Forward Secrecy of SPAKE2

José Becerra(✉), Dimiter Ostrev, and Marjan Škrobot

Interdisciplinary Centre for Security, Reliability and Trust,
University of Luxembourg, 6, Avenue de la Fonte,
4364 Esch-sur-Alzette, Luxembourg
{jose.becerra,dimiter.ostrev,marjan.skrobot}@uni.lu

Abstract. Currently, the Simple Password-Based Encrypted Key Exchange (SPAKE2) protocol of Abdalla and Pointcheval (CT-RSA 2005) is being considered by the IETF for standardization and integration in TLS 1.3. Although it has been proven secure in the Find-then-Guess model of Bellare, Pointcheval and Rogaway (EUROCRYPT 2000), whether it satisfies some notion of *forward secrecy* remains an open question.

In this work, we prove that the SPAKE2 protocol satisfies the so-called *weak forward secrecy* introduced by Krawczyk (CRYPTO 2005). Furthermore, we demonstrate that the incorporation of key-confirmation codes in SPAKE2 results in a protocol that provably satisfies the stronger notion of *perfect forward secrecy*. As forward secrecy is an explicit requirement for cipher suites supported in the TLS handshake, we believe this work could fill the gap in the literature and facilitate the adoption of SPAKE2 in the recently approved TLS 1.3.

Keywords: Provable security
Password Authenticated Key Exchange · Forward secrecy
Common Reference String

1 Introduction

1.1 SPAKE2 Protocol

Password Authenticated Key Exchange (PAKE) protocols allow two users, who only share a *password*, to agree on a high-entropy *session key* over a hostile network. The goal is to use the established session key to build a *secure channel* between the involved parties. The nature of passwords makes PAKEs vulnerable to *on-line* dictionary attacks, where an adversary tries to impersonate a user by guessing his password, engaging in a protocol execution and verifying if its guess was correct. An offline dictionary attack occurs when the protocol execution allows an adversary to launch an exhaustive offline search of the password. The intuition of security requires PAKEs to be vulnerable to online dictionary attacks only. The seminal work in this area is the Encrypted Key Exchange (EKE) protocol of Bellovin and Merritt [1]. Since then, various PAKE protocols

J. Baek et al. (Eds.): ProvSec 2018, LNCS 11192, pp. 366–384, 2018.
https://doi.org/10.1007/978-3-030-01446-9_21

have been proposed: PPK and PAK [2,3], J-PAKE [4,5], SRP [6], SPEKE [7] and SPAKE2 [8]. In parallel, prominent complexity-theoric security models for PAKEs have been proposed to get assurance on the claimed security properties by performing a rigorous analysis of the protocol in question [2,9–12].

The SPAKE2 protocol, proposed by Abdalla and Pointcheval [8], is a one-round PAKE protocol proven secure in the Find-then-Guess (FtG) model of Bellare et al. [9] without considering *forward secrecy*. It is a simple, yet efficient protocol that, in addition to the pre-shared password, requires the protocol participants to share two Common Reference Strings (CRS) prior to the execution of the protocol. The adoption of the CRS yields to an elegant construction that does not require full domain hash functions, which are hard to implement efficiently in practice. On the other side, the CRS requires extra security assumptions that might be easy to satisfy in some scenarios but may be very restrictive in others [13]. Also, as it is a one-round protocol, only *implicit authentication* can be satisfied. Fortunately, the incorporation of *key-confirmation codes* allows the protocol participants to explicitly authenticate each other [14] and [15, Chap. 40].

Recently, the Internet Engineering Task Force (IEFT) community has revisited the deployment of SPAKE2 protocol: (i) as stand alone specification [16], (ii) its usage as pre-authentication mechanism in Kerberos protocol [17] and (iii) its adoption in TLS 1.3 protocol, specifically in the *handshake* when *pre-shared* keys for authentication are available [18,19]. The discussion of forward secrecy in SPAKE2 has been a common factor in the aforementioned Internet Drafts.

1.2 PAKEs Adoption in TLS

Nowadays, the Transport Layer Security (TLS) is the *de-facto* standard to protect internet communications. It consists of two stages: the *Handshake* protocol where two parties agree on a session key, and the *Record* protocol where the communication is protected using the previously negotiated keys. Most of the TLS implementations provide only *unilateral* authentication, where client C authenticates server S during the handshake by means of public-key infrastructure (PKI), therefore identity disclosure of client to server is usually not supported.

While the unilateral server-authenticated approach might be sufficient for scenarios like internet surfing, it is certainly inadequate for real-world applications including email access, internet banking and social media, where client C needs to authenticate to server S to gain access to resources in S. In practice, the common approach for authenticating the client asks the client to send his user/password protected through a server-authenticated TLS channel. This approach protects the password against eavesdroppers but not against phishing attacks: An adversary can clone a legitimate website and fool the client to visit the fake website where he input his credentials. To make things worse, the adversary can manage to obtain a valid public-key certificate from a certification authority (CA) for his illegitimate web page. Indeed, the client may see on

his web browser "secure connection" as a TLS connection may be established between the client and the cloned website controlled by the adversary.[1]

Fortunately, PAKEs stand as a strong candidate for scenarios where two parties require to *mutually* authenticate each other while intrinsically protecting their shared password. In fact, the Secure Remote Password (SRP) protocol [6] has been incorporated in previous versions of TLS and standardized in the form of RFC5054 [20]. Specifically, the SRP protocol was made available as cipher suite in the TLS handshake. Similarly, the IETF is currently considering the adoption of SPAKE2 in TLS 1.3 *handshake* [18], in particular in the TLS handshake, for scenarios where authentication is made using pre-shared password available between the Client and Server.

In the recently approved TLS 1.3, it has explicitly been a design goal to provide *forward secrecy* for the session keys used to construct the TLS channel. In particular, *static* RSA and Diffie-Hellman cipher suites were removed to favor public-key based key-exchange mechanism that guarantee forward secrecy. Therefore, formally proving that SPAKE2 satisfies some significant notion of forward secrecy would increase its possibilities of acceptance into TLS 1.3.

Remark: While PAKEs adoption in web authentication is a good approach to protect user's password during the authentication phase, there are still *usability* concerns that slow down the implementation of PAKEs in TLS to properly prevent phishing attacks. This implementation requires an easy to identify "safe area" available in the web browser where the passwords should be entered [21].

1.3 Forward Secrecy

Forward secrecy is a desirable property which has been explicitly a design goal in relevant AKE and PAKE protocols [3,4,22,23], and more recently in TLS 1.3 [19].[2] Roughly speaking, it ensures the protection of session keys even if the long-term secret of the participants gets later compromised [24]. For instance: (i) the password file at the server could be leaked or (ii) via phishing attacks a client could reveal his password to some malicious entity.

The notion of forward secrecy appeared first in [24] and was later formalized in [23,25–27] for AKE and in [9,28] for PAKE protocols. We distinguish *weak forward secrecy* (wFS) from perfect forward secrecy (PFS): The former protects session keys after compromise of long-term key material, but only those sessions created *without the active participation of the attacker* [23], while the latter protects all session keys which were negotiated before corruption, i.e. even those created with the active intervention of the adversary. It is generally accepted that PFS is difficult to satisfy in protocols which only guarantee *implicit authentication*. For instance, Krawczyk [23] states that PFS cannot be satisfied by two-flow protocols using *public-key* as authentication mechanism. Therefore Krawczyk proposed the notion of *weak Forward Secrecy* (wFS) as an attempt to satisfy

[1] A typical client should not be expected to verify the certificate details.

[2] However, in TLS 1.3, there still remains some configurations that do not satisfy forward secrecy.

some notion of security when long-term material is compromised but only for those sessions without the active participation of the adversary.

