Perfect Forward Security of SPAKE2

Michel Abdalla[1,2] and Manuel Barbosa[3]

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

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

[4] https://irtf.org/cfrg

authenticated key establishment in IETF protocols. Currently, 4 candidates are under consideration by the CRFG under the balanced PAKE category: SPEKE [Jab97,Mac01,HS14], SPAKE2 [AP05,LK19], J-PAKE [HR10,ABM15], and CPace [HL18,HL19].

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 [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. 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 in the HMQV [Kra05] paper since the SPAKE2 protocol assumes the pre-existence of secure shared state between parties which is the password itself.

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.

2 Preliminaries

Notation. We write $x \leftarrow y$ for the action of assigning the value y to the variable x. We write $x_1, \ldots, x_n \leftarrow\!\!\$\ \mathsf{X}$ 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 $\{\,\}$ 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 queries. A game is run with an adversary $\mathcal{A}$ as follows. First Initialize runs and its outputs are passed to $\mathcal{A}$. Then $\mathcal{A}$ runs and its oracle queries are answered by the procedures of the game. When $\mathcal{A}$ terminates, its output is passed to 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.

4 Security Model

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

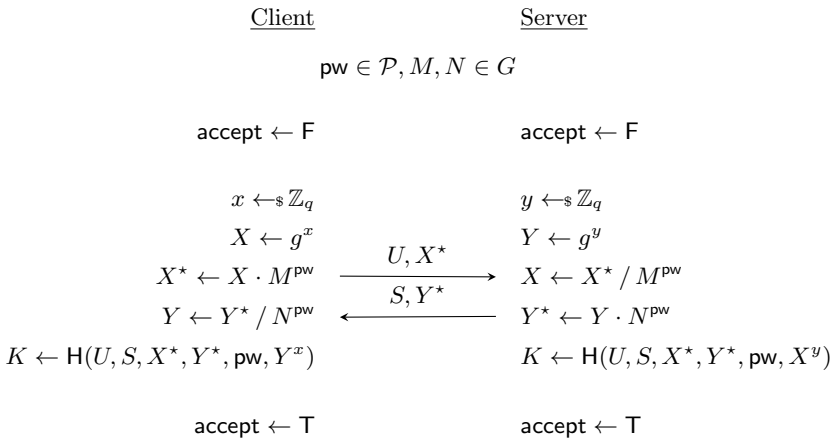

Fig. 1. The SPAKE2 protocol [AP05].

The security game already instantiated with SPAKE2 can be seen as G_1 in Fig. 2. The name spaces for servers $\mathcal{S}$ and users $\mathcal{U}$ 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 π_P^i is a tuple of the form $(\mathsf{e}, \mathsf{tr}, K, \mathsf{ac})$ where*

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

We will use $\pi_P^i.\mathsf{e}$, $\pi_P^i.\mathsf{tr}$, $\pi_P^i.K$, $\pi_P^i.\mathsf{ac}$ to denote the individual components of the state.

Definition 4.2 (Partnering). *A server instance π_S^i and a user instance π_U^j are partnered iff*

$$\pi_S^i.\mathsf{ac} = \mathsf{T} \wedge \pi_U^j.\mathsf{ac} = \mathsf{T} \wedge \pi_S^i.\mathsf{tr} = \pi_U^j.\mathsf{tr}$$

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 $\mathsf{Fresh}(P, i)$ if and only if all the following conditions hold:*

1. *the instance accepted;*
2. *the instance was not queried to* Test *or* Reveal *before;*
3. *At least one of the following conditions holds:*
   (a) *The instance accepted during a query to* Execute*;*
   (b) *There exists more than one partner instance;*
   (c) *No partner instance exists and* 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. Note that the CDH problem has been shown to be equivalent to the standard discrete logarithm problem (DL) in the Algebraic and Generic group models [Los19] so, our proof implies reductions to the DL problem in these idealized models of computation.

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. Active attackers that may try to take advantage of leaked passwords must compute the CDH of the global parameters M and N with the help of a DDH oracle. Indeed, 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.

Definition 5.1 (Gap Computational Diffie-Hellman ($\mathcal{D}$-GCDH)). *The $\mathcal{D}$-GCDH assumption states that, for any ppt adversary $\mathcal{A}$, the following probability is small.*

$$\mathsf{Adv}^{\mathsf{GCDH}} := \Pr\left[Z = g^{xy} \ : \ Z \leftarrow_{\$} \mathcal{A}^{\mathsf{DDH}(\cdot,\cdot,\cdot)}(g^x, g^y);\ (x,y) \leftarrow_{\$} \mathcal{D} \right]$$

The standard GCDH assumption is a particular case when $\mathcal{D}$ is the uniform distribution over $\mathbb{Z}_q^2$. We denote this by $\mathcal{U}$-GCDH. The Gap Squared Diffie-Hellman assumption is another particular case, when we restrict the previous case to $x = y$. For simplicity, in our analysis, we will restrict our attention to distributions $\mathcal{D}^\star$ where we exclude the possibility that $x = 0$ or $y = 0$. This means that the bound for our proof includes an extra statistical term of the form $(q_s + q_e)/q$ when we remove this restriction.

