sid, \mathcal{P}', \mathcal{P}, pw', \mathsf{server})$:

$g_{pw'} \leftarrow U \cdot V^{\mathsf{P}(sid,pw')}$
$y \leftarrow_{\mathrm{R}} \mathbb{Z}_p$
$Y \leftarrow g_{pw'}^y$

$\xleftarrow{Y}$

$K' \leftarrow \mathsf{H}(sid, \mathcal{P}, \mathcal{P}', g_{pw'}, X, Y, X^y)$
output K'

Fig. 7. TBPEKE protocol of [33], which uses an additional random oracle P for deriving the generator g_{pw}.

The proof that TBPEKE is lazy-extraction UC PAKE secure is structurally similar to that of SPAKE2. In particular, the simulator adopts the same high-level strategy for dealing with passive and active attacks, while simulating the random oracle that is used for key derivation by taking advantage of knowing the CRS. Random oracle P is trivially simulated, excluding collisions to avoid ambiguity between passwords. The sequence of games that justifies the simulation also follows the same pattern.

The only significant difference between the proofs for TBPEKE and SPAKE2 lies in the step where one must exclude the possibility that the adversary places two random-oracle queries for two different passwords that are consistent with the same protocol trace, which would prevent the simulator from maintaining consistency. In TBPEKE, this bad event corresponds to the case in which the adversary queries the hash oracle H on inputs (g_{pw_1}, X, Y, W_1) and (g_{pw_2}, X, Y, W_2) such that (X, Y, W_i) is a valid DDH tuple with respect to the generator g_{pw_i} for $i = 1, 2$. In the SPAKE2 proof, the corresponding event can be reduced to Gap CDH. For TBPEKE, the reduction to Gap CDH does not work and the stronger Gap SDH assumption (given above) is needed.

The strategy for embedding the Gap SDH assumption in the reduction is the same as that adopted in [33]; the reduction guesses the two queries to oracle P that correspond to the passwords that are involved in the bad event. By programming the CRS and the random oracle output for these passwords, the reduction can solve the Gap SDH problem if the bad event occurs and the query guesses are correct.

