Protoss
Protocol for Tight Optimal Symmetric Security

Emanuele Di Giandomenico[1], Yong Li[2], Sven Schäge[1]

[1] Eindhoven University of Technology
{e.di.giandomenico,s.schage}@tue.nl
[2] Huawei Technologies Düsseldorf
yong.li1@huawei.com

Abstract. We present Protoss, a new balanced PAKE protocol with optimal communication efficiency. Messages are only 160 bits long and the computational complexity is lower than all previous approaches. Our protocol is proven secure in the random oracle model and features a security proof in a strong security model with multiple parties and multiple sessions, while allowing for generous attack queries including multiple Test-queries. Moreover, the proof is in the practically relevant single-bit model (that is harder to achieve than the multiple-bit model) and tightly reduces to the Strong Square Diffie-Hellman assumption (SSQRDH). This allows for very efficient, theoretically-sound instantiations and tight compositions with symmetric primitives.

Keywords: Protoss, PAKE, password-based key exchange, tight, optimal

1 Introduction

In a password-based key agreement protocol two parties can securely compute a session key over some insecure network. The secret input to both parties is solely a common password. To restrict the damage from low entropy passwords, PAKE (password authenticated key exchange) protocols aim at additionally providing security against offline dictionary attacks, where the attacker can successfully launch a brute-force search on the password without further communication.

State-of-the-Art: Computational Complexity. The most efficient PAKE protocols today follow the SPEKE design (31) which intuitively first hashes the password to derive a new generator $g' = \mathsf{H}(\mathsf{pwd})$ of some common cyclic group G. Next, this generator is used to run a basic Diffie-Hellman key exchange. So instead of a fixed generator g that is part of a shared group setup as usual in classical authenticated key exchange protocols, the SPEKE design uses several generators, one g' for each communication partner, to run the Diffie-Hellman key exchange. A computational disadvantage of this approach is that we cannot easily apply well-known optimization techniques for exponentiation. The existing techniques for fast exponentiation of g essentially rely on using pre-computed powers, so-called ladders of g, consisting of $g^{2^0}, g^{2^1}, g^{2^2}, \ldots$, that are simply multiplied up according to the bit representation of the exponent. In cases where we only use a single generator g for all group related-operations, the computation of such a subset-product is generally very beneficial. Typically, g is chosen with certain properties that make these operations optimal, for example, by letting g have a small hamming-weight.
It is clear that this technique does not provide speedups in case the generator is computed on the fly with some fresh randomness that is derived by both parties, e.g. as $g' = \mathsf{H}(r, \mathsf{pwd})$. In these cases the ladder cannot be pre-computed without knowing r. Moreover, even in case where we have a fixed generator per pair of communication partners that share a password, i.e. $g' = \mathsf{H}(\mathsf{pwd})$, the device has to perform pre-processing operations before the PAKE can be used in an optimized way. Not only does this have to be done for all passwords stored per user. It is also important, that the so-preprocessed ladders $(g'^{2^0}, g'^{2^1}, \ldots)$ are all stored securely on the computer system as they depend on the password. Crucially, any ladder that we so obtain is not only dependent on a secret password, but can essentially be treated as a secret cryptographic key: once an attacker obtains the ladder, she can easily authenticate as the user by participating in a Diffie-Hellman key exchange using g'. Since each ladder for a g' acts like a conventional symmetric secret key, it needs to be protected in

the same way traditional cryptographic keys need to be protected. This defeats the purpose of PAKE protocols. Furthermore, any speed benefits that could be gained do not transfer to new computer systems: whenever Alice logs into a new computer system to run her PAKE protocol, she will always perform all the pre-computations (implicitly or explicitly) before deriving the key since she cannot use existing values that she had computed beforehand.
Using a fixed standardized group and a single generator g has the advantage that a) the ladder is readily available on every computer system, b) that the ladder is not security-critical and thus does not require extra protection, and c) that the ladder relies on a generator that is optimized yielding the highest speed benefits.
Let us clarify again that the use case for password-based protocols does not assume that passwords are securely stored on devices. In case we have secure key storage, we can readily rely on full entropy keys and use traditional key exchange protocols that are less vulnerable to guessing attacks. Indeed one of the main benefits of password-based protocols is that in contrast to plain passwords over TLS, plain passwords are not sent to the server and thus cannot be logged and (inadvertently) be stored on intermediate nodes. This makes password-based authentication much more robust to for example bugs in the logging system that major cloud provider have suffered from[3].

