Perfect Forward Security of SPAKE2
Michel Abdalla, Manuel Barbosa

▸ To cite this version:

HAL Id: hal-02317002
https://inria.hal.science/hal-02317002
Submitted on 13 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Perfect Forward Security of SPAKE2

Michel Abdalla\textsuperscript{1,2} and Manuel Barbosa\textsuperscript{3}

\textsuperscript{1} DIENS, École normale supérieure, CNRS, PSL University, Paris, France
\textsuperscript{2} INRIA, Paris, France
\textsuperscript{3} FCUP and INESC TEC, Porto, Portugal
michel.abdalla@ens.fr
mbb@fc.up.pt

Abstract. SPAKE2 is a balanced password-authenticated key exchange (PAKE) protocol, proposed by Abdalla and Pointcheval at CTRSA 2005. Due to its simplicity and efficiency, SPAKE2 is one of the balanced PAKE candidates currently under consideration for standardization by the CFRG, together with SPEKE, CPace, and J-PAKE. In this paper, we show that SPAKE2 achieves perfect forward security in the random-oracle model under the Gap Diffie-Hellman assumption. Unlike prior results, which either did not consider forward security or only proved a weak form of it, our results guarantee the security of the derived keys even for sessions that were created with the active involvement of the attacker, as long as the parties involved in the protocol are not corrupted when these sessions take place. Finally, our proofs also demonstrate that SPAKE2 is flexible with respect to the generation of its global parameters \( M \) and \( N \). This includes the cases where \( M \) is a uniform group element and \( M = N \) or the case where \( M \) and \( N \) are chosen as the output of a random oracle.

1 Introduction

Password-authenticated key exchange (PAKE) allows users to establish session keys among themselves with the help of a short secret, known as a password, which can be drawn from a small set of possible values. Passwords can be represented in shorter human-readable formats and distributed/stored using a wider range of mechanisms, which are important requirements in many applications. One important use-case is when secrets must be memorized by humans.

Since the seminal work by Bellovin and Merritt [BM92], several PAKE protocols have appeared in the literature, achieving different levels of security (such as indistinguishability-based security or universal composability) under a variety of assumptions.
Recently, the Crypto Forum Research Group (CFRG)\(^4\), which is an IRTF (Internet Research Task Force) research group focused on the discussion and review of uses of cryptographic mechanisms, started a PAKE selection process with the goal of providing recommendations for password-based authenticated key establishment in IETF protocols. Originally, 4 candidates were under consideration by the CRFG in the balanced PAKE category: SPEKE [Jab97, Mac01, HS14], SPAKE2 [AP05, LK19], J-PAKE [HR10, ABM15], and Cpace [HL18, HL19]. Currently, only SPAKE2 and Cpace are still under consideration in the balanced PAKE category.

**Security analyses for SPAKE2.** In the original security analysis [AP05], Abdalla and Pointcheval proved that SPAKE2 achieves implicit authentication in the indistinguishability-based model by Bellare, Pointcheval, and Rogaway [BPR00]. Their security analysis, however, did not take corruption queries into account, which are needed for proving forward security. To address this shortcoming, Becerra, Ostrev, and Skrobot [BOS18] recently provided a proof of weak forward security for SPAKE. Unlike perfect forward security, weak forward security only guarantees the security of sessions created without an active intervention by the attacker as long as the involved parties remain uncorrupted during the execution of these sessions [KOY03, Kra05].

In addition to proving weak forward security in [BOS18], the authors also provide a proof of perfect forward security for a variant of SPAKE2 which includes explicit authentication and in which one of the flows is not encrypted with the password. The latter result, however, seems to be a particular case of the scheme proven secure by Abdalla et al. in [ABC+06].

**Our contributions.** In this work, we provide further security analyses for SPAKE2 [AP05, LK19]. More precisely, we demonstrate that SPAKE2 achieves perfect forward security even without key confirmation. Our proof of security is based on the Gap Diffie-Hellman assumption [OP01] in the random-oracle model [BR93]. Note that this does not contradict the impossibility result about the perfect forward security of 2-message key-exchange protocols in the HMQV [Kra05] paper since the SPAKE2 protocol assumes the pre-existence of secure shared state between parties which is the password itself. We also include an analysis of the protocol when key confirmation is added, which yields a better bound.

Finally, our proofs also demonstrate that SPAKE2 is flexible with respect to the generation of its global parameters \(M\) and \(N\). This includes the cases where \(M\) is a uniform group element and \(M = N\) or the case where \(M\) and \(N\) are chosen as the output of a random oracle.

**Difference to previous version.** The proof of security of SPAKE2 without key confirmation has been updated to refine the analysis of a bad event that was not fully covered by the initial version. To deal with this, we added an extra game hop, for which we give two alternative justifications. In the first one, which yields a tight reduction to GapSqDH, we assume an algebraic adversary. In the second one, we get a looser reduction to GCDH via the splitting lemma of [PS00, AP05]. We also added the analysis of SPAKE2 with key confirmation.

## 2 Preliminaries

**Notation.** We write \(x \leftarrow y\) for the action of assigning the value \(y\) to the variable \(x\). We write \(x_1, \ldots, x_n \leftarrow X\) for sampling \(x_1, \ldots, x_n\) from a finite set \(X\) uniformly at random. If \(A\) is a probabilistic algorithm we write \(y_1, \ldots, y_n \leftarrow A(x_1, \ldots, x_n)\) for the action of running \(A\) on inputs \(x_1, \ldots, x_n\) with independently chosen coins, and assigning the result to \(y_1, \ldots, y_n\). PPT as usual abbreviates probabilistic polynomial-time. We use notation \(T[x]\) to denote access to a dictionary/table \(T\) at index \(x\), and \(\emptyset\) to denote the empty table. We abuse notation and use \(T\) to also represent the set of assigned indices in table \(T\).

**Games.** We use the code-based game-playing language [BR04]. Each game has an INITIALIZE and a FINALIZE procedure. It also has specifications of procedures to respond to an adversary’s various

\(^4\) [https://irtf.org/cfrg](https://irtf.org/cfrg)
queries. A game is run with an adversary $A$ as follows. First $\text{Initialize}$ runs and its outputs are passed to $A$. Then $A$ runs and its oracle queries are answered by the procedures of the game. When $A$ terminates, its output is passed to $\text{Finalize}$ which returns the outcome of the game.

3 SPAKE2

Fig. 1 shows a SPAKE2 protocol execution between a user $U$ and a server $S$.

![Fig. 1. The SPAKE2 protocol [AP05].](image)

4 Security Model

Our proof of security uses the indistinguishability-based model by Bellare, Pointcheval, and Rogaway [BPR00] and its extension to multiple $\text{Test}$ queries proposed by Abdalla, Fouque, and Pointcheval (AFP) [AFP05, AFP06]. In the latter, all $\text{Test}$ queries are answered with the same challenge bit $b$.

The security game already instantiated with SPAKE2 can be seen as $G_1$ in Fig. 3. The name spaces for servers $S$ and users $U$ are assumed to be disjoint; oracles reject queries inconsistent with these name spaces.

In the following, we describe more precisely the state of a party instance, the notion of partnering, and the freshness condition used in the proof.

**Definition 4.1 (Instance state).** The state of an instance $i$ at party $P$, denoted $\pi_i^P$, is a tuple of the form $(e, tr, K, ac)$ where

- $e$ is the secret exponent ($x$ or $y$) chosen by the party in that instance
- $tr$ is a session trace of the form $(U, S, X^*, Y^*)$
- $K$ is the accepted session key
- $ac$ is a boolean flag that indicates whether the instance accepted ($ac = T$) or is expecting a response ($ac = F$)

We will use $\pi_i^P.e$, $\pi_i^P.tr$, $\pi_i^P.K$, $\pi_i^P.ac$ to denote the individual components of the state.
Definition 4.2 (Partnering). A server instance \( \pi_S^i \) and a user instance \( \pi_U^j \) are partnered iff

\[
\pi_S^i.ac = T \land \pi_U^j.ac = T \land \pi_S^i.tr = \pi_U^j.tr
\]

Two user instances are never partnered. Two server instances are never partnered.

This is a definition in the style of matching-conversations. Note that partnering together with correctness implies that the the associated keys are the same.