PFS and Key-Confirmation: The authors in [3,9,25] demonstrated that PFS can be satisfied when *explicit authentication* is added to protocols that initially satisfy only wFS. The idea is the following: Suppose P is a 2-flow PAKE protocol satisfying only implicit authentication. The adversary sends the first message to Bob masquerading as Alice, Bob computes the session key, sends back the second message and finishes his protocol execution. Then the adversary waits for the leakage of the long-term key and that could *possibly* help her to compute the same session key as Bob. For this scenario, the notion of PFS requires the adversary not to learn Bob's session key, which can be easily avoided by requiring key-confirmation, since then Bob will not accept the session key before he authenticates his communication partner.

1.4 Our Contribution

We propose a new version of SPAKE2 which we name PFS-SPAKE2. This is essentially SPAKE2 but with key-confirmation codes incorporated into the protocol. This well known approach allowed us to meet the PFS requirement in a provably secure way even in the case of active adversaries, making it a suitable candidate for standardization and adoption in the TLS 1.3 protocol. In addition, we prove that the original SPAKE2 satisfies weak forward secrecy.

2 Security Model with Forward Secrecy

Notation. We use calligraphic letters to denote adversaries, typically $\mathcal{A}$ and $\mathcal{B}$. We write $s \xleftarrow{\$} S$ for sampling uniformly at random from set S and $|S|$ to denote its cardinality. The output of a probabilistic algorithm A on input x is denoted by $y \leftarrow A(x)$, while $y := F(x)$ denotes a deterministic assignment of $F(x)$ to the variable y. Let $\{0,1\}^*$ denote the bit string of arbitrary length while $\{0,1\}^l$ stands for those of length l. Let λ be the security parameter, $negl(\lambda)$ denote a negligible function and PPT stand for probabilistic polynomial time.

Next we describe the well-known security model of Bellare, Pointcheval and Rogaway [9], which we use to prove the security of PFS-SPAKE2 and SPAKE2 protocol. Frequently referred as the Find-then-Guess (FtG) model, it is an extension of [29,30] to the password setting. We assume the reader is familiar with the model.

PAKE PROTOCOL. A PAKE protocol is defined by a pair of algorithms $(Gen, \mathcal{P})$. Gen is the password generation algorithm. It takes as input the dictionary D, a probability distribution $\mathcal{Q}$ and initializes the protocol participants with some password. The protocol description $\mathcal{P}$ defines how honest participants behave.

PROTOCOL PARTICIPANTS. Each participant is either a client $C \in \mathcal{C}$ or a server $S \in \mathcal{S}$. Let $\mathcal{U} = \mathcal{C} \cup \mathcal{S}$ denote the set of all (honest) users and $\mathcal{C} \cap \mathcal{S} = \varnothing$.

LONG-TERM SECRETS. Each client C holds a password π_C and server S holds a vector of passwords for all clients i.e. $\pi_S =< \pi_C >_{C \in \mathcal{C}}$ s.t. for each client C $\pi_S[C] = \pi_C$. We consider the client-server scenario where there is a single server S. The passwords are assumed to be independent and uniformly distributed.
PROTOCOL EXECUTION. $\mathcal{P}$ is a probabilistic algorithm that defines how users respond to signals from the environment. We assume the presence of a PPT adversary $\mathcal{A}$ with full control of the network and an unlimited number of *user instances*. Specifically, let Π_U^i denote the instance i-th of user $U \in \mathcal{U}$. In cases where distinction matters, let Π_C^i and Π_S^j denote the i-th and j-th instance of client $C \in \mathcal{C}$ and server S respectively.

Security is defined via a game played between the challenger $\mathcal{CH}$ and adversary $\mathcal{A}$ whose goal is to break the semantic security of the established session keys. $\mathcal{A}$ controls the oracle user instances with the following queries:

- **Send**(U, i, m): A message m is sent to instance Π_U^i and processed according to the protocol description $\mathcal{P}$. Its output is given to $\mathcal{A}$.
- **Execute**(C, i, S, j): This query causes an honest run of protocol $\mathcal{P}$ between Π_C^i and Π_S^j, the transcript of execution is given to $\mathcal{A}$.
- **Reveal**(U, i): The session key sk_U^i held at Π_U^i is given to $\mathcal{A}$. It requires the sk_U^i to be already computed, i.e. Π_U^i must be on *terminate* state.
- **Corrupt**(U). The adversary obtains the password of user U. If $U = C \in \mathcal{C}$, then $\mathcal{A}$ receives π_C, else if $U = S$, then $\mathcal{A}$ receives $\pi_S =< \pi_C >_{C \in \mathcal{C}}$.
- **Test**(U, i): $\mathcal{CH}$ flips a bit b and answers the query as follows: if $b = 1$ $\mathcal{A}$ gets the session key sk_U^i, otherwise she receives $r \xleftarrow{\$} \{0,1\}^\kappa$, where $\{0,1\}^\kappa$ denotes the length of the session key space.

2.1 Definitions

Partnering. Two instances, Π_C^i and Π_S^j, are partnered if both *accept*, holding $(sk_C^i, sid_C^i, pid_C^i)$ and $(sk_S^j, sid_S^j, pid_S^j)$ respectively and also:

1. $sk_C^i = sk_S^j$, $sid_C^i = sid_S^j$, $pid_C^i = S$, $pid_S^j = C$ and
2. no other instance accepts with the same session identifier *sid*, except with negligible probability.

The notion of freshness avoids scenarios where an adversary could trivially win the security experiment. Next we define two notions of freshness depending on the desired of forward secrecy guarantee: The first flavour models PFS, where the intuition is to consider as legitimate targets of a Test query those instances which session keys were negotiated *before* the corruption of any principal. The second variant models wFS, which does not guarantee the secrecy of those sessions keys which were negotiated with the *active* intervention of an adversary (determined via *partnering*) whenever some user has been corrupted.

PFS-Freshness. An instance Π_U^i is *PFS-fresh* unless:

- A Reveal query was made to Π_U^i or its partner or

– There was a Corrupt(U') and a Send(U, i, m) query, Π_U^i does not have a partner and the corruption of any user U' occurs **before** the Test query.

wFS-Freshness. An instance Π_U^i is *wFS-fresh* unless:

– A Reveal query was made to Π_U^i or its partner or
– There was a Corrupt(U') and a Send(U, i, m) query, Π_U^i does not have a partner and the corruption of any user U' occurs at any time.

Advantage of the Adversary. Let $\text{Succ}_P^{\text{PFS-FtG}}$ be the event where $\mathcal{A}$ asks a single Test query directed to a *PFS-fresh* instance that has terminated, $\mathcal{A}$ outputs his guess b' and wins i.e. $b' = b$. The advantage of $\mathcal{A}$ attacking protocol P is:

$$\text{Adv}_P^{\text{PFS-FtG}}(\mathcal{A}) = 2 \cdot \Pr\left[\text{Succ}_P^{\text{FtG}}(\mathcal{A})\right] - 1 \tag{1}$$

Definition 1 *(PFS-FtG security). Protocol P is FtG secure and satisfies perfect forward secrecy if for all PPT adversaries there exists a negligible function $\epsilon(\cdot)$ such that:*

$$\text{Adv}_P^{\text{PFS-FtG}}(\mathcal{A}) \leq n_{se}/|D| + \epsilon(\lambda),$$

where n_{se} is the number of Send queries and D is the password dictionary.

We similarly define FtG security with *weak* forward secrecy, the only change is in the advantage function, where the Test query must be made to a wFS-fresh instance. From inspection, it is easy to see that PFS-FtG → wFS-FtG security.

2.2 Cryptographic Hardness Assumptions

Let $\mathbb{G}$ be a multiplicative a group, with generator g and $|\mathbb{G}| = q$. For $X = g^x$ and $Y = g^y$, let $\text{DH}(X, Y) = g^{xy}$, where $\{g^x, g^y, g^{xy}\} \in \mathbb{G}$.