State-of-the-Art: Optimal Message Number and Size. The optimal number of messages for a PAKE protocol in a strong security model is two, one per party. This is because to support perfect forward secrecy (PFS) we need that session keys are not only derived from long-term passwords but also from fresh (public) keys.[4] We note that the smallest asymmetric keys used in practice are always at least 160 bits long (at a security level of around 2^{80} bits) due to birthday attacks.

Tight Security Proofs. All existing practical PAKE protocols with optimal message number and size are characterized by non-tight security losses that either depend on the global parameters or the attacker's behavior (an overview can be found in Table 2). It is well-known that cryptographic constructions with a non-tight security loss transfer to higher security parameters in practice when instantiated in a theoretically-sound way, e.g. (10). Tight security reductions, in contrast, allow smaller parameters and so result in more efficient schemes. Thus they are preferable and in the last years a long line of research has been devoted to finding cryptographic schemes with tight security reductions for cryptographic systems (9; 41; 10; 28; 23; 24). Moreover, for PAKE protocols specifically tight security reductions, are extremely important as pinted out in (5).

Research Question. In this paper, we consider the following question:

Can we construct a PAKE scheme that is optimal in the following three dimensions?
1. *Communication Complexity: The number of bits exchanged should be optimal while only sending a single message per party. Due to the birthday bound this amounts to two 160-bit messages (see (42)).*
2. *To support implementations with aggressive parameter choices in a theoretically-sound way, the protocol should ideally feature a tight security proof in a setting with multiple users and sessions.*
3. *The computational complexity should outperform all existing PAKE protocols.*

Contribution. We present the first PAKE protocol, called Protoss (Protocol for Tight Optimal Symmetric Security), that fulfills all of these properties. The protocol resembles the EKE (16) construction while not relying on ideal ciphers (thus our assumptions are less demanding). This deviates from the widespread SPEKE method of designing PAKE protocols (31). Essentially, our protocol relies on a simple Diffie-Hellman key exchange where each contribution, i.e. each ephemeral public key $epk = g^a$, is blinded by the hashed password of that user as

$$m = \mathsf{H}(\mathsf{pwd}) \cdot g^a.$$

[3] `https://mailarchive.ietf.org/arch/msg/cfrg/1QQ_FfRsOxvoLGqXOt48WLTMSNU/`

[4] Jumping ahead, PFS guarantees that even in case the long-term password is revealed, security of a session is still guaranteed based on the security of the ephemeral asymmetric keys.

Of course knowing the password allows the receiver to re-compute $epk = g^a$ and compute the Diffie-Hellmann value g^{ab}. This key will additionally be hashed together with the transcript T, and other context information to yield the final session key

$$\mathcal{K} = \mathsf{H}(g^{ab}, T, context).$$

Security Assumption. The protocol is proven secure under a security assumption that we call the Strong Squared Diffie-Hellman assumption (SSQRDH). Essentially it says that given g, g^x it is hard to compute g^{x^2} even when given access to an oracle $\mathcal{O}$ that outputs $\mathsf{DDH}(g, g^x, \cdot, \cdot) \in \{0, 1\}$, s.t.

$$\mathsf{DDH}(g, g^x, S, T) = 1 \Leftrightarrow S^x = T.$$

This assumption is closely related to the Strong Diffie-Hellman assumption and in fact it is well known that for very weak oracles (where $\mathcal{O}$ does not output anything) and very powerful oracles (where $\mathcal{O}$ computes $\mathsf{DDH}(g, \cdot, \cdot, \cdot)$) both assumptions are equivalent under polynomial-time reductions.