Definition 4.3 (Freshness). Instance \( i \) at party \( P \) is fresh, written \( \text{Fresh}(P,i) \) if and only if all the following conditions hold:

1. the instance accepted;
2. the instance was not queried to \text{Test} or \text{Reveal} before;
3. At least one of the following conditions holds:
   (a) The instance accepted during a query to \text{Execute};
   (b) There exists more than one partner instance;
   (c) No partner instance exists and \text{Corrupt}(U,S) was not called prior to acceptance;
   (d) A unique fresh partner instance exists.

5 Assumptions

The perfect forward security of SPAKE2 is based on the difficulty of solving the Computational Diffie-Hellman (CDH) problem by attackers that have access to a Decisional Diffie-Hellman (DDH) oracle. This is usually known as the Gap Diffie-Hellman problem (GCDH) [OP01].

The proof is carried out in the Random-Oracle (RO) model and, to prove perfect forward secret, we need to restrict our attention to algebraic adversaries [FKL18]. Note that the GCDH problem has been shown to be equivalent to the standard discrete logarithm problem (DL) in the Algebraic and Generic Group models [FKL18, Los19] so, our proof implies reductions to the DL problem in these idealized models of computation.\(^5\)

Our proof is structured to highlight which problem an attacker would need to solve to break the protocol. On the one hand, we give tight reductions to well-known computational assumptions. On the other hand, it follows sequence of hops that clarifies how the assumptions map to the generation of SPAKE2 global parameters and runtime operation. We show that, as expected, passive behavior by the attacker implies that it needs to solve the standard Gap CDH problem to break the established session. Intuitively attackers have access to a DDH oracle by comparing random oracle outputs to the keys revealed by the game. Active attackers that may try to take advantage of leaked passwords must compute the CDH of the global parameters \( M \) and \( N \) or at least compute the squared version of this problem \( \text{CDH}(M,M) \), with the help of a DDH oracle. We show that the protocol is flexible with respect to the generation procedure, provided that the previous problem is hard to solve. This includes, in particular, the case where \( M \) is a uniform group element and \( M = N \).

To this end, we formalize a family of assumptions parametrized by a distribution \( \mathcal{D} \) that outputs a pair of elements in \( \mathbb{Z}_q \), where \( q \) is a large prime. The assumptions are stated with respect to a group \( G \) of order \( q \) and generator \( g \).

\(^5\) There is a subtlety here which further justifies our presentation with gap assumptions. Our proof in the algebraic model uses the gap oracles on values queried by attackers to a random oracle. As random oracles are an abstraction of local computation, it is perhaps unreasonable to ask that the algebraic adversary provides us with representations of those values. If this is not required from the adversary, then our proof does not necessarily imply a reduction to DL for algebraic attackers, and we must rely on the gap assumptions. For this reason, in our proof we only make explicit use of the algebraic nature of the adversary onmessages that it explicitly delivers.
Definition 5.1 (Gap Computational Diffie-Hellman (D-GCDH)). The D-GCDH assumption states that, for any ppt adversary $A$, the following probability is small.

$$\text{Adv}_{A}^{\text{GCDH}} := \Pr \left[ Z = g^{xy} : Z \leftarrow A^{\text{DDH}(\cdots)}(g^x, g^y); x \leftarrow \mathbb{Z}_q; y \leftarrow \mathcal{D}(x) \right]$$

The standard GCDH assumption is a particular case when $\mathcal{D}$ is the uniform distribution over $\mathbb{Z}_q$. We denote this by $\mathcal{U}$-GCDH. The Gap Squared Diffie-Hellman (GsqDH) assumption is another particular case, when we restrict the previous case to $\mathcal{D}$ outputting $x = y$. For simplicity, in our analysis, we will restrict our attention to global parameter sampling operations where we exclude the possibility that $x = 0$ or $y = 0$. This means that the bound for our proof includes an extra statistical when we remove this restriction (this will be of the form $(q_s + q_e)/q$ for standard GCDH). We will also rely on the weaker Gap Discrete Logarithm assumption.

Definition 5.2 (Gap Discrete Logarithm (GDL)). The GDL assumption states that, for any ppt adversary $A$, the following probability is small.

$$\text{Adv}_{A}^{\text{GDL}} := \Pr \left[ x' = x : x' \leftarrow A^{\text{DDH}(\cdots)}(g^x); x \leftarrow \mathbb{Z}_q \right]$$

Finally, for the version of our result where we do not rely on the assumption that the adversary is algebraic, we require the following assumption introduced in [AP05], which we extend to the gap version.

Definition 5.3 (Gap Discrete Logarithm Password-based Chosen-basis Computational Diffie-Hellman assumption (GPCCDH). The D-GPCCDH assumption states that, for any ppt adversary $A$, the following probability is small.

$$\text{Adv}_{A}^{\text{D-GPCCDH}} := \Pr \left[ C = \text{CDH}(A_1/B_1^{\text{pw}}, A_2/B_2^{\text{pw}}) \left| b_1 \leftarrow \mathbb{Z}_q; b_2 \leftarrow \mathcal{D}(b_1); A_1 \leftarrow g_b^b; B_1 \leftarrow g^b; B_2 \leftarrow g^{b_2}; A_2 \leftarrow \text{GPCCDH}(\cdots)(B_1, B_2, A_1); \text{pw} \leftarrow \mathcal{P}; C \leftarrow A^{\text{GPCCDH}(\cdots)}(\text{pw}, \text{st}) \right]$$

It was shown in [AP05] that the assumption above without the DDH oracle is implied by the standard CDH problem. The proof extends naturally to gap versions of the problems, so we have the following.

Lemma 5.4 ([AP05], Lemma 3.4). For all GPCCDH adversaries $A$ there exists an adversary $B$ such that:

$$\frac{2}{|\mathcal{P}|} \geq \text{Adv}_{A}^{\text{D-GPCCDH}} \geq \frac{1}{|\mathcal{P}|} + \epsilon \implies \text{Adv}_{B}^{\text{D-GCDH}} \geq \frac{|\mathcal{P}|}{128} \epsilon^3.$$
In our proof, we actually only require a PRF that is pseudorandom for one input per key, and we will denote this weaker requirement by $\text{1PRF}$.

Finally, in the proof of SPAKE2 with key confirmation, we require the standard unforgeability property for a one-time MAC.

**Definition 5.6 (One-time MAC security).** A MAC scheme with key space $K$ and domain $D$ is one-time unforgeable if, for any ppt adversary $A$, the following advantage term is small.

$$\text{Adv}_A^{\text{1UF}} := \Pr\left[t = \text{MAC}(K, m) : (m, t) \leftarrow A(); K \leftarrow \mathcal{K}\right]\right]$$

### 6 Perfect Forward Security Proof

**Theorem 6.1.** SPAKE2 is tightly PFS-secure in the random-oracle model under the Gap Squared Diffie-Hellman assumption. More precisely, for every algebraic attacker $A$ against SPAKE2, there exist attackers $B_1$ and $B_3$ against the Gap Diffie-Hellman problem, attacker $B_2$ against the Gap Discrete Logarithm problem, and attacker $B_4$ against the Gap Squared Diffie-Hellman problem such that

$$\text{Adv}_A^{\text{SPAKE2}}(\cdot) \leq \frac{q_s}{|P|} + \text{Adv}_{B_1}^{\text{GCDH}}(\cdot) + \text{Adv}_{B_2}^{\text{GDL}}(\cdot) + \text{Adv}_{B_3}^{\text{D}^*\text{-GCDH}}(\cdot) + \text{Adv}_{B_4}^{\text{GSeDH}}(\cdot) + \frac{(q_s + q_e)^2}{2q}$$

Here there is an extra $q_s/|P|$ term that is a consequence of our game hopping strategy, which has been chosen for clarity by separating the weak forward secrecy property from the strong one that relies on algebraic adversaries. Using a different sequence of hops, one can remove the extra term by aggregating the two bad events that give rise to these terms into a single one.