Definition 2 *(Computational Diffie-Hellman (CDH) Problem). Given (g, g^x, g^y) compute g^{xy}, where $\{g^x, g^y, g^{xy}\} \in \mathbb{G}$ and $(x, y) \xleftarrow{\$} \mathbb{Z}_q^2$. Let the advantage of an algorithm $\mathcal{A}$ in solving the CDH problem be:*

$$\text{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\mathcal{B}) = \Pr\left[(x, y) \xleftarrow{\$} \mathbb{Z}_q^2, X = g^x, Y = g^y : \mathcal{B}(X, Y) = DH(X, Y)\right].$$

Under the *CDH assumption* there exist sequences of cyclic groups $\mathbb{G}$ indexed by λ s.t. $\forall \mathcal{B}$ running in time t polynomial in λ, $\text{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\mathcal{B})$ is a negligible function.

3 PFS-SPAKE2

Inspired by MacKenzie's work [3], we propose to incorporate key-confirmation codes into the SPAKE2 protocol [8] to achieve PFS in a provably secure manner.

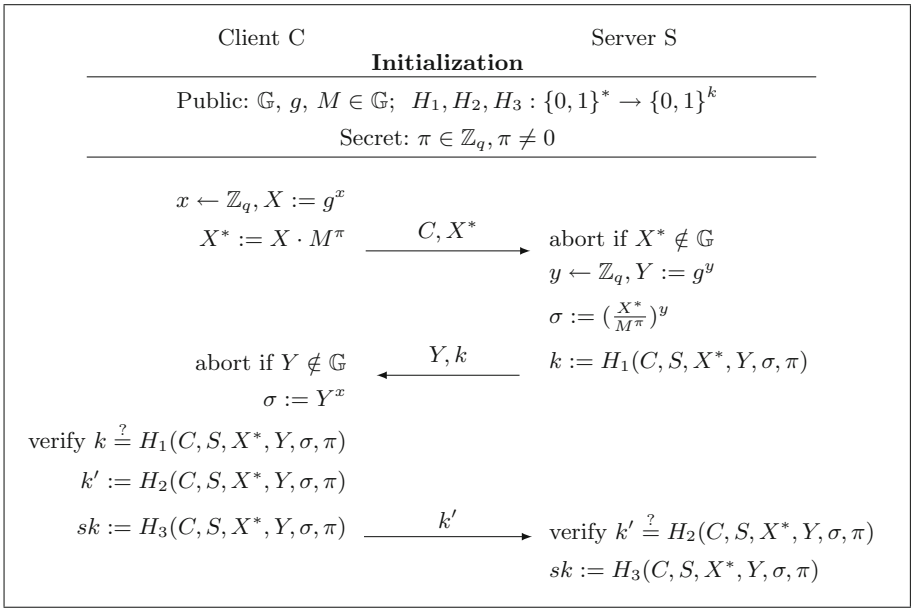

Fig. 1. PFS-SPAKE2 protocol.

3.1 Protocol Description

In Fig. 1 we provide the technical description of the proposed PFS-SPAKE2 protocol. Before the protocol is executed, public parameters must be chosen and published. These parameters include the description of group $\mathbb{G}$, hash functions H_1, H_2, H_3 and a CRS M – which we require to be choosen at random from $\mathbb{G}$ and its discrete logarithm to be kept secret. These constraints on the CRS can be achieved either by having a third trusted party or by assuming a public source of randomness to publicly derive M. Our protocol is instantiated over group $\mathbb{G}$, a q order subgroup of $\mathbb{Z}_p^*$ where CDH assumption holds and p, q are safe prime numbers. The protocol requires that passwords are encoded in $\mathbb{Z}_q$.

Comparison to Existing PAKEs. The efficiency of a PAKE protocol is defined by (i) the number of communication rounds until the protocol terminates, (ii) the total number messages exchanged and (iii) the computational cost of the protocol. Compared to the original SPAKE2, the proposed PFS-SPAKE2 protocol benefits from *explicit* authentication and strong security guarantees for PFS. It is also slightly less computationally expensive, as it requires the client to compute only three exponentiations instead of four, i.e. no need to compute $N^\pi \in \mathbb{G}$. These improvements usually come at the cost of increasing the number of rounds and message flows and unfortunately our protocol is not an exception [3,23].

Table 1. Comparison with existing PAKEs for Client-Server scenarios.

Protocol[a]	Commn.[b]	Computation[c]	Rounds/ Flows	Hardness Asm.[d]	Forward Secrecy	Key Confirm.
EKE [1,9]	$2 \times \mathbb{G}$	4 exp., 2 enc.	1/2	CDH	wFS	No
SPEKE [7,32]	$2 \times \mathbb{G} + 2\kappa$	4 exp.	2/4	DIDH	-	Yes
PPK [3]	$2 \times \mathbb{G}$	6 exp.	1/2	CDH	-	No
PAK [3]	$2 \times \mathbb{G} + 2\kappa$	5 exp.	3/3	CDH	PFS	Yes
J-PAKE[d] [4]	$12 \times \mathbb{G} + 6 \times \mathbb{Z}_q$	28 exp.	2/4	DSDH	PFS	No
J-PAKE* [4]	$12 \times \mathbb{G} + 6 \times \mathbb{Z}_q$	28 exp.	3/6	DSDH	PFS	Yes
SPAKE2 [8]	$2 \times \mathbb{G}$	6 exp.	1/2	CDH	wFS	No
PFS-SPAKE2	$2 \times \mathbb{G} + 2\kappa$	5 exp.	3/3	CDH	PFS	Yes

[a] J-PAKE* is simply J-PAKE but with an extra round for key-confirmation.
[b] Communication. $\mathbb{G}$ denotes a group element, $\mathbb{Z}_p$ a scalar and κ a κ-bit string.
[c] Exp. denotes an exponentiation in $\mathbb{G}$ and enc. an encryption and decryption operation.
[d] DSDH and DIDH stand for Decision Square and Decision Inverted-Additive Diffie-Hellman.

In Table 1 we summarize the comparison of PFS-SPAKE2 with other relevant PAKE protocols with full security proofs.[3] Notably J-PAKE satisfies PFS and requires only two communication rounds; however, it is computationally more expensive than PFS-SPAKE2 as the former requires 28 exponentiations while the latter only 5. Furthermore, J-PAKE with key-confirmation requires the same number of communication rounds as PFS-SPAKE2. Alternatively, PAK and PFS-SPAKE2 are similar in terms of efficiency, PFS and key confirmation guarantees, yet the usage of CRS in the latter allowed us to achieve *tighter* security reductions to the CDH assumption than the original results for PAK [3,31].

3.2 Security of PFS-SPAKE2

Theorem 1 *(Security in the PFS-FtG Model). Let P be the protocol specified in Fig. 1, instantiated in group $\mathbb{G}$ and with passwords uniformly distributed over dictionary D. Let $\mathcal{A}$ be an adversary that runs in time t polynomial in λ, makes at most n_{ex}, n_{se}, n_{ro} queries of type execute, send and random oracle. Then:*

$$\mathrm{Adv}_P^{\text{PFS-FtG}}(\mathcal{A}) \leq \frac{n_{se}}{|D|} + \mathcal{O}\left(\frac{(n_{se}+n_{ex})(n_{se}+n_{ex}+n_{ro})}{q} + n_{ro} \cdot \mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{B}^{\mathcal{A}}) + n_{se}n_{ro} \cdot \mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\tilde{\mathcal{B}}^{\mathcal{A}}) + n_{ro}^2 \cdot \mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\hat{\mathcal{B}}^{\mathcal{A}})\right),$$

where $\mathcal{B}^{\mathcal{A}}$, $\tilde{\mathcal{B}}^{\mathcal{A}}$ and $\hat{\mathcal{B}}^{\mathcal{A}}$ are CDH-solver algorithms running in time $t' = \mathcal{O}(t + (n_{se} + n_{ex} + n_{ro}) \cdot t_{exp})$, where t_{exp} is the time for an exponentiation in $\mathbb{G}$.