Finally, we will also rely on the weaker Gap Discrete Logarithm assumption.

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

$$\mathsf{Adv}^{\mathsf{GDL}} := \Pr\left[x' = x \ : \ x' \leftarrow_{\$} \mathcal{A}^{\mathsf{DDH}(\cdot,\cdot,\cdot)}(g^x);\ x \leftarrow_{\$} \mathbb{Z}_q \right]$$

6 Perfect Forward Security Proof

Theorem 6.1. *SPAKE2 is tightly PFS-secure in the random-oracle model under the Gap 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*

$$\mathsf{Adv}^{\mathrm{SPAKE2}}_{\mathcal{A}}() \leq \frac{q_s}{|\mathcal{P}|} + \mathsf{Adv}^{\mathcal{U}\text{-}\mathsf{GCDH}}_{\mathcal{B}_1}() + \mathsf{Adv}^{\mathsf{GDL}}_{\mathcal{B}_2}() + \mathsf{Adv}^{\mathcal{D}^\star\text{-}\mathsf{GCDH}}_{\mathcal{B}_3}() + \frac{(q_s + q_e)^2}{2q}$$

Proof. We prove the security of SPAKE2 using a sequence of games shown in Fig. 2 and Fig. 3.

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

$$\mathsf{Adv}^{\mathrm{SPAKE2}}_{\mathcal{A}}() = |\Pr[\mathsf{G}_1 \Rightarrow \mathsf{T}] - 1/2|$$

Game G_2. From game G_1 to game G_2 we introduce a bad flag bad_2. The bad_2 flag is set whenever

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

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 Execute and q_s queries to the Send oracles, we have for large q denoting the order of the group:

$$|\Pr[\mathsf{G}_1 \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_2 \Rightarrow \mathsf{T}]| \leq \frac{(q_e + q_s)^2}{2q}$$

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

$$\Pr[\mathsf{G}_3 \Rightarrow \mathsf{T}] = \Pr[\mathsf{G}_2 \Rightarrow \mathsf{T}]$$

Game G_4. In this game we make the keys of sessions accepted as a result of calls to 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 $(U, S, X^\star, Y^\star)$. Note that, due to the guarantee of session uniqueness on server and client side, 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 Execute collides with a previous call to H, or because a new call to H collides with a previously accepted session in Execute. The games are identical until bad occurs, so we have:

$$|\Pr[\mathsf{G}_3 \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_4 \Rightarrow \mathsf{T}]| \leq \Pr[\mathsf{G}_4 \Rightarrow \mathsf{bad}_4]$$

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

$$\Pr[\mathsf{G}_4 \Rightarrow \mathsf{bad}_4] \leq \mathsf{Adv}^{\mathcal{U}\text{-}\mathsf{GCDH}}_{\mathcal{B}_1}()$$

The reduction to Gap CDH is given in Lemma 6.2.

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 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.

Game G_6. In this game we change the way in which we generate keys for 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:

$$|\Pr[\mathsf{G}_5 \Rightarrow \mathsf{T}] - \Pr[\mathsf{G}_6 \Rightarrow \mathsf{T}]| \leq \Pr[\mathsf{G}_6 \Rightarrow \mathsf{bad}_6]$$

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:

$$\mathsf{Adv}^{\mathrm{SPAKE2}}_{\mathcal{A}}() \leq \frac{(q_s+q_e)^2}{2q} + \mathsf{Adv}^{\mathcal{U}\text{-}\mathsf{GCDH}}_{\mathcal{B}_1}() + \Pr[\,\mathsf{G}_6 \Rightarrow \mathsf{bad}_6\,]$$

To bound the probability of bad_6 we perform a final hop.

Game G_7. We perform a global change in the game that aims to ensure that the passwords for fresh sessions are only generated after the sessions are accepted. We first modify the way in which $X^\star$ and $Y^\star$ are generated, removing the component that depends on the password. The distribution of the values does not change, 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 do this change, honest passwords are now only used to check for the bad event bad_6, so we remove the check that passwords are consistent and delay this check until the end of the game: we store all problematic H tuples in a list.