**Theorem 6.2.** SPAKE2 is PFS-secure in the random-oracle model under the Gap Computational Diffie-Hellman assumption. More precisely, for every attacker $A$ against SPAKE2, there exist attackers $B_1$ and $B_3$ against the Gap Diffie-Hellman problem, attacker $B_2$ against the Gap Discrete Logarithm problem, and attacker $B_4$ against the Gap GPCCDH Diffie-Hellman problem such that

$$\text{Adv}_A^{\text{SPAKE2}}(\cdot) \leq \frac{q_s}{|P|} + \text{Adv}_{B_1}^{\text{GCDH}}(\cdot) + \text{Adv}_{B_2}^{\text{GDL}}(\cdot) + \text{Adv}_{B_3}^{\text{D}^*\text{-GCDH}}(\cdot) + 2q_s \cdot \text{Adv}_{B_4}^{\text{GPCCDH}}(\cdot) + \frac{(q_s + q_e)^2}{2q}$$

Note that in the above bound there exists an extra $2q_s/|P|$ within the GPCCDH advantage, since we only have guarantees based on GCDH once that advantage term grows above this limit, so this bound is worse than the one in Theorem 6.1.

We give a joint proof for both theorems, since they only differ in the proof of the last game hop. The code of all games and adversaries is given in Appendix A.

**Proof.** We prove the security of SPAKE2 using a sequence of games shown in Figs. 3 to 5. In Table 1 we give a summary of the meaning of bad events in each game.

**GAME G_1.** This is the initial security game, so we have:

$$\text{Adv}_A^{\text{SPAKE2}}(\cdot) = \left| \Pr[G_1 \Rightarrow T] - 1/2 \right|$$

**GAME G_2.** From game $G_1$ to game $G_2$ we introduce a bad flag $\text{bad}_2$. The $\text{bad}_2$ flag is set whenever

- a session is about to be accepted on the server side that collides on $(U, S, X^*, Y^*)$ with any previously accepted session;
- a session is about to be accepted on the user side that collides on $(U, S, X^*, Y^*)$ with any previously accepted user session.

Note that in the above bound there exists an extra $2q_s/|P|$ within the GPCCDH advantage, since we only have guarantees based on GCDH once that advantage term grows above this limit, so this bound is worse than the one in Theorem 6.1.

We give a joint proof for both theorems, since they only differ in the proof of the last game hop. The code of all games and adversaries is given in Appendix A.
If $bad_2$ is set, then the oracle call is ignored and the session is not accepted.

Note that this implies that all accepted sessions on the server (resp. client) side are unique and that no session can have more than one partner. The two games are identical until bad occurs, and this event can be bound by a statistical term. More precisely, for an attacker placing at most $q_e$ queries to $\text{EXECUTE}$ and $q_s$ queries to the $\text{SEND}$ oracles, we have for large $q$ denoting the order of the group:

$$| \Pr[G_1 \Rightarrow T] - \Pr[G_2 \Rightarrow T]| \leq \frac{(q_e + q_s)^2}{2q}$$

Here, $q_e + q_s$ denotes the maximum number of accepted sessions (we count passively partnered sessions as one here) and this is a birthday bound computed pessimistically for an attacker that makes accepted sessions collide by fixing the user, server and one of the transmitted group elements in all accepted sessions.

**GAME $G_3$.** In this game we take advantage of the fact that accepted sessions have unique traces (modulo partnering) to make the freshness condition explicit in the game (we extend the state of sessions to keep track of freshness with a new field $fr$). We also remove the code that refers to $bad_2$. Since nothing changes, we have

$$\Pr[G_3 \Rightarrow T] = \Pr[G_2 \Rightarrow T]$$

**GAME $G_4$.** In this game we make the keys of sessions accepted as a result of calls to $\text{EXECUTE}$ independent from the random oracle accessible to the attacker. These sessions obtain keys from a new random oracle $T_e$ that is indexed by the session trace $(U, S, X^*, Y^*)$. Note that, due to the guarantee of session uniqueness on the server and client sides, there is no room for ambiguity. We set a bad flag $bad_4$ if ever the answers given to the attacker could be inconsistent with the previous game: either because a new accepted session in $\text{EXECUTE}$ collides with a previous call to $H$, or because a new call to $H$ collides with a previously accepted session in $\text{EXECUTE}$. The games are identical until bad occurs, so we have:

$$| \Pr[G_3 \Rightarrow T] - \Pr[G_4 \Rightarrow T]| \leq \Pr[G_4 \Rightarrow bad_4]$$

The probability of $bad_4$ occurring in $G_4$ can be tightly reduced to the standard Gap CDH problem, which means that:

$$\Pr[G_4 \Rightarrow bad_4] \leq \text{Adv}^\text{H-GCDH}_{T_e}(\cdot)$$

The reduction to Gap CDH is given in Lemma 6.4.

**GAME $G_5$.** In game $G_5$ we again rearrange code in preparation for the next hop. We remove the code that deals with bad events, and we change $\text{EXECUTE}$ so that nothing depends on passwords. We can do this because the random oracle that generates these keys only depend on the trace, which can be sampled independently of passwords. We have

$$\Pr[G_5 \Rightarrow T] = \Pr[G_4 \Rightarrow T]$$

**GAME $G_6$.** In this game we change the way in which we generate keys for $\text{SEND}$ queries. We treat corrupt sessions, for which the attacker may trivially compute the key, as before. The remaining keys are derived using an independent random oracle $T_s$. We set a bad flag $bad_6$ if ever there could be an inconsistency with the main random oracle. As before, this means that:

$$| \Pr[G_5 \Rightarrow T] - \Pr[G_6 \Rightarrow T]| \leq \Pr[G_6 \Rightarrow bad_6]$$

We also observe that, since all fresh sessions that can be tested now have keys derived using random oracles independent from $H$, accepted keys of fresh sessions are independently distributed from anything else in the game. This means that the attacker has no information on bit $b$ and has therefore 0 advantage in this game. From the previous equations we can derive that:

$$\text{Adv}^{\text{SPAKE2}}_{\Delta}(\cdot) \leq \frac{(q_s + q_e)^2}{2q} + \text{Adv}^\text{H-GCDH}_{T_e}(\cdot) + \Pr[G_6 \Rightarrow bad_6]$$
To bound the probability of \( b_{ad_6} \) we need to continue our sequence of hops.

**Game \( G_7 \).** We modify the way in which \( X^* \) and \( Y^* \) are generated, removing the component that depends on the password. The distribution of the \( X^* \) and \( Y^* \) values does not change with respect to the previous game, but we need to adapt the way in which the Diffie-Hellman tuples are computed, to make sure we obtain the same values as in the previous game. After we make this change, passwords associated with fresh sessions are still (and only) used to check for the \( b_{ad} \) event. We add a flag \( b_{ad_7} \) to account for the unlikely case that a password occurring in the game (either queried by the adversary or sampled by the challenger) collides with the secret exponents used in \( X^* \) and \( Y^* \) sampled by the challenger. If this happens, we reset \( b_{ad_6} \) at the end of the game. The probability of this event occurring can be easily reduced to the GDL problem, as shown in Lemma 6.5. We therefore have:

\[
\Pr[G_6 \Rightarrow b_{ad_6}] \leq \Pr[G_7 \Rightarrow b_{ad_6}] + \operatorname{Adv}_{GDL}^{GDL}(\cdot)
\]

**Game \( G_8 \).** We now change the checking of \( b_{ad_6} \) to allow delaying all password samplings as late as possible in the game: upon corruption or at the end of the game. This means we stop verifying password equality when checking for \( b_{ad_6} \) in the SEND oracles (there is no impact in the random oracle simulation because the attacker provides the password explicitly there). Without additional changes, we might not be able to tightly relate the probability of \( b_{ad_6} \) in this new game to its probability in the previous game, since we have relaxed the check that detects it. We therefore need to add additional checks that allow us to continue the proof. There are two important cases to distinguish:

- the attacker causes \( b_{ad_6} \) for a password that has not been generated yet (this includes all occurrences in the SEND oracles and some in the random oracle simulation).
- the attacker causes \( b_{ad_6} \) by querying the random oracle on the correct password after the password has been corrupted (intuitively it does not need to guess the password in this case).

For the second case we set a bad flag \( b_{ad_8} \) and no longer set \( b_{ad_6} \).

For the other cases we add the potentially problematic random oracle entries to a list \( T_{bad_4} \), so that we can delay the analysis of potential inconsistencies until the end of the game. At the end of the game we split the checking of \( b_{ad_6} \) into two cases. We set a bad flag \( b_{ad_8} \) if in the \( T_{bad} \) log of problematic \( H \) calls there are two entries consistent with a single entry in \( T_s \), i.e., with different passwords; in this case we no longer set \( b_{ad_6} \). Finally, we check for \( b_{ad_6} \) simply by going through the list of problematic entries and checking whether there is a password match.