[3] The server usually stores some function $f(\cdot)$ of the password while the clients needs to compute $f(\pi)$ for every protocol run. This difference is relevant in (i) PPK, PAK and (ii) SPAKE2 and PFS-SPAKE2, as $f(\cdot)$ requires hashing into groups in (i) and group exponentiations in (ii).

To prove the security of PFS-SPAKE2, we introduce a sequence of protocols $\mathrm{P}_0 \ldots \mathrm{P}_7$, where P_0 is the original protocol and P_7 allows only online dictionary attacks. Let $\mathrm{G_i}$ be the security game associated to $\mathrm{P_i}$. We borrow from [3] the structure and the nomenclature to prove the security of our PFS-SPAKE2 protocol and refer to Appendix A for the necessary terminology.

The security proof requires the random oracle model: each *new* random oracle query H_l for $l \in \{1, 2, 3\}$ is answered with a fresh random output, however, if the query has been previously made, it is answered consistenly with previous queries. In cases where it is clear enough, we write $H_l(\cdot)$ to refer to query of the form $H_l(C, S, X^*, Y, \sigma, \pi)$. For easiness of the proof we assume that for each $H_l(C, S, X^*, Y, \sigma, \pi)$ query made by $\mathcal{A}$, with $l \in \{1, 2, 3\}$, the corresponding $H_{l'}(\cdot)$ and $H_{l''}(\cdot)$ are also made, with $l', l'' \in \{1, 2, 3\} \backslash \{l\}$ and $l' \neq l''$. The simulator sets $M := g^m \in \mathbb{G}$, where $m \xleftarrow{\$} \mathbb{Z}_q$.

In the following games, we simply write $\mathrm{Succ}^{\mathrm{FtG}}_{P_i}$ instead of $\mathrm{Succ}^{\mathrm{PFS\text{-}FtG}}_{P_i}$ to denote the success probability of $\mathcal{A}$ winning in game $\mathrm{G_i}$.

Game $\mathbf{G_0}$: Execution of original protocol.
Game $\mathbf{G_1}$: Uniqueness of honest sessions.

During the interaction with adversary $\mathcal{A}$, the challenger needs to simulate honest instances and generate the X^* and Y terms according to the protocol description. Let F_1 be the event where there is a collision between either an X^* or Y value, with previously seen X^* or Y values. If F_1 occurs, the challenger draws random values again until he arrives at a X^* or Y term that has not been previously seen. It is easy to show that the probability of F_1 occurring is bounded by the birthday paradox. Then for all $\mathcal{A}$:

$$\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_0}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_1}(\mathcal{A})\right] + \mathcal{O}\left(\frac{(n_{se} + n_{ex})(n_{se} + n_{ex} + n_{ro})}{q}\right).$$

Game $\mathbf{G_2}$: Prevent Lucky Guesses on Hash Outputs.
This game forces $\mathcal{A}$ to query the random oracle whenever she needs to compute any hash $H(\cdot)_l$. As a result, this game rules out the possibility of $\mathcal{A}$ to output correct values k, k' or sk without calling the corresponding random oracle.

Let P_2 be a protocol identical to P_1, except that honest instances respond to Send and Execute queries without making any random oracle queries and subsequent random oracle queries made by $\mathcal{A}$ are backpatched to be consistent with previous queries. Next we detail the changes in P_2.

- In an Execute(C, i, S, j) query set $X^* = g^{\tau[C,i]}$ and $Y = g^{\tau[S,j]}$, where $\tau[\cdot] \xleftarrow{\$} \mathbb{Z}_q$, $k, k' \xleftarrow{\$} \{0,1\}^\kappa$ and $sk^j_S \leftarrow sk^i_C \xleftarrow{\$} \{0,1\}^\kappa$, where $\{0,1\}^\kappa$ denotes the session key space.
- In a CLIENT ACTION 0 query to Π^i_C, set $X^* = g^{\tau[C,i]}$, where $\tau[C, i] \xleftarrow{\$} \mathbb{Z}_q$.
- In a SERVER ACTION 1 query to Π^j_S, set $Y = g^{\tau[S,j]}$ and $k \xleftarrow{\$} \{0,1\}^\kappa$, where $\tau[S, j] \xleftarrow{\$} \mathbb{Z}_q$.

– In a CLIENT ACTION 1 query to Π_C^i proceed as follows:
  - If Π_C^i is paired with Π_S^j then set $k', sk_C^i \xleftarrow{\$} \{0,1\}^\kappa$.
  - Else if this query triggers a $\mathbf{testpw}(C, i, S, \pi_c, l)$ event, for some $l \in \{1,2,3\}$, then set k' and sk_C^i to the associated value of the event $\mathbf{testpw}(C, i, S, \pi_c, 2)$ and $\mathbf{testpw}(C, i, S, \pi_c, 3)$ respectively.
  - Else Π_C^i aborts.

– In a SERVER ACTION 2 query to Π_S^j proceed as follows:
  - If Π_S^j is *paired with* Π_C^i *after* some CLIENT ACTION 1 query to Π_C^i, then set $sk_S^j \leftarrow sk_C^i$.
  - Else this query triggers a $\mathbf{testpw}(S, j, C, \pi_c, l)$, with $l \in \{1,2,3\}$, set sk_S^j to the associated value of the event $\mathbf{testpw}(S, j, C, \pi_c, 3)$.
  - Else instance Π_S^j aborts.

– In an $H_l(C, S, X^*, Y, \sigma, \pi)$ query made by $\mathcal{A}$, if it triggers a $\mathbf{testpw}(C, i, S, \pi_C, l)$, $\mathbf{testpw}(S, j, C, \pi_C, l)$ or $\mathbf{testexecpw}(C, i, S, j, \pi_C)$ event, then output the associated event of the corresponding event. Otherwise output $v \xleftarrow{\$} \{0,1\}^\kappa$.

Claim 1. *For all adversaries* $\mathcal{A}$, $\Pr\left[\mathrm{Succ}_{P_1}^{\mathrm{FtG}}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}_{P_2}^{\mathrm{FtG}}(\mathcal{A})\right] + \frac{n_{se}}{2^\kappa}$.

Proof. In SERVER ACTION 2 to Π_S^j, the input k' determines whether the instance Π_S^j should *terminate* or *abort*. Let F_1 be the event where in a SERVER ACTION 2 to Π_S^j, it terminates such that (i) Π_S^j is not paired with Π_C^i and (ii) $\mathbf{testpw}(S, j, C, \pi_C, l)$ event does not occur, for $l \in \{1,2,3\}$, i.e. $\mathcal{A}$ luckily guessed the correct k' value. Then $\Pr[F_1] \leq n_{se}/2^\kappa$. □

Game $\mathbf{G_3}$: Do not backpatch $H_l(\cdot)$ queries against Execute queries. This game shows that there is no need to backpatch $H_l(\cdot)$ queries to maintain consistent views against Execute queries. More formally, let P_3 be identical to P_2 except that, in a $H_l(C, S, X^*, Y, \sigma, \pi_C)$ query made by $\mathcal{A}$, the simulator does not verify whether the $\mathbf{testexec}(C, i, S, j, \pi_C)$ event occurs or not. Let F_2 and F_3 denote the $\mathbf{testexec}(C, i, S, j, \pi_C)$ event occurring in P_2 and P_3 respectively.

Claim 2. *For all adversaries* $\mathcal{A}$, $\left|\Pr\left[\mathrm{Succ}_{P_2}^{\mathrm{FtG}}(\mathcal{A})\right] - \Pr\left[\mathrm{Succ}_{P_3}^{\mathrm{FtG}}(\mathcal{A})\right]\right| \leq \Pr[F_2]$.