We can now move password generation to occur after fresh session acceptance: for corrupt queries we generate the password at the time of corruption; all other passwords are generated at the end of the game. Finally, we check if bad_6 might have occurred by going through the log of problematic entries, to complete the $\mathsf{pw} = \mathsf{pw}_{\mathrm{US}}$ test a posteriori. The probability of bad_6 occurring would be exactly the same, but we would not be able to bound it since the size of the problematic list could be as large as q_h. For this reason, we add two additional bad flags bad^1_7 and bad^2_7. We check for these bad events if in the log of problematic H calls there is an entry in T_s that is consistent with two entries in H with different passwords; in this case we no longer set bad_6. We set bad^1_7 if one of these passwords collides with the hidden x or y exponent associated with that session trace, and bad^2_7 otherwise. The probability of bad^1_7 occurring in G_7 can be tightly reduced to the GDL problem. The probability of bad^2_7 occurring in G_7 can be tightly reduced to Gap CDH, where we use here the generalized version for any parameter distribution $\mathcal{D}$ where the CDH problem is hard to compute with the help of a DDH oracle.[5] Putting these results together, we obtain:

$$\begin{aligned}\Pr[\,\mathsf{G}_6 \Rightarrow \mathsf{bad}_6\,] &\leq \Pr[\,\mathsf{G}_7 \Rightarrow \mathsf{bad}_6\,] + \Pr[\,\mathsf{G}_7 \Rightarrow \mathsf{bad}^1_7\,] + \Pr[\,\mathsf{G}_7 \Rightarrow \mathsf{bad}^2_7\,] \\ &\leq \Pr[\,\mathsf{G}_7 \Rightarrow \mathsf{bad}_6\,] + \mathsf{Adv}^{\mathsf{GDL}}_{\mathcal{B}_2}() + \mathsf{Adv}^{\mathcal{D}^\star\text{-}\mathsf{GCDH}}_{\mathcal{B}_3}()\end{aligned}$$

The reduction to GDL is given in Lemma 6.3. The reduction to Gap CDH is given in Lemma 6.4. To conclude the proof, we observe that the probability of bad_6 happening in game G_7 is now easy to bound. Indeed, the size of the log in which bad_6 is checked has size at most q_s, and all entries are added to this set before the corresponding password is sampled. We therefore have

$$\Pr[\,\mathsf{G}_6 \Rightarrow \mathsf{bad}_7\,] \leq \frac{q_s}{|\mathcal{P}|}$$

This completes the proof, as we can derive

$$\mathsf{Adv}^{\mathrm{SPAKE2}}_{\mathcal{A}}() \leq \frac{q_s}{|\mathcal{P}|} + \mathsf{Adv}^{\mathcal{U}\text{-}\mathsf{GCDH}}_{\mathcal{B}_1}() + \mathsf{Adv}^{\mathsf{GDL}}_{\mathcal{B}_2}() + \mathsf{Adv}^{\mathcal{D}^\star\text{-}\mathsf{GCDH}}_{\mathcal{B}_3}() + \frac{(q_s+q_e)^2}{2q}$$

□

Lemma 6.2. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_1$, such that*

$$\Pr[\,\mathsf{G}_4 \Rightarrow \mathsf{bad}_4\,] \leq \mathsf{Adv}^{\mathcal{U}\text{-}\mathsf{GCDH}}_{\mathcal{B}_1}()$$

Proof. Let us consider a GDL attacker $\mathcal{B}_1$ in Fig. 4. It gets a generalized challenge of the form $(X_1, \ldots, X_{q_e}, Y_1, \ldots, Y_{q_e})$ and finds $\mathsf{CDH}(X_i, Y_j)$, for some $1 \leq i, j \leq q_e$. This problem is tightly

[5] Note that this does not weaken our result, as one can take $\mathcal{D}$ 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.

equivalent to $\mathcal{U}$-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 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.[6] □

Lemma 6.3. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_2$, such that*

$$\Pr[\,\mathsf{G}_7 \Rightarrow \mathsf{bad}_7^1\,] \leq \mathsf{Adv}^{\mathsf{GDL}}_{\mathcal{B}_2}(\,)$$

Proof. Let us consider a GDL attacker $\mathcal{B}_2$ in Fig. 4. It gets a challenge of the form A and finds $\mathsf{DL}(A)$. The attacker uses A in SEND queries to compute $X^\star = A \cdot g^x$ and $Y^\star = A \cdot g^y$. Note that the distribution of these values is correct. 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_7^1 and if offending entries exist, it recovers the discrete logarithm of A offset by a known quantity. □

Lemma 6.4. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_3$, such that*

$$\Pr[\,\mathsf{G}_7 \Rightarrow \mathsf{bad}_7^2\,] \leq \mathsf{Adv}^{\mathcal{D}^\star\text{-}\mathsf{GCDH}}_{\mathcal{B}_3}(\,)$$

Proof. Let us consider a $\mathcal{D}^\star$-GCDH attacker $\mathcal{B}_3$ in Fig. 4. It gets a challenge of the form (M, N) and finds $\mathsf{CDH}(M, N)$. The attacker uses (M, N) as the variables of the same name in game G_7, but it further uses these values in SEND queries to compute $X^\star = M^x$ and $Y^\star = N^y$. 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_7^2 and if offending entries exist, it recovers $(X^\star, Y^\star, \mathsf{pw}, Z)$ and $(X^\star, Y^\star, \mathsf{pw}', Z')$ such that $\mathsf{pw} \neq \mathsf{pw}'$ and the following holds for known x or known y, and known pw and pw':