The combined actions of the three bad flags in this game account for all possible cases in which \( b_{ad_6} \) can occur in game \( G_7 \), so we can write:

\[
\Pr[G_6 \Rightarrow b_{ad_7}] \leq \Pr[G_8 \Rightarrow b_{ad_6}] + \Pr[G_8 \Rightarrow b_{ad_8}] + \Pr[G_8 \Rightarrow b_{ad_8}]
\]

The probability of \( b_{ad_8} \) occurring in \( G_8 \) can be tightly reduced to Gap CDH, where we use here the generalized version for any parameter distribution \( D \) where the CDH problem is hard to compute with the help of a DDH oracle,\(^6\) the reduction is given in Lemma 6.6.

We complete the analysis of this game by further observing that the probability of \( b_{ad_6} \) happening is easy to bound. Indeed, the size of the log in which \( b_{ad_6} \) is checked has size at most \( q_\star \) (since there can be no duplicates for the same trace) and all entries are added to this set before the corresponding password is sampled. We therefore have

\[
\Pr[G_8 \Rightarrow b_{ad_6}] \leq \frac{q_\star}{|P|}
\]

Putting these results together, we obtain:

\[
\Pr[G_7 \Rightarrow b_{ad_6}] \leq \Pr[G_8 \Rightarrow b_{ad_6}] + \Pr[G_8 \Rightarrow b_{ad_8}] + \Pr[G_8 \Rightarrow b_{ad_8}] \leq \frac{q_\star}{|P|} + \operatorname{Adv}_{GDL}^{GDL}(\cdot)
\]

\(^6\) Note that this does not weaken our result, as one can take \( D \) to be the uniform distribution. On the other hand, it makes it clear that different parameter generation procures can be used and our proof stil applies.
To complete the proof we need to bound the probability of \( \text{bad}_8^2 \) occurring in \( G_8 \), and so we continue our sequence of hops.

GAME \( G_9 \). For clarity of presentation, in game \( G_9 \) we perform a cleanup, removing all code unrelated to \( \text{bad}_8^2 \). We also add the possibility that the attacker passes an additional parameter \( \text{alg} \) when it delivers a message. This additional parameter is ignored for now, so clearly

\[
\Pr[ G_8 \Rightarrow \text{bad}_8^2 ] = \Pr[ G_9 \Rightarrow \text{bad}_8^2 ]
\]

To complete the proof for Theorem 6.2 we give in Lemma 6.8 a reduction of the above probability to GPCCDH, which yields the desired bound:

\[
\text{Adv}_{\mathcal{A}}^{\text{SPAKE2}} ( ) \leq \frac{q_s}{|P|} + \text{Adv}_{\mathcal{B}_1}^{\mathcal{U}^{\mathcal{GCDH}}}( ) + \text{Adv}_{\mathcal{B}_2}^{\mathcal{GDL}}( ) + \text{Adv}_{\mathcal{B}_3}^{\mathcal{D}^{\ast} - \mathcal{GCDH}}( ) + 2q_s \cdot \text{Adv}_{\mathcal{B}_4}^{\text{GPGCDH}}( ) + \frac{(q_s + q_b)^2}{2q}
\]

We proceed with a final game hop to complete the proof for Theorem 6.1.

GAME \( G_{10} \). This is our final game. Here we restrict our attention to algebraic adversaries. This means that, when performing an active attack, the adversary provides an additional input \( \text{alg} \) to the \( \text{SEND} \) oracles that gives a representation of the group element it is producing in terms of the other group elements it observed in the game. These elements include \( g, M, N \) and all the honestly generated \( X^* = M^2 \) and \( Y^* = N^y \). We note that this means that any group element produced by \( \mathcal{A} \) can be rewritten as a representation of the form \( \text{alg} = [ (g^x), (M^y), (N^e) ] \).

We now add a bad flag \( \text{bad}_{10} \) that excludes some password guessing events related to \( \text{bad}_8^2 \). When these happen, we no longer set \( \text{bad}_8^2 \), whereas this would have been set in the previous game. This means that:

\[
\Pr[ G_9 \Rightarrow \text{bad}_8^2 ] \leq \Pr[ G_{10} \Rightarrow \text{bad}_8^2 ] + \Pr[ G_{10} \Rightarrow \text{bad}_{10} ]
\]

Event \( \text{bad}_{10} \) is detected whenever a password is sampled at the moment of corruption, and it hits an already fixed representation of a group element produced by the attacker. The offending values are \( b + \lambda (c - \text{pw}) \) for group elements delivered at \( \text{SENDTERM} \) and \( b - \text{pw} + \lambda c \) for group elements delivered at \( \text{SENDRESP} \). If this event is detected, then we never set \( \text{bad}_8^2 \), so in this game the analysis of \( \text{bad}_8^2 \) can proceed under the assumption that \( \text{bad}_{10} \) did not occur. The probability of this event is easy to bound as

\[
\Pr[ G_{10} \Rightarrow \text{bad}_{10} ] \leq \frac{q_s}{|P|}
\]

The remaining cases where we set \( \text{bad}_8^2 \) can be reduced to the Gap Squared Diffie-Hellman problem (GSqDH), as shown in Lemma 6.7. This completes the proof and yields the bound:

\[
\text{Adv}_{\mathcal{A}}^{\text{SPAKE2}} ( ) \leq \frac{2q_s}{|P|} + \text{Adv}_{\mathcal{B}_1}^{\mathcal{U}^{\mathcal{GCDH}}}( ) + \text{Adv}_{\mathcal{B}_2}^{\mathcal{GDL}}( ) + \text{Adv}_{\mathcal{B}_3}^{\mathcal{D}^{\ast} - \mathcal{GCDH}}( ) + \text{Adv}_{\mathcal{B}_4}^{\text{GSqDH}}( ) + \frac{(q_s + q_b)^2}{2q}
\]

A different sequence of games permits removing a \( q_s/|P| \) term, yielding the expected bound for PAKE. \( \Box \)

**Corollary 6.3.** SPAKE2 is weak PFS-secure in the random-oracle model under the Gap Computational Diffie-Hellman assumption. More precisely, for every attacker \( \mathcal{A} \) against SPAKE2, there exist attackers \( \mathcal{B}_1 \) and \( \mathcal{B}_3 \) against the Gap Diffie-Hellman problem and attacker \( \mathcal{B}_2 \) against the Gap Discrete Logarithm problem such that

\[
\text{Adv}_{\mathcal{A}}^{\text{SPAKE2}} ( ) \leq \frac{q_s}{|P|} + \text{Adv}_{\mathcal{B}_1}^{\mathcal{U}^{\mathcal{GCDH}}}( ) + \text{Adv}_{\mathcal{B}_2}^{\mathcal{GDL}}( ) + \text{Adv}_{\mathcal{B}_3}^{\mathcal{D}^{\ast} - \mathcal{GCDH}}( ) + \frac{(q_s + q_b)^2}{2q}
\]

**Proof.** In the weak forward secrecy model, corruption of password \( \text{pw}_{138} \) makes all fresh session where the adversary was active immediately unfresh and, furthermore, corruption is not allowed if such a
session was already tested. It is easy to see that a simple adaptation of the previous proof can be done for this setting, where the probability of event $bad_2$ occurring is 0 and the advantage of the adversary is still 0: for such cases where the event would occur we simply return $T_s[tr]$ as the answer to the problematic $H$ query.

We now prove the auxiliary lemmas supporting the proofs of Theorems 6.1 and 6.2.