Proof. P_2 and P_3 are identical protocols until the $\mathbf{testexec}(C, i, S, j, \pi_C)$ event occurs. The observation is that the events F_2 and F_3 are triggered as result of some interaction $\mathcal{CH}_2$ vs $\mathcal{A}$ and $\mathcal{CH}_3$ vs $\mathcal{A}$ respectively, however by definition they are identical. Then it follows that $\Pr[F_2] = \Pr[F_3]$ and to conclude the proof we simply apply Shoup's Difference Lemma [33]. □

Claim 3. *Given* $\mathcal{A}$, *there exists a CDH-solver* $\mathcal{B}^{\mathcal{A}}$ *with running time* $t' = \mathcal{O}(t + (n_{se} + n_{ex} + n_{ro}) \cdot t_{exp})$ *such that:*

$$\Pr\left[\mathrm{Succ}_{P_2}^{\mathrm{FtG}}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}_{P_3}^{\mathrm{FtG}}(\mathcal{A})\right] + n_{ro} \cdot \mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{B}^{\mathcal{A}}),$$

Proof. Let ϵ be the probability that **testexec**(C, i, S, j, π) event occurs in P_2. We build an adversary $\mathcal{B}^{\mathcal{A}}$ whose goal is to solve the CDH problem using adversary $\mathcal{A}$ as a subroutine and with success probability ϵ/n_{ro}. On input $(A = g^{\alpha}, B = g^{\beta})$, $\mathcal{B}^{\mathcal{A}}$ *simulates* P_2 to $\mathcal{A}$ with the following changes:

1. For every Execute(C, i, S, j) query made by $\mathcal{A}$, the simulator $\mathcal{B}^{\mathcal{A}}$ sets $X^* = A \cdot g^{r_1}$, $Y = B \cdot g^{r_2}$, $k, k' \xleftarrow{\$} \{0,1\}^{\kappa}$ and $sk_S^j \leftarrow sk_C^i \xleftarrow{\$} \{0,1\}^{\kappa}$, where $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$ are known to the simulator.
2. For every $H_l(C, S, X^*, Y, \sigma, \pi_C)$ query, where $l \in \{1, 2, 3\}$, X^* and Y are generated via an Execute(C, i, S, j) query, add γ to the set S-DH, where:

$$\gamma = \sigma \cdot B^{m \cdot \pi_C} \cdot M^{r_2 \cdot \pi_C} / B^{r_1} \cdot A^{r_2} \cdot g^{r_1 r_2}$$

3. When $\mathcal{A}$ finishes, the set S-DH contains at most n_{ro} elements, where each item a possible solution to DH(g^{α}, g^{β}). Then $\mathcal{B}^{\mathcal{A}}$ outputs $\gamma \xleftarrow{\$}$ L-DH.

The adversary $\mathcal{A}$ can only distinguish P_2 from P_3 once **testexec**(C, i, S, j, π) has occurred, but this happens with probability $\epsilon \leq n_{ro} \cdot \mathrm{Adv}_{G_q}^{\mathrm{CDH}}(t')$. We make the observation that G_3 guarantees forward secrecy for session keys established via Execute queries. □

Game $\mathbf{G_4}$: Check for successful password guesses.
Let P_4 be identical to P_3, except that if **correctpw** event occurs, the protocol stops and the adversary automatically wins.

Claim 4. *For all PPT adversaries* $\mathcal{A}$, $\Pr\left[\mathrm{Succ}_{P_3}^{\mathrm{FtG}}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}_{P_4}^{\mathrm{FtG}}(\mathcal{A})\right]$.

Proof. Obvious. □

This game simply counts for an adversary who is successful in an online dictionary attack by impersonating either a Client or the Server. The implication is that from P_4, until either **correctpw** event or a Corrupt query occurs, no *unpaired* client or server instance will terminate.

Game $\mathbf{G_5}$: Randomized session keys for paired instances.
Let P_5 be identical to P_4 except that if the **pairedpwguess** event occurs the protocol stops and the adversary fails.

In this game we will demonstrate that an adversary $\mathcal{A}$ who (i) may actively *corrupt* any Client or Server, i.e. $\mathcal{A}$ knows the corresponding *correct* password π_C and (ii) manages to compute k, k' or sk for *paired* instances Π_C^i and Π_S^j, is also a CDH-solver. Let F_4 and F_5 denote the **pairedpwguess** event occurring in P_4 and P_5 respectively.

Claim 5. *For all adversaries* $\mathcal{A}$, $\left|\Pr\left[\mathrm{Succ}_{P_4}^{\mathrm{FtG}}(\mathcal{A})\right] - \Pr\left[\mathrm{Succ}_{P_5}^{\mathrm{FtG}}(\mathcal{A})\right]\right| \leq \Pr\left[F_4\right]$.

Proof. Identical to Claim 2. □

Claim 6. *Given $\mathcal{A}$, there exists CDH-solver $\tilde{\mathcal{B}}^{\mathcal{A}}$ with running time $t' = \mathcal{O}(t + (n_{se} + n_{ex} + n_{ro}) \cdot t_{exp})$ such that:*

$$\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_4}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_5}(\mathcal{A})\right] + n_{se} \cdot n_{ro} \cdot \mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\tilde{\mathcal{B}}^{\mathcal{A}}),$$

Proof. Let ϵ be the probability of **pairedpwguess** event happening. We build $\mathcal{B}^{\mathcal{A}}$, a CDH-solver with success probability $\epsilon/(n_{se} \cdot n_{ro})$. On input $(A = g^{\alpha}, B = g^{\beta})$, $\tilde{\mathcal{B}}^{\mathcal{A}}$ sets $M = g^{m} \in \mathbb{G}$ for $m \xleftarrow{\$} \mathbb{Z}_q$, chooses $d \in \{1...n_{se}\}$ at random – a session target of the Test query – and *simulates* P_4 to $\mathcal{A}$ with the following changes:

1. In a CLIENT ACTION 0 query to Π^d_C with input S, set $X^* \leftarrow A$, where Π^d_C is the client instance that $\tilde{\mathcal{B}}^{\mathcal{A}}$ hopes it remains *PFS-fresh*.
2. In a SERVER ACTION 1 query to Π^j_S with input $\langle C, m\rangle$, where there was previous a CLIENT ACTION 0 query to Π^d_C with input S and output $\langle C, m\rangle$, set $Y = B \cdot g^{r_{S,j}}$, where $r_{S,j} \xleftarrow{\$} \mathbb{Z}_q$.
3. In a CLIENT ACTION 1 query to Π^d_C, if Π^d_C is *unpaired* then it aborts an also $\tilde{\mathcal{B}}^{\mathcal{A}}$ stops the simulation.
4. In a SERVER ACTION 2 query to Π^j_S, if it was *paired* with Π^d_C after its SERVER ACTION 1 but now is *not paired*, then Π^j_S aborts. However, the simulation continues as the instance Π^d_C may still be target of the Test query.
5. When $\mathcal{A}$ finishes, then for every $H_l(C, S, X^*, Y, \sigma, \pi_C)$, made by $\mathcal{A}$, with $l \in \{1, 2, 3\}$ and where (i) X^* and Y were generated by Π^d_C and Π^j_S respectively, (ii) Π^j_S was paired with Π^d_C after its SERVER ACTION 1 and (iii) Π^d_C was paired with Π^j_S, then add γ to the set S-DH, where:

$$\gamma = \sigma \cdot B^{m \cdot \pi_C} \cdot M^{r_{S,j} \cdot \pi_C} \cdot A^{-r_{S,j}}$$

6. The set S-DH contains at most n_{ro} elements, where each one is a possible solution to $\mathrm{DH}(g^{\alpha}, g^{\beta})$. Then $\tilde{\mathcal{B}}^{\mathcal{A}}$ picks $\gamma \xleftarrow{\$}$ L-DH as its output.