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

Letting $Z = g^z$, $Z' = g^{z'}$, we can rewrite these equations as

$$\begin{cases} m(x - \mathsf{pw}) \cdot (ny - n\mathsf{pw}) = z \\ m(x - \mathsf{pw}') \cdot (ny - n\mathsf{pw}') = z' \end{cases}$$

Let us assume that the attacker knows x (the other case is symmetric). We scale the equations and rearrange the formula as follows:

$$\begin{cases} m(x - \mathsf{pw})(x - \mathsf{pw}') \cdot (ny - n\mathsf{pw}) \cdot x\mathsf{pw}' = z \cdot (x - \mathsf{pw}') \\ m(x - \mathsf{pw}')(x - \mathsf{pw}) \cdot (ny - n\mathsf{pw}') \cdot (x - \mathsf{pw}) = z' \cdot (x - \mathsf{pw}) \end{cases}$$

Subtracting the two equations, we get

$$z(x - \mathsf{pw}') - z'(x - \mathsf{pw}) = mn(x - \mathsf{pw})(x - \mathsf{pw}')(\mathsf{pw}' - \mathsf{pw})$$

And we can derive a formula for the desired CDH:

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

This formula can be computed by the attacker, provided that $x \neq \mathsf{pw}$ and $x \neq \mathsf{pw}'$, which we know to be the case as otherwise the bad event would not have occurred. □

[6] 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.

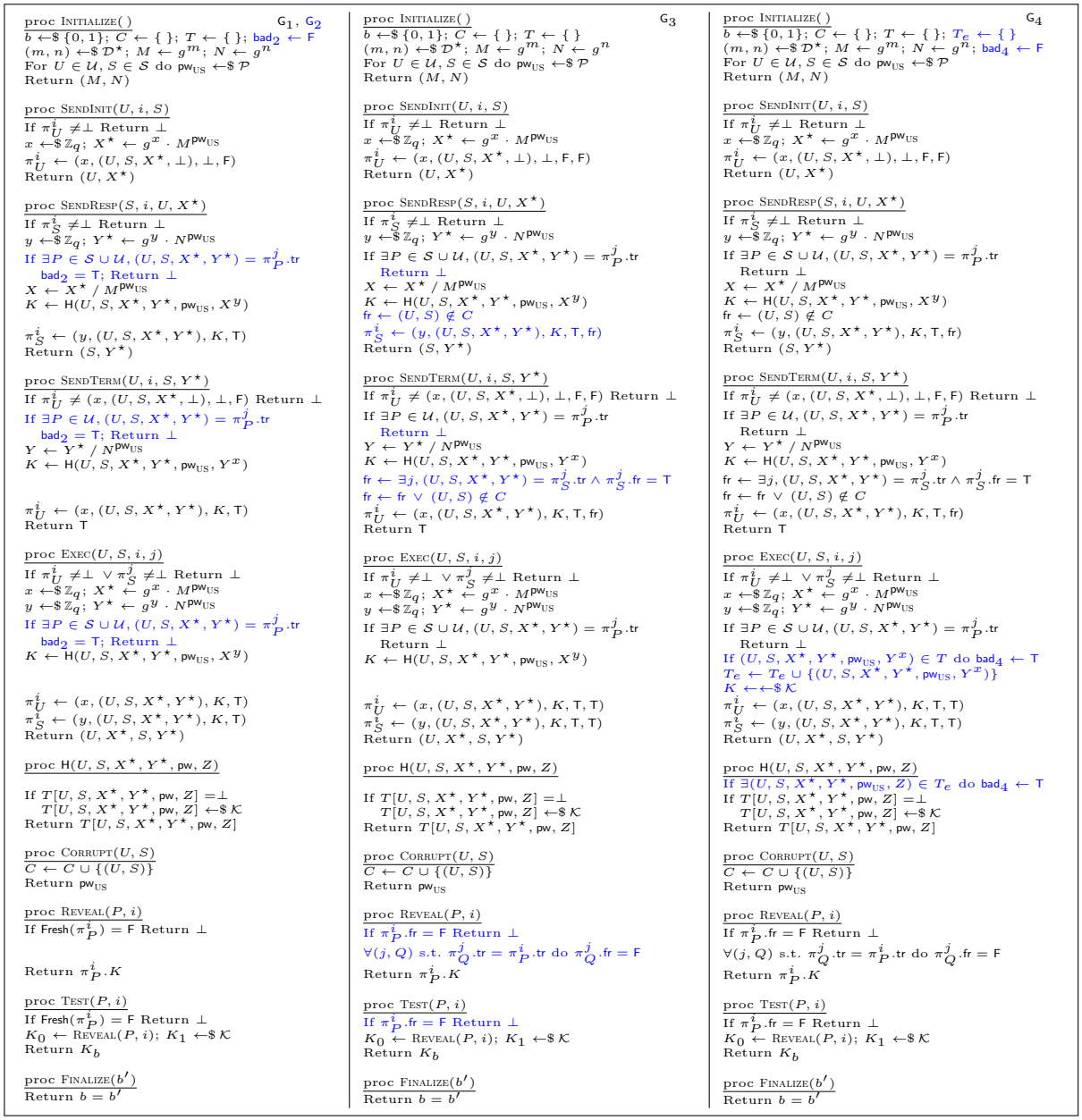

Fig. 2. Security proof for SPAKE2. Games 1 to 4.