**Lemma 6.4.** For every attacker $A$, there exists an attacker $B_1$, such that
\[
Pr[\mathcal{G}_4 \Rightarrow bad_4] \leq \text{Adv}_{\mathcal{B}_4}^{\mathcal{H}-\text{GCDH}}()
\]

**Proof.** Let us consider the GCDH attacker $B_1$ in Fig. 6. It gets a generalized challenge of the form $(X_1, \ldots, X_{q_x}, Y_1, \ldots, Y_{q_x})$ and finds $\text{CDH}(X_i, Y_j)$, for some $1 \leq i, j \leq q_x$. This problem is tightly equivalent to $\mathcal{H}$-GCDH by random self-reducibility. The attacker embeds $X_i$ and $Y_j$ in traces for protocol executions that result from calls to EXECUTE. For all calls to SEND oracles, the attacker runs everything as in $\mathcal{G}_4$. The DDH oracle is used to check for the bad event, in which case a solution to the GCDH problem was found. It uses the DDH oracle whenever the rules of the game require it, and it can also use it to check if $bad_4$ occurred, in which case it can solve the Gap CDH challenge. $\square$

**Lemma 6.5.** For every attacker $A$, there exists an attacker $B_2$, such that
\[
Pr[\mathcal{G}_7 \Rightarrow bad_7] \leq \text{Adv}_{\mathcal{B}_7}^{\text{GDL}}()
\]

**Proof.** Let us consider a GDL attacker $B_2$ in Fig. 6. It gets a challenge of the form $A$ and finds $\text{DL}(A)$. The attacker uses $A$ to compute $X^* = A \cdot g^\Delta$ and $Y^* = A \cdot g^\Delta$. Note that the distribution of these values is correct. The checking of CDH tuples, which is needed for correctly maintaining random-oracle consistency, is carried out using the DDH oracle.

If $bad_7$ occurs, our reduction recovers the discrete logarithm of $A$ offset by a known quantity. Indeed, suppose the attacker guessed $pw = x$ or $pw = y$ in game $\mathcal{G}_7$, which means we have $X^* = g^{mpw}$ or $Y^* = g^{npw}$. In our reduction we program both $X^*$ and $Y^*$ as $g^{a+\Delta}$, where $a$ is the unknown discrete logarithm, and we know all of $m$, $pw$ and $\Delta$. This means that, if the guess was successful, we can compute the discrete logarithm as $a = mpw - \Delta$ or $a = npw - \Delta$. $\square$

**Lemma 6.6.** For every attacker $A$, there exists an attacker $B_3$, such that
\[
Pr[\mathcal{G}_7 \Rightarrow bad_8] \leq \text{Adv}_{\mathcal{B}_8}^{\mathcal{D}^*-\text{GCDH}}()
\]

**Proof.** Let us consider a $\mathcal{D}^*$-GCDH attacker $B_3$ in Fig. 6. It gets a challenge of the form $(M, N)$ and finds $\text{CDH}(M, N)$. The attacker uses $(M, N)$ as the variables of the same name in game $\mathcal{G}_s$. The checking of CDH tuples, which is needed for correctly maintaining random-oracle consistency and detecting bad events, is carried out using the DDH oracle.

Once the game terminates, the attacker checks for the occurrence of $bad_8$ and if offending entries exist, it recovers $(X^*, Y^*, pw, Z)$ and $(X^*, Y^*, pw', Z')$ such that $pw \neq pw'$ and the following holds for known $x$ or known $y$, and known $pw$ and $pw'$:

\[
\text{CDH}(g^{mx}/M^{pw}, g^{ny}/N^{pw}) = Z \land \text{CDH}(g^{mx}/M^{pw'}, g^{ny}/N^{pw'}) = Z'
\]

Letting $Z = g^z$, $Z' = g^{z'}$, we can rewrite these equations as
\[
\begin{cases}
m(x - pw) \cdot n(y - pw) = z \\
m(x - pw') \cdot n(y - pw') = z'
\end{cases}
\]

Note that it actually suffices to have a restricted DDH oracle in which one of the inputs is fixed as in the Strong Diffie-Hellman (SDH) problem [ABR01], so a tight reduction to SDH is also possible. Furthermore, storing all possible values that cause the bad event and returning one at random gives a reduction to CDH that loses a linear factor in the maximum size of the random-oracle table.
Let us assume that the attacker knows \( x \) (the other case is symmetric). We scale the equations and rearrange the formula as follows:

\[
\begin{align*}
\left\{ \begin{array}{l}
m(x - pw)(x - pw') \cdot n(y - pw) = z \cdot (x - pw') \\
m(x - pw')(x - pw) \cdot n(y - pw') = z' \cdot (x - pw)
\end{array} \right. 
\]

Subtracting the two equations, we get

\[ z(x - pw') - z'(x - pw) = mn(x - pw)(x - pw')(pw' - pw) \]

And we can derive a formula for the desired CDH:

\[ g^{mn} = \left( Z^{x - pw'}, Z^{(pw - x)} \right)^{(x - pw)(x - pw')(pw' - pw)} \]

This formula can be computed by the attacker, provided that \( x \neq pw \) and \( x \neq pw' \), which we know to be the case as otherwise the bad event would not have occurred due to the action of \( \text{bad}_7 \).

**Lemma 6.7.** For every algebraic attacker \( A \), there exists an attacker \( B \), such that

\[ \Pr[G_{10} \Rightarrow \text{bad}_3] \leq \text{Adv}^{G_{\text{SqDH}}}_{B}(\) \]

**Proof.** The reduction is given in Fig. 7. In the unlikely event that \( M = 1 \) there is nothing to do, as the problem is trivial to solve. We can also assume that the conditions checked to activate \( \text{bad}_7 \) all failed, as otherwise \( \text{bad}_6 \) would have been reset. This implies, in particular, that there are no entries in \( T_s \) such that the secret exponent \( x \) or \( y \) collides with the password \( pw \).

Given \( M \), the reduction sets \( N = M^\lambda \), for random non-zero \( \lambda \), which yields correctly distributed global parameters. Similarly to what happened in the proof of Lemma 6.6, we use the DDH oracle to ensure the simulation is perfect, so it remains to show that our reduction can compute \( \text{CDH}(M, M) \) whenever \( \text{bad}_6 \) occurs. There are two cases to consider, depending on whether we are dealing with a session accepted by a user or by a server.

**Sessions accepted in SendTerm.** We know that the following equation holds.

\[ Z = \text{CDH}(g^{aM^bN^c}, N^{pw}, g^{aM^bN^c}, N^{pw}) \]

This can be rewritten as:

\[ Z = \text{CDH}(M^{x - pw}, g^{aM^bN^c}, N^{pw}) \leftrightarrow \frac{Z}{M^{a(x - pw)}} = \text{CDH}(M^{x - pw}, M^{bN^c}) \]

Replacing \( N = M^\lambda \) and rearranging the exponents we get

\[ \frac{Z}{M^{a(x - pw)}} = \text{CDH}(M, M)^{(x - pw)(b + \lambda(c - pw))} \]

We can therefore recover the desired \( \text{SqDH} \) provided that \((x - pw)(b + \lambda(c - pw)) \neq 0\). So see that this can never happen, note \( b + \lambda(c - pw) \neq 0 \), as otherwise \( \text{bad}_{10} \) would have happened and this entry would not be in the list, and we also excluded \( x = pw \) above. This means that we can compute the \( \text{SqDH} \).

**Sessions accepted in SendResp.** In this case we know that the following equation holds.

\[ Z = \text{CDH}(g^{aM^bN^c}, N^{pw}, g^{aM^bN^c}, N^{pw}) \]

This can be rewritten as:

\[ Z = \text{CDH}(g^{aM^{b - pw}N^c}, N^{y - pw}) \]
Replacing \( N = M^λ \) and rearranging the exponents we get

\[
Z = \text{CDH}(g^a M^{b-pw+\lambda c}, M^{λ(y-pw)}) \iff \frac{Z}{M^{αλ(y-pw)}} = \text{CDH}(M, M^{(b-pw+λc)(y-pw)})
\]

We can therefore recover the desired SqDH provided that \((b - pw + \lambda c)(y - pw) \neq 0\). So see that this can never happen, note that we know that \(y - pw \neq 0\) and it must be the case that \(b - pw + \lambda c \neq 0\), as otherwise \(\text{bad}_0\) would have happened and this entry would not be in the list. This means that the SqDH can be computed.

\[ \square \]

**Lemma 6.8.** For every attacker \( A \), there exists an attacker \( B_s \), such that

\[
\text{Pr}[G_9 \Rightarrow \text{bad}_s^2] \leq 2q_s \cdot \text{Adv}_{B_s}^{G_{CDH}}
\]

**Proof.** The reduction is given in Fig. 7. Our adversary \( B_s \) guesses a query in the range \([0, q_s-1]\) and chooses a role uniformly at random to guess whether the challenge should be injected on the responder side or on the initiator side. The behavior conditioned on these guesses is shown in the figure as two different versions of \( B_s \), one for initiator side and another for responder side. Similarly to what happened in the proof of Lemma 6.6, we use the DDH oracle to ensure the simulation is perfect, and the correct result is produced as long as the guesses are correct. On the initiator side, the reduction is slightly more intricate as one needs to embed \( A_1 \) in all possible initiator sessions that might lead to the bad event. The bound on the theorem follows from the fact that the reduction only works if our attackers guesses are correct.

\[ \square \]

## 7 Security with Key Confirmation

The result we obtained in the previous section provides guarantees even if the derived secret keys are used before a key confirmation step. However, in most practical settings a key confirmation step exists and, in this case, a much better security bound can be proved. This is what we do in this section. We show the modified protocol in Fig. 2. Note that a one-time secure MAC suffices.