Model. We use a strong (game-based) model that captures multiple users and sessions while allowing for generous attack capabilities. We allow adaptive corruptions of the passwords and that the attacker can make multiple Test-queries. To provide meaningful results for security loss we also use a single-bit notion of security as advocated in (32) and formally shown necessary for tightly secure hybrid constructions in (43). In Appendix A, we also provide a tight proof of $\mathsf{Protoss}$ security in the UC-based model of (35).

Challenges. Although our construction is very simple and extremely efficient when instantiated in elliptic curves, providing a tight proof (under a single computational security assumption) in our strong model is challenging. In particular, we cannot rely on classical partitioning arguments that are so common in AKE research as they lead to non-constant security losses via guessing. For example, allowing multiple Test queries complicates the proof, as it is now not clear anymore which of the Test-sessions the attacker used to decide whether the challenge key is real or random.

Achieved Security Loss. Our final result shows that the security loss L, i.e. the relative increase

$$L = t_R/\epsilon_R \cdot \epsilon_A/t_A$$

in resources used by the reduction, does not lose a multiplicative factor, i.e. $\epsilon_R \approx \epsilon_A$ and $t_R \approx t_A$ for runtime t and success probability ϵ. This results in a tight security proof (with a constant loss).

Construction Idea. Our construction deviates from the widespread SPEKE formula where, intuitively, classical Diffie-Hellman-based AKE protocols are turned into PAKE protocols by making the generator be computed as the hashed password, i.e. $g_{\mathsf{pwd}} = \mathsf{H}(\mathsf{pwd})$ and then applying classical ideas. $\mathsf{Protoss}$ rather follows the EKE2 paradigm (12) that encrypts Diffie-Hellman shares symmetrically as $m = \mathsf{ENC}(\mathsf{pwd}, g^a)$ where ENC is modeled as an ideal cipher and the password is used as the symmetric encryption key. However, we do not rely on ideal ciphers in our construction and thus overall can rely on weaker assumptions. For the proof we follow a technique that is inspired by (42). Intuitively, $\mathsf{Protoss}$ consists of a simple exchange of the hashed password that additionally is multiplicatively blinded in each message by a fresh Diffie-Hellman share. This guarantees that the hashed password is statistically hidden thus making offline attacks impossible from the protocol messages alone. The Diffie-Hellman shares are then used to derive a Diffie-Hellman key which, after another hashing with additional context information, results in the final session key.
On a high level, our proof idea is similar to the one taken by Paterson et al. (34). In a nutshell, the reduction makes each session key be a truly random value output by the random oracle. The difficulty lies in the fact that we need to ensure consistency with random oracle queries in case the attacker can correctly compute all the inputs to the final hash call and send it to the random oracle. However, the use of the additional decision oracle helps us to recognize such input and, once it is received, we can break the complexity assumption. One qualitative difference between our work and (34) is that we do not rely on full-fledged gap problems where the attacker is given access to a decision oracle that can recognize tuples g, S, T such