Fig. 3. Security proof for SPAKE2. Games 5 to 7.

Adversary $\mathcal{B}_1$

proc Initialize($X_1, \ldots, X_{q_e}, Y_1, \ldots, Y_{q_e}$)
$b \leftarrow\$ \{0,1\}$; $C \leftarrow \{\}$; $T \leftarrow \{\}$; $T_e \leftarrow \{\}$; $k \leftarrow 0$
$(m, n) \leftarrow\$ \mathcal{D}^\star$; $M \leftarrow g^m$; $N \leftarrow g^n$
For $U \in \mathcal{U}, S \in \mathcal{S}$ do $\mathsf{pw}_{US} \leftarrow\$ \mathcal{P}$
Return (M, N)

proc SendInit(U, i, S)
If $\pi^i_U \neq \perp$ Return $\perp$
$x \leftarrow\$ \mathbb{Z}_q$; $X^\star \leftarrow g^x \cdot M^{\mathsf{pw}_{US}}$
$\pi^i_U \leftarrow (x, (U, S, X^\star, \perp), \perp, \mathsf{F}, \mathsf{F})$
Return $(U, X^\star)$

proc SendResp($S, i, U, X^\star$)
If $\pi^i_S \neq \perp$ Return $\perp$
$y \leftarrow\$ \mathbb{Z}_q$; $Y^\star \leftarrow g^y \cdot N^{\mathsf{pw}_{US}}$
If $\exists P \in \mathcal{S} \cup \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
$X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$
$K \leftarrow \mathsf{H}(U, S, X^\star, Y^\star, \mathsf{pw}_{US}, X^y)$
$\mathsf{fr} \leftarrow (U, S) \notin C$
$\pi^i_S \leftarrow (y, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{fr})$
Return $(S, Y^\star)$

proc SendTerm($U, i, S, Y^\star$)
If $\pi^i_U \neq (x, (U, S, X^\star, \perp), \perp, \mathsf{F}, \mathsf{F})$ Return $\perp$
If $\exists P \in \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
$Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
$K \leftarrow \mathsf{H}(U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Y^x)$
$\mathsf{fr} \leftarrow \exists j, (U, S, X^\star, Y^\star) = \pi^j_S.\mathsf{tr} \wedge \pi^j_S.\mathsf{fr} = \mathsf{T}$
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (U, S) \notin C$
$\pi^i_U \leftarrow (x, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{fr})$
Return T

proc Exec(U, S, i, j)
If $\pi^i_U \neq \perp \vee \pi^j_S \neq \perp$ Return $\perp$
$k \leftarrow k + 1$
$X^\star \leftarrow X_k \cdot M^{\mathsf{pw}_{US}}$
$Y^\star \leftarrow Y_k \cdot N^{\mathsf{pw}_{US}}$
If $\exists P \in \mathcal{S} \cup \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
If $(U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$
  If $\mathsf{DDH}(X_k, Y_k, Z) = \mathsf{T}$
    call Terminate(Z); stop
$T_e \leftarrow T_e \cup \{(U, S, X^\star, Y^\star, \mathsf{pw}_{US}, k)\}$
$K \leftarrow\$ \mathcal{K}$
$\pi^i_U \leftarrow (x, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{T})$
$\pi^j_S \leftarrow (y, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{T})$
Return $(U, X^\star, S, Y^\star)$

proc H($U, S, X^\star, Y^\star, \mathsf{pw}, Z$)
If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, k) \in T_e$
  If $\mathsf{DDH}(X_k, Y_k, Z) = \mathsf{T}$
    call Terminate(Z); stop
If $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z] = \perp$
  $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z] \leftarrow\$ \mathcal{K}$
Return $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z]$

proc Corrupt(U, S)
$C \leftarrow C \cup \{(U, S)\}$
Return pw_{US}

proc Reveal(P, i)
If $\pi^i_P.\mathsf{fr} = \mathsf{F}$ Return $\perp$
$\forall (j, Q)$ s.t. $\pi^j_Q.\mathsf{tr} = \pi^i_P.\mathsf{tr}$ do $\pi^j_Q.\mathsf{fr} = \mathsf{F}$
Return $\pi^i_P.K$

proc Test(P, i)
If $\pi^i_P.\mathsf{fr} = \mathsf{F}$ Return $\perp$
$K_0 \leftarrow$ Reveal(P, i); $K_1 \leftarrow\$ \mathcal{K}$
Return K_b

proc Finalize(b')
Abort.