**Theorem 7.1.** SPAKE2 with key confirmation is tightly PFS-secure in the random-oracle model under the GCDH assumption. More precisely, for every attacker \( A \) against SPAKE2, there exist attackers \( B_6 \) and \( B_8 \) against the Gap Diffie-Hellman problem, attacker \( B_2 \) against the Gap Discrete Logarithm problem, attacker \( B_7 \) against the pseudorandomness of KDF and attacker \( B_8 \) against the unforgeability of the MAC scheme such that

\[
\text{Adv}_{A_{\text{spake}}}^{\text{GCDH}}(\cdot) \leq \frac{q_s}{|P|} + \text{Adv}_{B_6}^{G_{\text{GCDH}}}() + \text{Adv}_{B_7}^{\text{GDL}^\star}() + \text{Adv}_{B_8}^{\text{GDL}}() + q_s \left( \text{Adv}_{B_6}^{\text{1PRF}}() + \text{Adv}_{B_8}^{\text{1UF}}() \right) + \frac{(q_s + q_e)^2}{2q}
\]

**Proof.** Again we use a sequence of hops, but we explain here only the differences to the proofs in the previous section. We give the games in Figs. 8 to 10 in Appendix B.

The hops up to game \( G_4 \) are identical to those for the proofs in the previous section. In game \( G_4 \), as before, we exclude the possibility of the adversary breaking a session where it behaved passively but, in addition to EXECUTE queries, we handle also the sequences of SEND queries where the adversary establishes sessions with traces \((U, S, X^*, Y^*)\) where the messages were generated by the game. We give a modified reduction to deal with this case in Lemma 7.3 and note that here the gap oracle is not only used to detect the correct CDH answer, but to maintain consistency of all other sessions where the attacker is active.

We now carry on the the discussion of the following hops in detail from game \( G_6 \).

**GAME \( G_6 \).** This game introduces an independent random oracle for keys derived for (possibly aborted) fresh sessions. Flag \( \text{bad}_6 \) corresponds to the attacker observing an inconsistency in the hash queries for \( H \). As in the previous proof, we have

\[
\text{Adv}_{A_{\text{spake}}}^{\text{GCDH}}(\cdot) \leq \frac{(q_s + q_e)^2}{2q} + \text{Adv}_{B_6}^{G_{\text{GCDH}}}() + \text{Adv}_{B_8}^{G_{\text{GDL}}}() + \text{Pr}[G_7 \Rightarrow \text{bad}_6],
\]
<table>
<thead>
<tr>
<th>G_i</th>
<th>bad_i</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_2</td>
<td>bad_2</td>
<td>Accepted trace on responder side collides with any previously accepted trace or accepted trace on initiator side collides with trace of any previously accepted initiator session. Bounded by a statistical term.</td>
</tr>
<tr>
<td>G_4</td>
<td>bad_4</td>
<td>Random oracle query consistent with the trace accepted as a result of a query to Execute. Such an event would allow attacker to detect an inconsistency between the outputs of Test or Reveal and the random oracle. Bounded by a tight reduction to GCDH.</td>
</tr>
<tr>
<td>G_6</td>
<td>bad_6</td>
<td>Random oracle query consistent with the trace accepted as a result of a query to one of the Send oracles and declared fresh. Such an event would allow attacker to detect an inconsistency between the outputs of Test or Reveal and the random oracle. Adversary’s view is independent of b in this game, but bad_6 is hard to bound.</td>
</tr>
<tr>
<td>G_7</td>
<td>bad_7</td>
<td>The meaning is the same as in the previous game, but it now is only triggered in cases where the other bad event in this game does not occur. Either there is a call to the random oracle with $pw = x$ or $pw = y$, where $x$ or $y$ is one of the random values sampled in the game. Or a password was sampled by the game that also collides with one of these values. Bounded by a tight reduction to GDL.</td>
</tr>
<tr>
<td>G_8</td>
<td>bad_8</td>
<td>The meaning is the same as in the previous games, but it now is only triggered in cases where the other bad event in this game does not occur. At this point this event can only happen for random oracle queries placed before the associated password was sampled in the game. Furthermore, we know there can be at most $q_s$ entries, which results in a $q/</td>
</tr>
<tr>
<td>G_9</td>
<td>bad_9^1</td>
<td>Same meaning as in the previous game and same probability of occurrence. We fork here the proofs of Theorems 6.1 and 6.2. For the latter we immediately bound the event with a tight reduction to GPCCDH, which yields a loose reduction to GCDH via the splitting lemma of [PS00, AP05]. For Theorem 6.2 we delay bounding this event until $G_{10}$.</td>
</tr>
<tr>
<td>G_10</td>
<td>bad_10^2</td>
<td>Same meaning as in the previous game, but now only set when the other bad flag in this game is not set. Bound using a reduction to GSqDH assuming an algebraic adversary. The adversary committed to a representation of a group element in a Send query that predicts a password sampled later in the game. The probability of this event occurring can be easily bound as $q_s/</td>
</tr>
</tbody>
</table>

Table 1. Meaning of the bad events in the different games.
\[
\text{Client} \quad \text{Server}
\]
\[
pw \in \mathcal{P}, M, N \in G
\]
\[
\text{accept} \leftarrow F
\]
\[
x \leftarrow Z_q
\]
\[
X \leftarrow g^x
\]
\[
X^* \leftarrow X \cdot M^{pw}
\]
\[
\rightarrow \quad U, X^*
\]
\[
Y \leftarrow x \leftarrow Z_q
\]
\[
Y \leftarrow g^y
\]
\[
Y^* \leftarrow Y \cdot N^{pw}
\]
\[
\rightarrow \quad X \leftarrow X^* / M^{pw}
\]
\[
(U, X^*) \quad \text{and} \quad (K_a, K_e) \leftarrow H(U, S, X^*, Y^*, \text{pw}, X^y)
\]
\[
(K_{U,S}, K_{S,U}) \leftarrow \text{KDF}(K_a, \text{const})
\]
\[
\text{Abort if tag}_{S,U} \neq \text{MAC}(K_{S,U}, \text{sid})
\]
\[
\text{tag}_{S,U} \leftarrow \text{MAC}(K_{U,S}, \text{sid})
\]
\[
\text{tag}_{U,S} \leftarrow \text{MAC}(K_{U,S}, \text{sid}) \quad \rightarrow \text{Abort if tag}_{U,S} \neq \text{MAC}(K_{U,S}, \text{sid})
\]
\[
\text{accept} \leftarrow T
\]
\[
\text{accept} \leftarrow T
\]

\textbf{Fig. 2.} The SPAKE2 protocol [AP05] with key confirmation. Here, \text{const} is a fixed point in the domain of the KDF, \text{sid} = (U, S, X^*, Y^*) and \( K_e \) is the accepted session key.
where the term corresponding to $B_b$ highlights the fact that the reduction that deals with passive sessions is the one given in Lemma 7.3. Furthermore, as before, the adversary’s view in this game is independent of the secret bit $b$ and it only remains to bound the probability of $\text{bad}_6$ occurring. Let us take a closer look at game $G_6$ to confirm that this is the case.

First of all, let us confirm that all the session keys corresponding to fresh accepted sessions have been moved to an independent random oracle, whereas all revealable (accepted) non-fresh keys have not. Notice that random oracle queries are added to $T_s$ rather than $T$ in two occasions:

- When a $\text{Send}$ query starts a responder session for an uncorrupted password; this session may or may not turn out to be accepted and, if it is, it may or may not be fresh.
- When a $\text{Send}$ query is about to conclude a fresh initiator session (such an entry has not yet been previously added to $T_s$ since we never do this for sessions where the attacker is passive); this session may or may not be accepted due to MAC verification.