Acknowledgments. Michel Abdalla was supported in part by the ERC Project aSCEND (H2020 639554) and by the French ANR ALAMBIC Project (ANR-16-CE39-0006). Work of Manuel Barbosa was supported in part by the grant SFRH/BSAB/143018/2018 awarded by FCT, Portugal, and by the ERC Project

```
generate CRS
$U \leftarrow g^u; V \leftarrow g^v$ for $(u, v) \leftarrow_{\mathrm{R}} \mathbb{Z}_p$
return $U, V$

on $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', \mathsf{role})$ from $\mathcal{P}$
if $\pi_{\mathcal{P}}^{sid} \neq \perp$: return $\perp$
$(X, Y) \leftarrow (\perp, \perp)$
if $\mathsf{role} = \mathsf{client}$ :
    $(\mathcal{C}, \mathcal{S}) \leftarrow (\mathcal{P}, \mathcal{P}')$
    $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$; $X \leftarrow g^z$; $Z \leftarrow X$
    if $\pi_{\mathcal{P}'}^{sid} \neq \perp$ and $\pi_{\mathcal{P}'}^{sid}.\mathsf{role} \neq \mathsf{client}$: $Y \leftarrow \pi_{\mathcal{P}'}^{sid}.Y$; $\pi_{\mathcal{P}'}^{sid}.X \leftarrow X$
else if $\mathsf{role} = \mathsf{server}$ :
    $(\mathcal{C}, \mathcal{S}) \leftarrow (\mathcal{P}', \mathcal{P})$
    $z \leftarrow_{\mathrm{R}} \mathbb{Z}_p$; $Y \leftarrow g^z$; $Z \leftarrow Y$
    if $\pi_{\mathcal{P}'}^{sid} \neq \perp$ and $\pi_{\mathcal{P}'}^{sid}.\mathsf{role} \neq \mathsf{server}$: $X \leftarrow \pi_{\mathcal{P}'}^{sid}.X$; $\pi_{\mathcal{P}'}^{sid}.Y \leftarrow Y$
$\pi_{\mathcal{P}}^{sid} \leftarrow (\mathsf{role}, z, \mathcal{C}, \mathcal{S}, X, Y, \perp, \perp, \perp, \perp, \mathtt{T})$
send $Z$ from $\mathcal{P}$ to $\mathcal{A}$

on $Z^*$ from $\mathcal{A}$ as msg to $(sid, \mathcal{P})$
if $\pi_{\mathcal{P}}^{sid} = \perp$ or $\pi_{\mathcal{P}}^{sid}.\mathsf{waiting} = \mathtt{F}$: return $\perp$
$(\mathsf{role}, \cdot, \mathcal{C}, \mathcal{S}, X, Y, \cdot, \cdot, \cdot, \mathsf{guesses}, \cdot) \leftarrow \pi_{\mathcal{P}}^{sid}$
$K \leftarrow 0^{\kappa}$
if $\mathsf{role} = \mathsf{client}$ :
    $\pi_{\mathcal{P}}^{sid}.Y^* \leftarrow Z^*$
    if $Z^* = Y$: jump to end
else if $\mathsf{role} = \mathsf{server}$ :
    $\pi_{\mathcal{P}}^{sid}.X^* \leftarrow Z^*$
    if $Z^* = X$: jump to end
if $\pi_{\mathcal{P}}^{sid}.\mathsf{guesses}[Z^*] = (pw, K^*)$:
    $\mathsf{reply} \leftarrow (\mathsf{TestPwd}, sid, \mathcal{P}, pw)$ to $\mathcal{F}$
    if $\mathsf{reply} =$ "correct": $K \leftarrow K^*$
else: send $(\mathsf{RegisterTest}, sid, \mathcal{P})$ to $\mathcal{F}$
end: $\pi_{\mathcal{P}}^{sid}.\mathsf{waiting} \leftarrow \mathtt{F}$
       send $(\mathsf{NewKey}, sid, \mathcal{P}, K)$ to $\mathcal{F}$

on $\mathsf{H}(sid, \mathcal{C}, \mathcal{S}, g_{pw}, X', Y', W)$ from $\mathcal{A}$:
if $\mathsf{T_H}[sid, \mathcal{C}, \mathcal{S}, g_{pw}, X', Y', W] = \perp$:
    $K \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$
    find $pw \in \mathsf{T_P}$ s.t. $\mathsf{T_P}[sid, pw] = g_{pw}$; if none found, jump to end
    $(\hat{x}, \hat{y}) \leftarrow (\perp, \perp)$
    if $\pi_{\mathcal{C}}^{sid} \neq \perp$: $\hat{x} \leftarrow \pi_{\mathcal{C}}^{sid}.\mathsf{exp}/(u + v \cdot pw)$
    if $\pi_{\mathcal{S}}^{sid} \neq \perp$: $\hat{y} \leftarrow \pi_{\mathcal{S}}^{sid}.\mathsf{exp}/(u + v \cdot pw)$
    if $\pi_{\mathcal{C}}^{sid}.X = X'$ and $\pi_{\mathcal{S}}^{sid}.Y = Y'$ and $W = g_{pw}^{\hat{x}\hat{y}}$: abort
    else if $\pi_{\mathcal{C}}^{sid}.X = X'$ and $W = (Y')^{\hat{x}}$ :
        if $Y' = \pi_{\mathcal{C}}^{sid}.Y^*$: $\mathcal{P} \leftarrow \mathcal{C}$; jump to late_test_pw
        else: $\pi_{\mathcal{C}}^{sid}.\mathsf{guesses}[Y'] \leftarrow (pw, K)$
    else if $\pi_{\mathcal{S}}^{sid}.Y = Y'$ and $W = (X')^{\hat{y}}$:
        if $X' = \pi_{\mathcal{S}}^{sid}.X^*$: $\mathcal{P} \leftarrow \mathcal{S}$; jump to late_test_pw
        else: $\pi_{\mathcal{S}}^{sid}.\mathsf{guesses}[X'] \leftarrow (pw, K)$
    jump to end
    late_test_pw: $\mathsf{reply} \leftarrow (\mathsf{LateTestPwd}, sid, \mathcal{P}, pw)$ to $\mathcal{F}$
                  $K \leftarrow \mathsf{reply}$; if no reply, abort
    end: $\mathsf{T_H}[sid, \mathcal{C}, \mathcal{S}, g_{pw}, X', Y', W] \leftarrow K$
send $\mathsf{T_H}[sid, \mathcal{C}, \mathcal{S}, g_{pw}, X', Y', W]$ to $\mathcal{A}$

on $\mathsf{P}(sid, pw)$ from $\mathcal{A}$:
if $\mathsf{T_P}[sid, pw] = \perp$:
    $\hat{p} \leftarrow_{\mathrm{R}} \mathbb{Z}_p$; if $\hat{p} \in \mathsf{T_P}.\mathsf{values}$: abort
    $\mathsf{T_P}[sid, pw] \leftarrow \hat{p}$
send $\mathsf{T_P}[sid, pw]$ to $\mathcal{A}$
```