Adversary $\mathcal{B}_2$

proc Initialize(A)
$b \leftarrow\$ \{0,1\}$; $C \leftarrow \{\}$; $T \leftarrow \{\}$; $T_s \leftarrow \{\}$; $T_{\mathsf{bad}} \leftarrow \{\}$
$(m, n) \leftarrow\$ \mathcal{D}^\star$; $M \leftarrow g^m$; $N \leftarrow g^n$
Return (M, N)

proc SendInit(U, i, S)
If $\pi^i_U \neq \perp$ Return $\perp$
$x \leftarrow\$ \mathbb{Z}_q$; $X^\star \leftarrow A \cdot g^x$
$\pi^i_U \leftarrow (x, (U, S, X^\star, \perp), \perp, \mathsf{F}, \mathsf{F})$
Return $(U, X^\star)$

proc SendResp($S, i, U, X^\star$)
If $\pi^i_S \neq \perp$ Return $\perp$
$y \leftarrow\$ \mathbb{Z}_q$; $Y^\star \leftarrow A \cdot g^y$
If $\exists P \in \mathcal{S} \cup \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
If $(U, S) \in C$
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $K \leftarrow T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z]$
  Else $K \leftarrow\$ \mathcal{K}$; $T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, (X, Y)] \leftarrow K$
Else
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{U, S, X^\star, Y^\star, \mathsf{pw}, Z\}$
  $K \leftarrow\$ \mathcal{K}$
  $T_s[U, S, X^\star, Y^\star] \leftarrow (S, y, K)$
$\mathsf{fr} \leftarrow (U, S) \notin C$
$\pi^i_S \leftarrow (y, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{fr})$
Return $(S, Y^\star)$

proc SendTerm($U, i, S, Y^\star$)
If $\pi^i_U \neq (x, (U, S, X^\star, \perp), \perp, \mathsf{F}, \mathsf{F})$ Return $\perp$
If $\exists P \in \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
If $(U, S) \in C \wedge (U, S, X^\star, Y^\star) \notin T_s$
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $K \leftarrow T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z]$
  Else $K \leftarrow\$ \mathcal{K}$; $T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, (X, Y)] \leftarrow K$
Else
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{U, S, X^\star, Y^\star, \mathsf{pw}, Z\}$
  If $T_s[U, S, X^\star, Y^\star] \neq (S, y, K)$
    $K \leftarrow\$ \mathcal{K}$
    $T_s[U, S, X^\star, Y^\star] \leftarrow (U, x, K)$
$\mathsf{fr} \leftarrow \exists j, (U, S, X^\star, Y^\star) = \pi^j_S.\mathsf{tr} \wedge \pi^j_S.\mathsf{fr} = \mathsf{T}$
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (U, S) \notin C$
$\pi^i_U \leftarrow (x, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{fr})$
Return T

proc Exec(U, S, i, j)
If $\pi^i_U \neq \perp \vee \pi^j_S \neq \perp$ Return $\perp$
$x \leftarrow\$ \mathbb{Z}_q$; $X^\star \leftarrow g^x$
$y \leftarrow\$ \mathbb{Z}_q$; $Y^\star \leftarrow g^y$
If $\exists P \in \mathcal{S} \cup \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
$K \leftarrow\$ \mathcal{K}$
$\pi^i_U \leftarrow (x, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{T})$
$\pi^j_S \leftarrow (y, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{T})$
Return $(U, X^\star, S, Y^\star)$

proc H($U, S, X^\star, Y^\star, \mathsf{pw}, Z$)
If $(U, S, X^\star, Y^\star) \in T_s$
  $X \leftarrow X^\star / M^{\mathsf{pw}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}}$
  If $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{U, S, X^\star, Y^\star, \mathsf{pw}, Z\}$
If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}, (X, Y) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
  Return $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z]$
If $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z] = \perp$
  $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z] \leftarrow\$ \mathcal{K}$
Return $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z]$

proc Corrupt(U, S)
$C \leftarrow C \cup \{(U, S)\}$; $\mathsf{pw}_{US} \leftarrow\$ \mathcal{P}$
Return pw_{US}

proc Reveal(P, i)
If $\pi^i_P.\mathsf{fr} = \mathsf{F}$ Return $\perp$
$\forall (j, Q)$ s.t. $\pi^j_Q.\mathsf{tr} = \pi^i_P.\mathsf{tr}$ do $\pi^j_Q.\mathsf{fr} = \mathsf{F}$
Return $\pi^i_P.K$

proc Test(P, i)
If $\pi^i_P.\mathsf{fr} = \mathsf{F}$ Return $\perp$
$K_0 \leftarrow$ Reveal(P, i); $K_1 \leftarrow\$ \mathcal{K}$
Return K_b