For the latter type of entries, if the MAC verification passes, then the session will be accepted as fresh, so we must ensure that such an entry is never removed from $T_s$ in this case. Removal could only occur in the $\text{Corrupt}$ oracle, but here the check performed guarantees that entries are only removed from $T_s$ when no accepted fresh session with that trace exists.

For the first type of entry, we must ensure that the entry is removed from $T_s$ whenever it might be accepted as unfresh. This is ensured by the corruption oracle: it removes any such entries upon corruption and reprograms the global random oracle with the corresponding key material when corruption occurs before any session with that trace is accepted.

Note further that the corrupt oracle also cleans up any entries in $T_s$ corresponding to traces of potentially fresh sessions that the attacker failed to conclude before corruption (e.g., by delivering an inconsistent MAC). This means that, after corruption of $\text{pwUS}$, the only entries that remain in $T_s$ corresponding to $(U,S)$ interactions must correspond to at least one fresh accepted session.

Now let us confirm that any inconsistencies detected by the adversary in its queries to the global random oracle (because it has observed in the game the output of the independent random oracle or some usage of it) are correctly signalled by activating the $\text{bad}_6$ flag.

The attacker may observe the action of the independent random oracle via the following oracle queries:

- $\text{Reveal}$ or $\text{Test}$ queries to accepted fresh sessions.
- MAC values computed using keys derived from entries in $T_s$ rather than the global random oracle.

For all entries added and never removed from $T_s$, the $\text{bad}_6$ flag is activated if a matching entry is ever queried to $H$ in the game. This covers the first class of observations. MAC checks and computations for such sessions are handled similarly.

The possibility remains, however, that the attacker might observe a MAC value related to an entry in $T_s$ that is later moved to $T$. If the query to $H$ that causes the inconsistency comes while the offending entry is in $T_s$, then $\text{bad}_6$ is activated as before. Otherwise, note that entries moved from $T_s$ to $T$ retain the key material, so subsequent queries to $H$ will actually be consistent.

**GAME $G_7$.** The intuition of this hop is identical to that for game $G_7$ in the proofs of the previous section: we change the way we generate messages in $\text{Send}$ queries and set a bad flag if, due to some unlikely event, one of the random exponents generated by the game to do this occurs as a password in the game. This event can be bounded by a reduction to GDL, as shown in Lemma 6.5, so we have:

$$\text{Adv}_{A}^{\text{SPAKE2}}() \leq \frac{(q_s + q_e)^2}{2q} + \text{Adv}_{B_1}^{\text{H-GCDH}}() + \text{Adv}_{B_2}^{\text{GDL}}() + \Pr[ G_7 \Rightarrow \text{bad}_6 ]$$

**GAME $G_8$.** In this game we reject any tags submitted by the attacker to a $\text{Send}$ oracle that would result in a new fresh session being accepted, when it has no information on the corresponding authentication key. More in detail:
If a fresh session is about to be accepted on the initiator side, and there does not exist a matching session on the responder side, then the attacker has never observed any computation based on the corresponding $T_s$ entry.

If a fresh session is about to be accepted on the responder side, and there does not exist a matching session on the initiator side, then the attacker may have observed a MAC computed with $K_{S,U}$, but it has no information about $K_{U,S}$.

Note that, by rejecting these sessions, the attacker can now only complete fresh sessions by behaving passively.\(^8\) We add a bad flag $\text{bad}_6$ to deal with cases where the tag would have been accepted in the previous game and, when this flag is activated, the potentially fresh session is forgotten and removed from the independent random oracle. This also means that, in this game, $\text{bad}_6$ can now only occur after corruption for sessions where the attacker behaved passively, which is why we can further restrict the setting of the bad fact to non-corrupted passwords. To see that this is indeed the case, observe that $\text{bad}_6$ happens while entries are in $T_s$, but in this game we remove from $T_s$ traces corresponding to active behaviour before the corresponding password is corrupted (as otherwise these sessions would not lead to fresh sessions anyway).

The probability of $\text{bad}_6$ occurring can be bounded using the PRF property of KDF and the standard (one-time) unforgeability of the MAC scheme. A formal proof is given in Lemma 7.4. Therefore we have

$$\Pr|G_7 \Rightarrow \text{bad}_6| \leq \Pr|G_8 \Rightarrow \text{bad}_6| + q_s \cdot \left(\text{Adv}^{\text{PRF}}_{B_7}() + \text{Adv}^{\text{UF}}_{B_8}()\right)$$

GAME $G_9$. In this game we introduce changes similar to those we introduced in $G_8$ of the proofs of Theorems 6.1 and 6.2 in moving all password-related checks to the end of the game. However, unlike in the previous section, we can now move all occurrences of $\text{bad}_6$ to the end, since we handled the problematic queries placed after corruption in the previous game hops. We move checks to the end by introducing a $T_{\text{bad}}$ list, and a final bad event $\text{bad}_9$ for executions where the attacker is able to construct two different consistent entries in this list for the same trace and different passwords.

Analysis of $\text{bad}_9$ and $\text{bad}_6$ in game $G_9$ are identical to the analyses of $\text{bad}_6$ and $\text{bad}_8$ in game $G_8$ of the proofs in the previous section: $\text{bad}_6$ is the probability of passwords guessing, whereas $\text{bad}_9$ can be reduced to GCDH as proved in Lemma 6.6. We therefore have

$$\Pr|G_8 \Rightarrow \text{bad}_6| \leq \Pr|G_9 \Rightarrow \text{bad}_6| + \Pr|G_9 \Rightarrow \text{bad}_9| \leq \frac{q_e}{|P|} + \text{Adv}^{\text{GCDH}}_{B_3}(\ )$$

This concludes the proof.

The above proof also establishes that SPAKE2 with key confirmation provides explicit authentication defined formally as:

- if a client $U$ accepts a session with server $S$, then $S$ has a pending unique session where it has derived the same key and session identifier ($U,S,X^*,Y^*$);
- if a server $S$ accepts a session with client $U$, then $U$ has already accepted a unique session with the same key and session identifier ($U,S,X^*,Y^*$).

We state this as the following corollary, which shows that impersonation can only be achieved with negligible probability beyond guessing the password.

**Corollary 7.2.** SPAKE2 with key confirmation provides explicit authentication.

**Proof.** The proof follows the same sequence of hops as that of Theorem 7.1. We can observe that in the final game a session is accepted only if the adversary behaved passively, which means that explicit

\(^8\)Forging MACs that result the same trace being accepted as in passive behaviour is not considered an active attack. This maps to the standard existential unforgeability notion.
Lemma 7.3. For every attacker $A$, there exists an attacker $B_6$, such that
\[
\Pr[ G_4 \Rightarrow \text{bad}_4] \leq \text{Adv}^{U\text{-GCDH}}_{\text{GCDH}}()
\]

Proof. Let us consider the GCDH attacker $B_6$ in Fig. 11. It gets a generalized challenge of the form $(X_1, \ldots, X_{q_e}, Y_1, \ldots, Y_{q_e})$ and finds $\text{CDH}(X_i, Y_j)$, for some $1 \leq i, j \leq q_e$. This problem is tightly equivalent to $U$-GCDH by random self-reducibility. The attacker embeds $X_{k_0}$ and $Y_{k_0}$ in all game-generated $X^*$ and $Y^*$, which means that it cannot compute the CDH for any session. However, it can use the DDH oracle to check when $H$ queries for a given group element $Z$ match traces where $\text{CDH}(X, Y) = Z$. Additionally, the DDH oracle is used to check for the bad event, in which case a solution to the GCDH problem was found.