that $S^x = T$ for any x. The SSQRDH variant that we require only needs to work for a fixed exponent x.
A construction very similar to ours has already been proposed as a variant of the EKE2 protocol in (12) but has not been formally investigated further. Here the authors propose to use $\mathsf{ENC}(pw, M) = M \cdot h(pw)$ and $\mathsf{DEC}(pw, M) = M/h(pw)$ as encryption and decryption algorithms and refer to the analysis of this case as ongoing work. To the best of our knowledge, there is neither a dedicated analysis of this protocol in the literature nor do the authors show that their proposal is even suitable to instantiate their generic EKE2 protocol. In any case, the work of (12) treats the algebraic operations only as instantiations of ciphers that could be used for EKE2. Moreover, their generic proof is non-tight. (As a side note, we remark, in 2006, Zhao, Dong, and Wang (44) showed an attack on an instantiation of the idea. However, a closer inspection reveals, that this attack relies on a hash function that does not map to the underlying prime order group). We remark that (12) rely on standard definitions of classical symmetric encryption systems where encryption and decryption operations cannot rely on common intermediate results. Essentially, this reflects the typical application scenario of encryption systems where encryption and decryption operations are performed by distinct parties and thus cannot rely on common computations. Protoss however, heavily exploits the fact that both message generation (that corresponds to an encryption operation in EKE2) and key derivation (that includes a decryption operation in EKE2) rely on the same value $V = h(pw)$. In Protoss, this value thus only has to be computed once. This represents a new avenue for computational savings as compared to EKE2. At the same time, however, the proof of EKE2 (12) that relies on the ideal cipher abstraction does formally not apply to Protoss, and we need to provide a new, dedicated proof. Our new proof only relies on the (weaker) random oracle model, and fortunately, it features a tight security reduction.
Another construction that is very similar to Protoss is the PPK protocol in (20) that is defined over finite fields. The main differences are that it uses a more complex password hashing while not re-using the hashed passwords when deriving the symmetric key of the initiator. Moreover, it uses a simulation-based security model and features a non-tight security proof that relies on guessing arguments.

Performance. In terms of efficiency, our construction outperforms all previous constructions. This is because essentially, each message consists of only a blinded hashed password. The blinding factor that is required to compute the session key consists of a single Diffie-Hellman share that was computed via an exponentiation to a *fixed* base. Moreover, this base is public and the same for each communication partner so that a single ladder in combination with a single well-chosen generator can be used everywhere. For standard groups, this ladder is likely to be already be pre-computed on all systems that support the group. At the same time, extracting the DH share from a message simply consists of a single group operation.

Related Work. In the last three decades, several PAKE protocols have been proposed. They can roughly be grouped in two sets, symmetric or asymmetric (balanced or unbalanced) PAKE protocols. Asymmetric PAKE protocols like (33; 38; 22; 40) typically consider a communication scenario where several clients communicate with a single server. To contain the damage from server corruptions, the server may not obtain the full password of the client but only a value that is derived from it. Our protocol Protoss in contrast is symmetric such that both communication partners share the same password. Generally, symmetric PAKE protocols are overall more efficient than asymmetric ones. Other important symmetric PAKE protocols are EKE (17), EKE2 (12), SPAKE1 (7), SPAKE2 (7), SPEKE (31; 30), J-PAKE (29), CPace (6). However, none of them fulfills all our requirements simultaneously:

– Several protocols or variants like EKE, J-PAKE, and PAK, have more than two protocol moves.
– Several protocols, like EKE, EKE2 and J-PAKE, use additional cryptographic primitives like symmetric encryption systems or zero-knowledge proofs, besides group operations.
– Protocols like J-PAKE exchange messages that can be larger than 160 bits, since they have to contain zero-knowledge proofs.
– Protocols like (35) have message sizes much larger than 160 bits, since each message contains the encryption of two group elements.
– With respect to computational efficiency we have that the most efficient among these protocols, like CPace, rely on the SKEME approach where in the first step, fresh generators g' are derived that are then used in a Diffie-Hellman key exchange. When considering the difference between the computation of $m_1 = \mathsf{H}(\mathsf{pwd})^x$ and Protoss's approach

$m_2 = \mathsf{H}(\mathsf{pwd})g^x$, we can see that the latter involves a single fixed-base exponentiation while the former deals with variable-base exponentiations. It is well-known that fixed-base exponentiations can be faster than variable-base exponentiations (36). When deriving keys later, the two approaches are similar in efficiency, both accounting for a single variable-base exponentiations: $m_1^y = (\mathsf{H}(\mathsf{pwd}))^{xy}$ vs. $(m_2/(\mathsf{H}(\mathsf{pwd})))^y = g^{xy}$. We note that the computation of $(\mathsf{H}(\mathsf{pwd}))^{-1}$ simply amounts to a change of sign in elliptic curves when using point coordinates.