In this reduction the simulator $\tilde{\mathcal{B}}^{\mathcal{A}}$ has to guess the client instance target of the Test query, say Π^d_C. The freshness requirement guarantees that a Corrupt query is only possible after the Test query, directed to Π^d_C (or its partner), has been placed. Following the reductionist approach, we showed that the **pairedpwguess** event occurs at most with probability $\epsilon \leq n_{se} \cdot n_{ro} \cdot \mathrm{Adv}^{\mathrm{CDH}}_{G_q}(\tilde{\mathcal{B}}^{\mathcal{A}})$. □

Game $\mathbf{G_6}$: Prevent testing more two passwords per server instance. In P_6 we restrict an adversary, who tries to masquerade as a client, from testing two passwords per session, say π_1 and π_2, in an online dictionary attack. Concretely, let P_6 be identical to P_5 except that if **doublepwserver** event occurs, the protocol stops and the adversary fails.

Let F_5 and F_6 denote the **doublepwserver** event occurring in P_5 and P_6 respectively. By definition it follows that $\mathrm{Succ}^{\mathrm{FtG}}_{P_5}(\mathcal{A}) \wedge \neg F_5 \Leftrightarrow \mathrm{Succ}^{\mathrm{FtG}}_{P_6}(\mathcal{A}) \wedge \neg F_6$.

Claim 7. *For all adversaries $\mathcal{A}$,* $\left|\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_5}(\mathcal{A})\right] - \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_5}(\mathcal{A})\right]\right| \leq \Pr[F_6]$.

Proof. Identical to Claim 2. □

Claim 8. *Given $\mathcal{A}$, there exists a CDH-solver $\hat{\mathcal{B}}^{\mathcal{A}}$ with running time $t' = \mathcal{O}(t + (n_{se} + n_{ex} + n_{ro}) \cdot t_{exp})$ such that:*

$$\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_5}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_6}(\mathcal{A})\right] + n_{ro}^2 \cdot \mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\hat{\mathcal{B}}^{\mathcal{A}}),$$

Proof. We construct an algorithm $\hat{\mathcal{B}}^{\mathcal{A}}$ that solves the CDH problem probability ϵ/n_{ro}^2, where ϵ is the probability of **pairedpwguess** event occurring. On input $(A = g^a, B = g^b)$, $\hat{\mathcal{B}}^{\mathcal{A}}$ simulates G_5 to $\mathcal{A}$ with the following changes:

1. Set $M := A$
2. In a SERVER ACTION 1 to Π_S^j with input $\langle C, X^* \rangle$ set $Y \leftarrow B \cdot g^y$, where $y \xleftarrow{\$} \mathbb{Z}_q$, and sends back $\langle Y, k \rangle$. From P_4 it holds that no unpaired instances can terminate. Specifically, unpaired client and server instances abort in CLIENT ACTION 1 and SERVER ACTION 2 respectively.
3. When $\mathcal{A}$ terminates, for every pair of queries $H_l(C, S, X^*, Y, \sigma_1, \pi_1)$ and $H_l(C, S, X^*, Y, \sigma_2, \pi_2)$, where $\pi_1 \neq \pi_2$, add γ to the S-DH, where:

$$\gamma = A^{-y} \cdot (\sigma_1/\sigma_2)^{(\pi_2 - \pi_1)}$$

4. The set S-DH contains at most $(n_{ro})^2$ elements and each element in the set is a possible solution to $\mathrm{DH}(A, B)$. Then $\hat{\mathcal{B}}^{\mathcal{A}}$ outputs $\gamma \xleftarrow{\$}$ S-DH.

P_6 and P_5 are identical unless the **doublepwserver** event occurs, however, this only occurs with probability $\epsilon \leq n_{ro}^2 \cdot \mathrm{Adv}^{\mathrm{CDH}}_{G_q}(\hat{\mathcal{B}}^{\mathcal{A}})$. The quadratic degradation factor is due to $\hat{\mathcal{B}}^{\mathcal{A}}$ having to guess two queries $H_l(C, S, X^*, Y, \sigma_1, \pi_1)$ and $H_l(C, S, X^*, Y, \sigma_2, \pi_2)$ such that $\sigma_1 = \mathrm{DH}\ (X^*/M^{\pi_1}, Y)$ and $\sigma_2 = \mathrm{DH}\ (X^*/M^{\pi_2}, Y)$. □

Game $\mathbf{G_7}$: Internal password oracle.
In protocol P_7, we consider an internal password oracle $\mathcal{O}_\pi$ who handles every password request and is only available to the challenger. Specifically, the challenger queries the $\mathcal{O}_\pi$ to (i) assign passwords to users, (ii) answer Corrupt queries and (iii) determine if the **correctpw** event occurs.

Claim 9. *For all adversaries $\mathcal{A}$,* $\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_6}(\mathcal{A})\right] = \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A})\right]$.

Proof. It follows from inspection. □

Claim 10. *For all adversaries $\mathcal{A}$,* $\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A})\right] \leq \frac{1}{2} + \frac{n_{se}}{2 \cdot |D|}$.

Proof.

$$\begin{aligned}\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A})\right] = \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A}) \mid \mathrm{correctpw}\right] \cdot \Pr\left[\mathrm{correctpw}\right] \\ + \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A}) \mid \neg\mathrm{correctpw}\right] \cdot \Pr\left[\neg\mathrm{correctpw}\right] \qquad (2)\end{aligned}$$

We know from P_6 that $\mathcal{A}$ can test at most one password per instance in an active attack, then $\Pr\left[\mathrm{correctpw}\right] \leq n_{se}/|D|$. We examine the second term of

Eq. 2. The security experiment requires the adversary to make a Test query to some *PFS-fresh* instance Π_U^i of his choice. It is easy to show that the view of $\mathcal{A}$ is independent of the *sk* on which she is challenged: (i) P_1 prevents two or more instances accepting with the same *sid*, which would violate the partnering definition allowing $\mathcal{A}$ to trivially win, (ii) it follows from P_4 that, before any Corrupt query, only instances that are *paired* instances can reach terminate state – and therefore be target of a Test query – and (iii) from P_5 it holds that for such *paired* instances, the view of $\mathcal{A}$ is independent of *sk* for the session target of the Test query. Then $\Pr\left[\mathrm{Succ}_{P_7}^{\mathrm{FtG}}(\mathcal{A}) \mid \neg\mathrm{correctpw}\right] = 1/2$. ■

4 The SPAKE2 Protocol

4.1 Security of SPAKE2

SPAKE2 protocol is already proven secure in the FtG model [8] without considering any notion of forward secrecy. Here, we show that SPAKE2 also satisfies *weak* forward secrecy in the FtG model assuming the CDH problem is hard in $\mathbb{G}$. The security proof of SPAKE2 is similar to that of PFS-SPAKE2 protocol; the biggest difference is game G_6, where $\mathcal{A}$ is prevented from testing two different passwords when she masquerades as C but also when masquerading as S. The later scenario does not occur in PFS-SPAKE2 since a client instance aborts the protocol whenever it receives an invalid key-confirmation code k.