Lemma 7.4. For every attacker $A$, there exist attackers $B_7$ and $B_8$, such that
\[
\Pr[ G_8 \Rightarrow \text{bad}_8] \leq q_s \cdot \left( \text{Adv}^{\text{IPRF}}_{B_7}() + \text{Adv}^{\text{UF}}_{B_8}() \right)
\]

Proof. Consider the modified version of game $G_8$ we present in Figure 11 (game $G^{\nu}_8$) and let us consider for now the behaviour of this game when $c = F$. In this case, the only difference between the two games is that game $G^{\nu}_8$ only sets bad$_8$ in one special $\ell$-th query to SEND, while the original version of $G_8$ might set the flag in many SEND queries. However, this special query is chosen uniformly at random and is outside of the adversary’s view, so it must be the case that, conditioning on the probability that the randomly chosen $\ell$ correctly guesses the query that causes the first occurrence of bad$_8$ in game $G_8$, the probability of bad$_8$ occurring in the two games is the same. This implies,
\[
\Pr[ G_8 \Rightarrow \text{bad}_8] = \Pr[ G_8^\nu \Rightarrow \text{bad}_8 | \ell \text{ is correct}] \leq q_s \Pr[ G_8^\nu \Rightarrow \text{bad}_8]
\]

We can now focus on game $G_8^\nu$. Observe that parameter $c = T$ replaces the keys produced by the KDF with random ones for the $\ell$-th query. This means that it is easy to construct an attacker $B$ such that
\[
\text{Adv}^{\text{IPRF}}_{B_7}() = | \Pr[ G_8^\nu \Rightarrow \text{bad}_8] - \Pr[ G_8^\nu \Rightarrow \text{bad}_8] |
\]

We omit the details of this reduction. At this point, it suffices to bound the probability of bad$_8$ in game $G_8$. We do this via a direct reduction to the one-time unforgeability property of the MAC scheme. This is shown in Figure 11 as attacker $B_8$. Again the reduction is simple: for the critical $\ell$-th query, the attacker fixes the authentication key that causes bad$_8$ implicitly as the external MAC key for which it is trying to forge. When bad$_8$ occurs, it returns the offending trace and authenticator as a forgery. \qed
8 Implications

Nothing-up-your-sleeve global parameters. Our security proof applies without change to any distribution of the global parameters \((M, N)\) under which computing the CDH between the two with the help of a DDH oracle is assumed to be hard. This means that simply sampling \((M, N)\) uniformly at random is a good choice when the GCDH assumption holds, and so is choosing \(M = H(K)\) and \(N = H(K')\) when \(H\) is modeled as a random oracle. Here the choice of \(K \neq K'\) means one is relying on the standard GCDH assumption, whereas \(K = K'\) leads to \(M = N\), which means relying on the squared GCDH problem.

Removing the global parameters. Our proof extends to a variant of SPAKE2 where \((M, N)\) are not generated in a global setup. Instead one can use \((M, N) = H(U, S)\) and \(N = H(S, U)\), where different values of these parameters are precomputed for each user-server pair. The online efficiency of the protocol is not affected. Modeling \(H\) as a random oracle, this means that each \((U, S)\) pair now poses an independent GCDH challenge to the active attacker. Our proof can be easily modified to cover this case with the same bound by observing that bad events that involve multiple sessions are always defined between sessions established for the same user-server \((U, S)\).

Reusing passwords. Our model assumes that user passwords are sampled independently for each pair \((U, S)\). The simple case of password reuse, where passwords are either repeats or they are sampled independently can be easily addressed by extending the corrupt oracle to exclude additional trivial attacks: it should declare as corrupt all \((U', S')\) pairs that use the same password. The same proof applies and the bound is not affected in this case, as the entropy of non-repeat passwords is assumed to be unaffected.

The treatment of more complex distributions follows in the same lines as described in [KOY09, Section 2.2.1].

Acknowledgments. We would like to thank David Pointcheval for helpful discussions regarding the proof of Lemma 6.6 and Doreen Riepel for discussions that helped us clarify the proofs of Lemmas 6.5 and 6.6. This work was supported in part by the ERC Project aSCEND (H2020 639554) and by the French ANR ALAMBIC Project (ANR-16-CE39-0006). Manuel Barbosa was supported by grant SFRH/BSAB/143018/2018 awarded by FCT, Portugal.

References


A Games and adversaries for the proof of Theorems 6.1 and 6.2
Fig. 3. Security proof for SPAKE2. Games 1 to 4.
Fig. 4. Security proof for SPAKE2. Games 5 to 7.
\textbf{Fig. 5.} Security proof for SPAKE2. Games 8 to 10.
\textbf{Adversary B1}  
\begin{align*}
\text{proc Initiate}(X_1, \ldots, X_{24}, Y_1, \ldots, Y_{24}) &= 0 \text{ s.t. } T \in \{ \cdots, 1 \}. \quad \text{Proc. 1.} \\
\text{Return} (M, N)
\end{align*}

\text{proc Simulate}(i, S) 
\begin{align*}
\text{If } \exists j \in \{ \cdots, 1 \} \text{ s.t. } S_{jQ} = \phi \rightarrow
\begin{align*}
\text{Return } (M, N) \\
\text{proc Simulate}(S, U, X^*)
\end{align*}
\end{align*}

\text{proc Finalise}(k^*) 
\text{Stop.}

\textbf{Adversary B2}  
\begin{align*}
\text{proc Initiate}(A) 
\begin{align*}
b &\leftarrow \{ 0, 1 \}, \quad C \leftarrow \{ \cdots, 1 \} \\
\text{Return} (M, N)
\end{align*}
\end{align*}

\text{proc Simulate}(U, Z) 
\begin{align*}
x &\leftarrow \{ 1, 2 \} \\
T \leftarrow \{ \cdots, 1 \} \\
\text{Return} (T, U, X^*)
\end{align*}

\text{proc Finalise}(k^*) 
\text{Stop.}

\textbf{Adversary B3}  
\begin{align*}
\text{proc Initiate}(M, N) 
\begin{align*}
b \leftarrow \{ T \}, \quad C \leftarrow \{ \cdots, 1 \} \\
\text{Return} (M, N)
\end{align*}
\end{align*}

\text{proc Simulate}(S, U, X^*) 
\begin{align*}
y &\leftarrow \{ 1, 2 \} \\
\text{Return} (S, Y^*)
\end{align*}

\text{proc Finalise}(k^*) 
\text{Stop.}

\textbf{Fig. 6.} Security proof for SPAKE2. GCDH and GDL attackers.
B  Games and adversary for the proof of Theorem 7.1
Fig. 7. Security proof for SPAKE2. Algebraic reduction to GCDH and alternative reduction to GPCCDH.
proof Initialize()

G1, G2

For U ∈ Ut, S ∈ S do pskU ← F; SendInit(U, S

SendInit(U, S, KU, S, KS, U)

if πi ∉ S, U, X∗ then πi ← S, U, X∗, Y∗ = ππi, tr

Finalize(U, S)

proof Initialize()

G3

for U ∈ Ut, S ∈ S do pskU ← F

SendInit(U, S)

SendTermInit(U, S, πi)

if πj ∉ S, U then πj ← S, U, X∗, Y∗ = ππj, tr

Finalize(U, S)

proof SendResp(S, i, U, X∗, Y∗)

if b ∈ S ∪ Ut, (U, S, X∗, Y∗) then ππππi

Return X ← X∗ / kPort

proof SendResp(S, i, X∗, Y∗, tag)

if πj ∉ (U, S, X∗, Y∗) then πj ← (x, U, S, X∗, Y∗) = ππj, tr

Return X ← X∗ / kPort

proof SendInit(U, i, S, KU, S, KS, U)

if tag MAC(Kj, gKj, (U, S, X∗, Y∗)) then ππππj

Return tag

proof SendTermInit(U, i, S, X∗, Y∗, tag)

if πj ∉ (y, (U, S, X∗, Y∗), (Kj, KS, U)) then πj ← (y, (U, S, X∗, Y∗), (Kj, KS, U))

Return T

proof Execute(U, s, i, j)

if πj ∉ (y, (U, S, X∗, Y∗), (Kj, KS, U)) then πj ← (y, (U, S, X∗, Y∗), (Kj, KS, U))

Return T

proof H(U, S, X∗, Y∗, tag)

proof Correct(U, S)

proof Review(P, i)

proof Test(P, i)

proof Finalize(f)

Return S ← T

Perfect Forward Security of SPAKE2 27

Fig. 8. Security proof for SPAKE2 with confirmation. Games 1 to 4.
Fig. 9. Security proof for SPAKE2 with confirmation. Games 5 to 7.
Fig. 10. Security proof for SPAKE2 with confirmation. Games 8 to 9.
If $\pi_i \not\in S \cup U$,\ Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.

Return $\bot$.