On the other hand, the SPAKE1 and SPAKE2 protocols use two fixed-base computations to create a message $m = g^x M^{\mathsf{pwd}}$ (or one multi-exponentiations) where both g, M are public generators. Since the key derivation is again comparable to the one in Protoss, Protoss is overall more efficient. For an overview see Table 2.

– None of the existing protocols has been shown to be provably secure with a tight reduction that allows aggressive but theoretically-sound parameter choices in practice. Protocols that give explicit concrete bounds include TBPEKE, e.g. (37). The security proof of that protocol refers to the proof of VBTBPEKE in the same paper. The corresponding security theorem claims a security loss of $(q_H)^2$ where H is a hash function. In contrast, the SPAKE2 (2) protocol loses a factor of $2q_s$ in the security reduction in a strong model that provides perfect forward secrecy, where q_s is the number of Send-queries. (The security proof of Protoss also provides perfect forward secrecy, see Section 3.1.)

On the Importance of the Security Framework. In the literature, we can find two different approaches to formally analyze PAKE protocols. Several authors have already contrasted these two frameworks, e.g. (37; 18; 1). The first framework uses classical game-based security reductions, whereas second relies on the universal composability (UC) framework. UC definitions are better suited for handling general correlations between passwords, e.g., when a client uses unequal but related passwords with different servers. At the same time, UC definitions also ensure security under arbitrary protocol compositions, an advantage that makes UC-secure protocols usable in a very flexible way. However, there are also major disadvantages when proving security in the UC framework. For our purposes, the most important one is that UC-based proofs say very little about how security parameters should be implemented in practice when considering realistic application scenarios with multiple users, multiple sessions, and adaptive corruptions. One the one hand, UC-based proofs have generally higher definitional complexity (double-quantifier structure) that complicates concrete-security assessments (13). On the other hand, they tend to simplify the analysis by showing security for only a single session. For the proof of CPace, this is explicitly stated as a goal to simplify the analysis (6). Next, the composition theorem is invoked to argue for the security of multiple sessions. We stress that this is a hybrid argument in essence. And as such it will generally incur a security loss in scenarios with multiple sessions that is equal to the maximum number of sessions per user. At the same time, analyses usually only concentrate on a single party. Again, when considering multi-user security, this essentially transfers to a hybrid argument that would account for another factor in the security loss - the number of users in the system. Thus, in this work, we rely on a game-based security analysis like (39; 2). This is the standard for all works that investigate tight security reductions. We proceed by specifying a strong and realistic security model with multiple users, multiple sessions, and even multiple test queries (while using only a single, global bit to define indistinguishability) and precisely derive the corresponding security loss. In Section 6.1, we argue why our notion is suited to be used in hybrid cryptographic systems (PAKE plus symmetric primitives) that feature tight security reductions.

Abdalla's Open Problem. By providing the first practical PAKE protocol with a tight reduction, Protoss addresses an observation by Abdalla:[5] "Interestingly, proving perfect forward secrecy without the key confirmation step seems to be significantly harder. In the case of SPAKE2, we were able to provide a game-based proof for it under gap CDH[6], but the reduction is not tight and requires an intermediate assumption previously used in the proof of SPAKE1. A similar result is currently not known for CPace, TBPEKE, and SPEKE."

[5] `https://mailarchive.ietf.org/arch/msg/cfrg/XOKmh5lKMsCQhWJTIjoYxmbleZU/`, see bullet point 5.
[6] `https://eprint.iacr.org/2019/1194`

2 Notation and Preliminaries

Notation We let $\kappa \in \mathbb{N}$ denote the security parameter and 1^κ the string that consists of κ ones. Let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ be the set of integers between 1 and n. If S is a set, $a \xleftarrow{\$} S$ denotes the action of sampling a uniformly random element from S. If $\mathcal{A}()$ is an algorithm, $m \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\cdot)}()$ denotes that $\mathcal{A}$ (probabilistically) outputs m with the help of another algorithm $\mathcal{O}(\cdot)$. Let $X||Y$ denote the operation concatenating two binary strings X and Y. For a generator g and two group elements S, T with $S = g^x$ and $T = g^y$ we use $\mathsf{CDH}(g, S, T) \in G$ to denote the Diffie-Hellman value of S and T as $\mathsf{CDH}(g, S, T) = g^{xy}$. Finally, we use the predicate $\mathsf{DDH}(g, S, T, U) \in \{0, 1\}$ which equals 1 iff $\mathsf{CDH}(g, S, T) = U$. If $\mathcal{A}(\cdot)$ is a probabilistic algorithm, we may also make the randomness rand explicit via $\mathcal{A}(\cdot; \mathsf{rand})$.