Theorem 2. *Let P be the protocol specified in Fig. 2 instantiated in group $\mathbb{G}$ and with passwords uniformly distributed over dictionary D. Let $\mathcal{A}$ be an adversary that runs in time t polynomial in λ, makes at most n_{ex}, n_{se}, n_{ro} queries of type execute, send and random oracle. Then:*

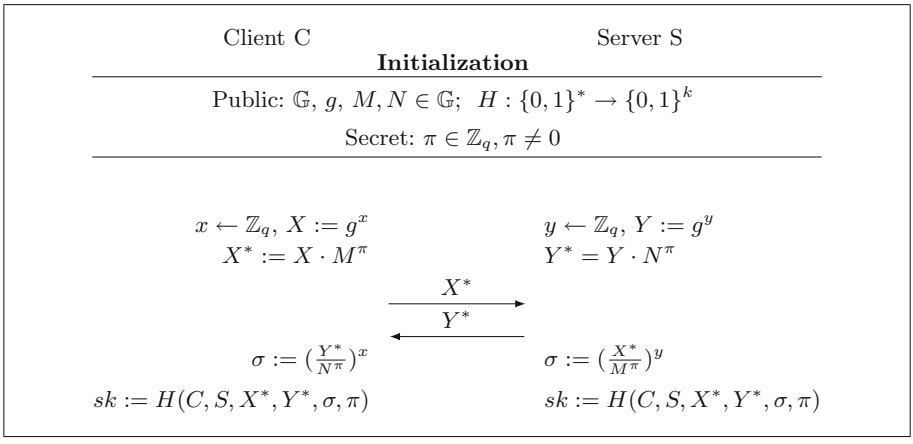

Fig. 2. SPAKE2 protocol.

$$\mathrm{Adv}_{P}^{\text{wFS-FtG}}(\mathcal{A}) \leq \frac{n_{se}}{|D|} + \mathcal{O}\left(\frac{(n_{se}+n_{ex})(n_{se}+n_{ex}+n_{ro})}{q} + n_{ro}\cdot \mathrm{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\mathcal{B}^{\mathcal{A}}) + n_{se}n_{ro}\cdot \mathrm{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\tilde{\mathcal{B}}^{\mathcal{A}}) + n_{ro}^2\cdot \mathrm{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\hat{\mathcal{B}}^{\mathcal{A}})\right),$$

where $\mathcal{B}^{\mathcal{A}}$, $\tilde{\mathcal{B}}^{\mathcal{A}}$ and $\hat{\mathcal{B}}^{\mathcal{A}}$ are CDH-solver algorithms running in time $t' = \mathcal{O}(t + (n_{se}+n_{ex}+n_{ro})\cdot t_{exp})$, where t_{exp} is the time for an exponentiation in $\mathbb{G}$.

Next we provide a sketch of the proof, where we simply write $\mathrm{Succ}_{P_i}^{\text{FtG}}$ instead of $\mathrm{Succ}_{P_i}^{\text{wFS-FtG}}$ to denote the success probability of $\mathcal{A}$ winning in game $\mathrm{G_i}$:

Game $\mathbf{G_0}$: Execution of original protocol.

Game $\mathbf{G_1}$: Force uniqueness of honest instances.
If honest instances generate X^* or Y^* terms equals those seen in previous executions of the protocol, the the protocol stops and $\mathcal{A}$ fails.

$$\mathrm{Succ}_{P_0}^{\text{FtG}} \leq \mathrm{Succ}_{P_1}^{\text{FtG}} + \mathcal{O}\left(\frac{(n_{se}+n_{ex})(n_{se}+n_{ex}+n_{ro})}{q}\right).$$

Game $\mathbf{G_2}$: Simulation without password.
The protocol is simulated without using password information, subsequent random oracle queries made by $\mathcal{A}$ are backpatch to generate consistent views. Also, $\mathcal{A}$ is forced to query the random oracle to compute $sk = H(\cdot)$.

$$\mathrm{Succ}_{P_1}^{\text{FtG}}(\mathcal{A}) \leq \mathrm{Succ}_{P_2}^{\text{FtG}}(\mathcal{A}) + \mathcal{O}(n_{se}/2^{\kappa}).$$

Game $\mathbf{G_3}$: No need to backpatch $\boldsymbol{H_l(\cdot)}$ queries against Execute queries.
We can show that the view of an $\mathcal{A}$ running in time t against $\mathrm{P_2}$ is computationally indistinguishable from that of P_3 via a CDH reduction.

$$\mathrm{Succ}_{P_2}^{\text{FtG}}(\mathcal{A}) = \mathrm{Succ}_{P_3}^{\text{FtG}}(\mathcal{A}) + n_{ro}\cdot \mathrm{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\mathcal{B}^{\mathcal{A}}).$$

where $\mathcal{B}^{\mathcal{A}}$ is a CDH-solver algorithm running in time $t' = \mathcal{O}(t + (n_{se}+n_{ex}+n_{ro})\cdot t_{exp})$ and t_{exp} the time for an exponentiation in $\mathbb{G}$.

Game $\mathbf{G_4}$: Check for successful password guesses.
If before any Corrupt query, the adversary is successful on a password guess against a client or server instance, the protocol stops and the adversary wins.

$$\mathrm{Succ}_{P_3}^{\text{FtG}}(\mathcal{A}) \leq \mathrm{Succ}_{P_4}^{\text{FtG}}(\mathcal{A}).$$

Game $\mathbf{G_5}$: Randomized session keys for paired instances. We build a CDH-solver algorithm from an adversary who manages to compute the sk established at paired instances Π_C^i and Π_S^j, even if $\mathcal{A}$ obtains π_C by adaptively corrupting any of the instances.

$$\Pr\left[\mathrm{Succ}_{P_4}^{\text{FtG}}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}_{P_5}^{\text{FtG}}(\mathcal{A})\right] + n_{se}\cdot n_{ro}\cdot \mathrm{Adv}_{\mathbb{G}}^{\texttt{CDH}}(\tilde{\mathcal{B}}^{\mathcal{A}}).$$

Game $\mathbf{G_6}$: Prevent testing more than one passwords per instance.
If before any Corrupt query, $\mathcal{A}$ manages to test more than one passwords per client or server instance, the protocol stops and the adversary fails. Via a CDH reduction, we show this may happend only with negligible probability.

$$\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_4}(\mathcal{A})\right] \leq \Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_5}(\mathcal{A})\right] + 2n_{ro}^2 \cdot \mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\hat{\mathcal{B}}^{\mathcal{A}}).$$

Game $\mathbf{G_7}$: Internal password oracle.
By inspection P_6 is statistically indistinguishable from P_7. Additionally, let Π^i_U be any instance that remains wFS-fresh and is the target of a Test query. In P_7, provided that $\mathcal{A}$ has not successfully guessed the password, the view of the adversary is independent of the sk^i_U. Then:

$$\Pr\left[\mathrm{Succ}^{\mathrm{FtG}}_{P_7}(\mathcal{A})\right] = \frac{1}{2} + \frac{n_{se}}{2 \cdot |D|}$$

5 Conclusion and Future Work

We proved that SPAKE2 protocol satisfies weak forward secrecy. Note that proving perfect forward secrecy for unmodified SPAKE2 seems to be a harder task. Consider the following scenario: $\mathcal{A}$ masquerades as a client and sends an arbitrary message X^* to a server instance Π^j_S, the latter computes Y^*, its session key, answers back with Y^* and *terminates*. Now $\mathcal{A}$ makes a Test(S, j) query, receives the challenge and then corrupts the tested server instance (as corruption occurred after the Test query the instance Π^j_S remains PFS-fresh). The difficulty is that, even though the proof shows that $\mathcal{A}$ cannot test two passwords per instance, in this particular scenario the simulator cannot determine the password to which $\mathcal{A}$ committed in X^* as she has not asked any random oracle query. Given the difficulty in proving perfect forward secrecy for SPAKE2, we modified the protocol by incorporating key-confirmation codes into it. We proved that the modified protocol satisfies perfect forward secrecy and therefore we called it PFS-SPAKE2.

In future work, we would like to study if the SPAKE2 and PFS-SPAKE2 protocols compose securely with symmetric-key encryption schemes. This question has practical relevance, as in TLS 1.3 the aforementioned primitives would be used not in stand alone operation but as a combined system.

Acknowledgements. The authors are especially grateful to the Luxembourg National Research Fund for supporting this work under CORE project AToMS.

A Terminology from [3]

We introduce the terminology necessary to refer to adversary's actions.

We say "in a CLIENT ACTION k query to Π^i_C" to refer to "in a Send query directed to the client instance Π^i_C that results in CLIENT ACTION k procedure

being executed" and "in a SERVER ACTION k" to refer to "in a Send query directed to the server instance Π_S^j that results in SERVER ACTION k procedure being executed".