proc Finalize(b')
For $(U, S) \in (\mathcal{U} \times \mathcal{S}) \setminus C$ do $\mathsf{pw}_{US} \leftarrow\$ \mathcal{P}$
If $\exists \mathsf{pw} \neq \mathsf{pw}'$,
  $(U, S, X^\star, Y^\star, \mathsf{pw}, Z) \in T_{\mathsf{bad}} \wedge$
  $(U, S, X^\star, Y^\star, \mathsf{pw}', Z') \in T_{\mathsf{bad}}$
    If $T_s[U, S, X^\star, Y^\star] = (\star, r, \star)$ s.t.
      $A = g^{m\mathsf{pw}-r} \vee A = g^{n\mathsf{pw}-r} \vee$
      $A = g^{m\mathsf{pw}'-r} \vee A = g^{n\mathsf{pw}'-r}$
    Then call Terminate(DL(A))
Stop.

Adversary $\mathcal{B}_3$

proc Initialize(M, N)
$b \leftarrow\$ \{0,1\}$; $C \leftarrow \{\}$; $T \leftarrow \{\}$; $T_s \leftarrow \{\}$; $T_{\mathsf{bad}} \leftarrow \{\}$
Return (M, N)

proc SendInit(U, i, S)
If $\pi^i_U \neq \perp$ Return $\perp$
$x \leftarrow\$ \mathbb{Z}_q$; $X^\star \leftarrow M^x$
$\pi^i_U \leftarrow (x, (U, S, X^\star, \perp), \perp, \mathsf{F}, \mathsf{F})$
Return $(U, X^\star)$

proc SendResp($S, i, U, X^\star$)
If $\pi^i_S \neq \perp$ Return $\perp$
$y \leftarrow\$ \mathbb{Z}_q$; $Y^\star \leftarrow N^y$
If $\exists P \in \mathcal{S} \cup \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
If $(U, S) \in C$
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $K \leftarrow T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z]$
  Else $K \leftarrow\$ \mathcal{K}$; $T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, (X, Y)] \leftarrow K$
Else
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{U, S, X^\star, Y^\star, \mathsf{pw}, Z\}$
  $K \leftarrow\$ \mathcal{K}$
  $T_s[U, S, X^\star, Y^\star] \leftarrow (S, y, K)$
$\mathsf{fr} \leftarrow (U, S) \notin C$
$\pi^i_S \leftarrow (y, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{fr})$
Return $(S, Y^\star)$

proc SendTerm($U, i, S, Y^\star$)
If $\pi^i_U \neq (x, (U, S, X^\star, \perp), \perp, \mathsf{F}, \mathsf{F})$ Return $\perp$
If $\exists P \in \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
If $(U, S) \in C \wedge (U, S, X^\star, Y^\star) \notin T_s$
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $K \leftarrow T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z]$
  Else $K \leftarrow\$ \mathcal{K}$; $T[U, S, X^\star, Y^\star, \mathsf{pw}_{US}, (X, Y)] \leftarrow K$
Else
  $X \leftarrow X^\star / M^{\mathsf{pw}_{US}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}_{US}}$
  If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}_{US}, Z) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{U, S, X^\star, Y^\star, \mathsf{pw}, Z\}$
  If $T_s[U, S, X^\star, Y^\star] \neq (S, y, K)$
    $K \leftarrow\$ \mathcal{K}$
    $T_s[U, S, X^\star, Y^\star] \leftarrow (U, x, K)$
$\mathsf{fr} \leftarrow \exists j, (U, S, X^\star, Y^\star) = \pi^j_S.\mathsf{tr} \wedge \pi^j_S.\mathsf{fr} = \mathsf{T}$
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (U, S) \notin C$
$\pi^i_U \leftarrow (x, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{fr})$
Return T

proc Exec(U, S, i, j)
If $\pi^i_U \neq \perp \vee \pi^j_S \neq \perp$ Return $\perp$
$x \leftarrow\$ \mathbb{Z}_q$; $X^\star \leftarrow g^x$
$y \leftarrow\$ \mathbb{Z}_q$; $Y^\star \leftarrow g^y$
If $\exists P \in \mathcal{S} \cup \mathcal{U}, (U, S, X^\star, Y^\star) = \pi^j_P.\mathsf{tr}$
  Return $\perp$
$K \leftarrow\$ \mathcal{K}$
$\pi^i_U \leftarrow (x, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{T})$
$\pi^j_S \leftarrow (y, (U, S, X^\star, Y^\star), K, \mathsf{T}, \mathsf{T})$
Return $(U, X^\star, S, Y^\star)$

proc H($U, S, X^\star, Y^\star, \mathsf{pw}, Z$)
If $(U, S, X^\star, Y^\star) \in T_s$
  $X \leftarrow X^\star / M^{\mathsf{pw}}$; $Y \leftarrow Y^\star / N^{\mathsf{pw}}$
  If $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
    $T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{U, S, X^\star, Y^\star, \mathsf{pw}, Z\}$
If $\exists (U, S, X^\star, Y^\star, \mathsf{pw}, (X, Y) \in T$ s.t. $\mathsf{DDH}(X, Y, Z) = \mathsf{T}$
  Return $T[U, S, X^\star, Y^\star, \mathsf{pw}, (X, Y)]$
If $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z] = \perp$
  $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z] \leftarrow\$ \mathcal{K}$
Return $T[U, S, X^\star, Y^\star, \mathsf{pw}, Z]$