Fig. 8. UC Simulator for TBPEKE.

aSCEND (H2020 639554). Stanisław Jarecki and Tatiana Bradley were supported by NSF SaTC award #1817143. Work of Jonathan Katz and Jiayu Xu was supported in part under financial assistance award 70NANB15H328 from the U.S. Department of Commerce, National Institute of Standards and Technology.

References

1. Abdalla, M., Barbosa, M.: Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194 (2019), https://eprint.iacr.org/2019/1194
2. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. IACR Cryptology ePrint Archive (2020), https://eprint.iacr.org/2020/320
3. Abdalla, M., Bellare, M., Rogaway, P.: DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a (Sep 1998)
4. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient two-party password-based key exchange protocols in the UC framework. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 335–351. Springer, Heidelberg (Apr 2008)
5. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (Jan 2005)
6. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. IEE Proceedings — Information Security 153(1), 27–39 (Mar 2006)
7. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (Feb 2005)
8. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000)
9. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: IEEE Symposium on Security and Privacy – S&P 1992. pp. 72–84. IEEE (1992)
10. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (May 2000)
11. Bradley, T., Camenisch, J., Jarecki, S., Lehmann, A., Neven, G., Xu, J.: Password-authenticated public-key encryption. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 19. LNCS, vol. 11464, pp. 442–462. Springer, Heidelberg (Jun 2019)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
13. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (May 2005)
14. Gennaro, R.: Faster and shorter password-authenticated key exchange. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 589–606. Springer, Heidelberg (Mar 2008)
15. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (Aug 2006)

16. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. Journal of Cryptology 19(3), 241–340 (Jul 2006)
17. Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010. pp. 516–525. ACM Press (Oct 2010)
18. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286 (2018), https://eprint.iacr.org/2018/286
19. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. IACR TCHES 2019(2), 1–48 (2019), https://tches.iacr.org/index.php/TCHES/article/view/7384
20. Hao, F., Shahandashti, S.F.: The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585 (2014), http://eprint.iacr.org/2014/585
21. Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Shin, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 485–504. Springer, Heidelberg (Sep 2018)
22. Jablon, D.P.: Extended password key exchange protocols immune to dictionary attacks. In: 6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997). pp. 248–255. IEEE Computer Society, Cambridge, MA, USA (Jun 18–20, 1997)
23. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Heidelberg (Apr / May 2018)
24. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163 (2018), https://eprint.iacr.org/2018/163
25. Jutla, C.S., Roy, A.: Dual-system simulation-soundness with applications to UC-PAKE and more. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 630–655. Springer, Heidelberg (Nov / Dec 2015)
26. Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer, Heidelberg (May 2001)
27. Katz, J., Ostrovsky, R., Yung, M.: Forward secrecy in password-only key exchange protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 02. LNCS, vol. 2576, pp. 29–44. Springer, Heidelberg (Sep 2003)
28. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Heidelberg (Mar 2011)
29. Krawczyk, H.: The OPAQUE asymmetric PAKE protocol, https://www.ietf.org/id/draft-krawczyk-cfrg-opaque-03.txt (Oct 2019)
30. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (Aug 2005)
31. MacKenzie, P.: On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057 (2001), http://eprint.iacr.org/2001/057
32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992)

33. Pointcheval, D., Wang, G.: VTBPEKE: Verifier-based two-basis password exponential key exchange. In: Karri, R., Sinanoglu, O., Sadeghi, A.R., Yi, X. (eds.) ASIACCS 17. pp. 301–312. ACM Press (Apr 2017)
34. Shoup, V.: Security analysis of spake2+. Cryptology ePrint Archive, Report 2020/313 (2020), https://eprint.iacr.org/2020/313