A client instance Π_C^i is *paired with* server instance Π_S^j if there was a CLIENT ACTION 0 query to Π_C^i with output $\langle C, X^* \rangle$, a SERVER ACTION 1 to Π_S^j with input $\langle C, X^* \rangle$ and output $\langle S, Y, k \rangle$ and a CLIENT ACTION 1 to Π_C^i with input $\langle S, Y, k \rangle$. A server instance Π_S^j is *paired with* client instance Π_C^i if there was a CLIENT ACTION 0 query to Π_C^i with output $\langle C, X^* \rangle$ and a SERVER ACTION 1 to Π_S^j with input $\langle C, X^* \rangle$ and output $\langle Y, k \rangle$, additionally, if there is a SERVER ACTION 2 query with input k', then there was a previous CLIENT ACTION 1 to Π_C^i with input $\langle Y, k \rangle$ and ouput k'.

Next we define the events that will allow us to proof the security of the protocol by sequence of games.

testpw(C, i, S, π, l): Adversary $\mathcal{A}$ makes (i) an $H_l(C, S, X^*, Y, \sigma, \pi)$ query for some $l \in \{1, 2, 3\}$, (ii) a CLIENT ACTION 0 to Π_C^i with output $\langle S, X^* \rangle$ and (iii) a CLIENT ACTION 1 to Π_C^i with input $\langle C, Y, k \rangle$, where $X^* = X \cdot M^\pi$ and $\sigma = DH(X, Y)$. The associated value to this event is the output of the $H_l(\cdot)$ query, or the k, k', sk_C^i values, respectively for $l = 1, 2, 3$, whichever is set first.

testpw(S, j, C, π, l): $\mathcal{A}$ makes an $H_l(C, S, X^*, Y, \sigma, \pi)$ for some $l \in \{1, 2, 3\}$ and a SERVER ACTION 1 to Π_S^j with input $\langle S, X^* \rangle$ and output $\langle C, Y, k \rangle$, where $X^* = X \cdot M^\pi$ and $\sigma = DH(X, Y)$. The associated value to this event is the output of the $H_l(\cdot)$ query, or the k, k', sk_S^j values, respectively for $l = 1, 2, 3$, whichever is set first.

testpw!(C, i, S, π): In a CLIENT ACTION 1 query with input $\langle \mu, k \rangle$, causes a testpw$(C, i, S, \pi, 2)$ event to occurs, with associated value k.

testexecpw(C, i, S, j, π): $\mathcal{A}$ makes (i) an $H_l(C, S, X^*, Y, \sigma, \pi)$ for some $l \in \{1, 2, 3\}$, where $X^* = X \cdot M^\pi$ and $\sigma = DH(X, Y)$ and (ii) previously an Execute(C, i, S, j) which produces X^*, Y. The associated value to this event is the output of the $H_l(\cdot)$ query, or the k, k', sk_S^j values, respectively for $l = 1, 2, 3$, whichever is set first.

correctpw: Before any Corrupt query, either a testpw!(C, i, S, π_c) event occurs, for some C, i, S, or a testpw(S, j, C, π_c, l) event occurs for some S, j, C and $l \in \{1, 2, 3\}$, where π_c is the correct password.

pairedpwguess: For some client and server instance Π_C^i and Π_S^j respectively, both testpw(C, i, S, π_c, l) and testpw(S, j, C, π, l) event occurs for $l \in \{1, 2, 3\}$, where Π_C^i is paired with Π_S^j, and Π_S^j is paired with Π_C^i after its SERVER ACTION 1.

doublepwserver: Before any Corrupt query, both a testpw(S, j, C, π_1, l) and a testpw(S, j, C, π_2, l) event occurs, for some S, j, π_1 and π_2, with $\pi_1 \neq \pi_2$ and $l \in \{1, 2, 3\}$.

References

1. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Research in Security and Privacy, SP 1992, pp. 72–84 (1992)

2. Boyko, V., MacKenzie, P., 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 (2000). https://doi.org/10.1007/3-540-45539-6_12
3. MacKenzie, P.: The PAK suite: protocols for password-authenticated key exchange. DIMACS Technical report 2002–46 (2002)
4. Hao, F., Ryan, P.: J-PAKE: authenticated key exchange without PKI. Trans. Comput. Sci. **11**, 192–206 (2010)
5. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE password authenticated key exchange protocol. In: IEEE Symposium on Security and Privacy, SP 2015, pp. 571–587. IEEE Computer Society (2015)
6. Wu, T.D.: The secure remote password protocol. In: Proceedings of the Network and Distributed System Security Symposium. The Internet Society (1998)
7. Jablon, D.P.: Strong password-only authenticated key exchange. ACM SIGCOMM Comput. Commun. Rev. **26**(5), 5–26 (1996)
8. 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 (2005). https://doi.org/10.1007/978-3-540-30574-3_14
9. 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 (2000). https://doi.org/10.1007/3-540-45539-6_11
10. 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 (2005). https://doi.org/10.1007/978-3-540-30580-4_6
11. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24
12. Chen, L., Lim, H.W., Yang, G.: Cross-domain password-based authenticated key exchange revisited. ACM Trans. Inf. Syst. Secur. **16**(4), 15:1–15:32 (2014)
13. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 408–432. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_24
14. Kunz-Jacques, S., Pointcheval, D.: About the security of MTI/C0 and MQV. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 156–172. Springer, Heidelberg (2006). https://doi.org/10.1007/11832072_11
15. Vacca, J.R.: Computer and Information Security Handbook, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco (2013)
16. Ladd, W., Kaduk, B.: SPAKE2, a PAKE. Internet-Draft draft-irtf-cfrg-spake2-05, IETF Secretariat, February 2018. http://www.ietf.org/internet-drafts/draft-irtf-cfrg-spake2-05.txt
17. McCallum, N., Sorce, S., Harwood, R., Hudson, G.: Spake pre-authentication. Internet-Draft draft-ietf-kitten-krb-spake-preauth-05, IETF Secretariat, February 2018. http://www.ietf.org/internet-drafts/draft-ietf-kitten-krb-spake-preauth-05.txt
18. Barnes, R., Friel, O.: Usage of spake with TLS 1.3. Internet-Draft draft-barnes-tls-pake-01, IETF Secretariat, April 2018. http://www.ietf.org/internet-drafts/draft-barnes-tls-pake-01.txt

19. Rescorla, E.: The transport layer security (TLS) protocol version 1.3. Internet-Draft draft-ietf-tls-tls13-28, IETF Secretariat, March 2018. http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-28.txt
20. Taylor, D., Wu, T., Mavrogiannopoulos, N., Perrin, T.: Using the secure remote password (SRP) protocol for TLS authentication. RFC 5054, RFC Editor, November 2007
21. Engler, J., Karlof, C., Shi, E., Song, D.: Is it too late for PAKE? In: Web 2.0 Security and Privacy Workshop 2009 (W2SP 2009), May 2009
22. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Des. Codes Cryptogr. **28**(2), 119–134 (2003)
23. Krawczyk, H.: HMQV: a high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_33
24. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptogr. **2**(2), 107–125 (1992). Jun
25. Shoup, V.: On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012 (1999). http://eprint.iacr.org/1999/012
26. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_28
27. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75670-5_1
28. Katz, J., Ostrovsky, R., Yung, M.: Forward secrecy in password-only key exchange protocols. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 29–44. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_3
29. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_21
30. Bellare, M., Rogaway, P.: Provably secure session key distribution: the three party case. In: Leighton, F.T., Borodin, A. (eds.) Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, STOC 1995, pp. 57–66. ACM (1995)
31. Becerra, J., Iovino, V., Ostrev, D., Šala, P., Škrobot, M.: Tightly-secure PAK(E). Cryptology ePrint Archive, Report 2017/1045 (2017). https://eprint.iacr.org/2017/1045
32. 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
33. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. IACR Cryptology ePrint Archive 2004/332 (2004)