proc Corrupt(U, S)
$C \leftarrow C \cup \{(U, S)\}$; $\mathsf{pw}_{US} \leftarrow\$ \mathcal{P}$
Return pw_{US}

proc Reveal(P, i)
If $\pi^i_P.\mathsf{fr} = \mathsf{F}$ Return $\perp$
$\forall (j, Q)$ s.t. $\pi^j_Q.\mathsf{tr} = \pi^i_P.\mathsf{tr}$ do $\pi^j_Q.\mathsf{fr} = \mathsf{F}$
Return $\pi^i_P.K$

proc Test(P, i)
If $\pi^i_P.\mathsf{fr} = \mathsf{F}$ Return $\perp$
$K_0 \leftarrow$ Reveal(P, i); $K_1 \leftarrow\$ \mathcal{K}$
Return K_b

proc Finalize(b')
For $(U, S) \in (\mathcal{U} \times \mathcal{S}) \setminus C$ do $\mathsf{pw}_{US} \leftarrow\$ \mathcal{P}$
If $\exists \mathsf{pw} \neq \mathsf{pw}'$,
  $(U, S, X^\star, Y^\star, \mathsf{pw}, Z) \in T_{\mathsf{bad}} \wedge$
  $(U, S, X^\star, Y^\star, \mathsf{pw}', Z') \in T_{\mathsf{bad}}$
    If $T_s[U, S, X^\star, Y^\star] = (\star, r, \star)$ s.t. $r = \mathsf{pw} \vee r = \mathsf{pw}'$
    Then Stop
    Else
      $u \leftarrow ((x - \mathsf{pw})(x - \mathsf{pw}')(\mathsf{pw}' - \mathsf{pw}))^{-1}$
      call Terminate($(Z^{x-\mathsf{pw}'} \cdot Z'^{\mathsf{pw}-x})^u$)
Stop.

Fig. 4. Security proof for SPAKE2. GCDH and GDL attackers.

7 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 $M = \mathsf{H}(K)$ and $N = \mathsf{H}(K')$ 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 $(M, N) = \mathsf{H}(U, S)$ and $N = \mathsf{H}(S, U)$, 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 unnafected.

The treatment of cases where more complex correlations between passwords may exist is the subject of ongoing work.

Acknowledgments. We would like to thank David Pointcheval for helpful discussions regarding the proof of Lemma 6.4. 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

ABC+06. M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in TLS. In *ASIACCS 06*, pages 35–45. ACM Press, March 2006.

ABM15. M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy*, pages 571–587. IEEE Computer Society Press, May 2015.

ABR01. M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *CT-RSA 2001*, *LNCS* 2020, pages 143–158. Springer, Heidelberg, April 2001.

AFP05. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC 2005*, *LNCS* 3386, pages 65–84. Springer, Heidelberg, January 2005.

AFP06. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. *IEE Proceedings — Information Security*, 153(1):27–39, March 2006.

AP05. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, *LNCS* 3376, pages 191–208. Springer, Heidelberg, February 2005.

BM92. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

BOS18. J. Becerra, D. Ostrev, and M. Skrobot. Forward secrecy of SPAKE2. In *ProvSec 2018*, *LNCS* 11192, pages 366–384. Springer, Heidelberg, October 2018.

BPR00. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, *LNCS* 1807, pages 139–155. Springer, Heidelberg, May 2000.

BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

BR04. M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. http://eprint.iacr.org/2004/331.

HL18. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286, 2018. https://eprint.iacr.org/2018/286.

HL19. B. Haase and B. Labrique. Aucpace: Efficient verifier-based pake protocol tailored for the iiot. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.

HR10. F. Hao and P. Ryan. J-PAKE: Authenticated key exchange without PKI. Cryptology ePrint Archive, Report 2010/190, 2010. http://eprint.iacr.org/2010/190.

HS14. F. Hao and S. F. Shahandashti. The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585, 2014. http://eprint.iacr.org/2014/585.

Jab97. D. P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.

Kra05. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO 2005*, *LNCS* 3621, pages 546–566. Springer, Heidelberg, August 2005.

LK19. W. Ladd and B. Kaduk. SPAKE2, a PAKE. Internet-Draft draft-irtf-cfrg-spake2-09, IRTF, October 2019.

Los19. J. Loss. *New techniques for the modular analysis of digital signature schemes*. PhD Thesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2019.

Mac01. P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. http://eprint.iacr.org/2001/057.

OP01. T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *PKC 2001*, *LNCS* 1992, pages 104–118. Springer, Heidelberg, February 2001.