Strong Square Diffie-Hellman Problem Let G be a cyclic group of a large prime order p. Let g be a generator of G. The Strong Square Diffie-Hellman Problem (SSQRDH) is defined as: given a pair (g, g^x) for $x \xleftarrow{\$} \mathbb{Z}_p$, find the element $C = g^{x^2}$ with the help of a Decisional Diffie-Hellman Oracle

$$\mathcal{O}_x(\cdot, \cdot) := \mathsf{DDH}(g, g^x, \cdot, \cdot).$$

This means, the oracle $\mathcal{O}_x(\cdot, \cdot)$ answers whether for a given pair $S, T \in G$ we have $S^x = T$.

Definition 1. *We say that A (t, ϵ)-breaks the Strong Square Diffie-Hellman problem if A runs in time t while having the following success probability*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SSQRDH}} := \Pr\left[Z = g^{x^2} : Z \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_x(\cdot,\cdot)}(g, g^x); x \xleftarrow{\$} \mathbb{Z}_p\right] \geq \epsilon,$$

where the probability is taken over the random coins of the attacker and the random choice of x. We say that the (t, ϵ)-SSQRDH assumption holds if no attacker can (t, ϵ)-break the Strong Square Diffie-Hellman problem.

In case no oracle queries are allowed in the above definition we obtain the classical Square Diffie-Hellman assumption (SQRDH) (25). The SSQRDH assumption is conjectured to hold in bilinear groups where an efficient decision procedure is readily available via the pairing operation. In this work, we assume that the assumption also holds in elliptic curves more generally. For comparison let us review the classical Strong Diffie-Hellman assumption (SDH) (4; 22).

Definition 2. *We say that A (t, ϵ)-breaks the Strong Diffie-Hellman problem if A runs in time t while the following success probability*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SDH}} := \Pr\left[Z = g^{xy} : Z \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_x(\cdot,\cdot)}(g, g^x, g^y); x, y \xleftarrow{\$} \mathbb{Z}_q\right] \geq \epsilon$$

where the probability is taken over the random coins of the attacker and the random choice of x, y. We say that the (t, ϵ)-SDH assumption holds if no attacker can (t, ϵ) break it.

As before, in case no oracle queries are allowed, we obtain the well-known Decisional Diffie-Hellman (DDH) assumption.
Let us investigate the relationship between the SSQRDH assumption and the SDH assumption. We can show that the SSQRDH assumption implies the SDH assumption.

Lemma 1. *The (t, ϵ)-SSQRDH assumption implies the (t, ϵ)-SDH assumption.*

Proof. Assume we are given the SSQRDH challenge $g, g^{x'}$ while having access to an SDH attacker A that takes as input g, g^x, g^y. The reductions draws random exponent r and sets $g^x := g^{x'}$ and $g^y := g^{x'r}$. Moreover, the reduction relays any oracle queries that A makes to the SDH oracle and the results back to A. Finally, A outputs $T = g^{xy} = g^{(x')^2 r}$ which immediately helps the reduction to compute a solution to the SSQRDH problem as $T^{1/r} = g^{(x')^2}$.

It is also well-known that in case the attacker is not allowed to query oracles, the SSQRDH and the SDH assumptions are equivalent:

Lemma 2. *If the attacker is not allowed to make any oracle calls, the* $(2t, \epsilon^2)$-SDH *assumption implies the* (t, ϵ)-SSQRDH *assumption under Turing reductions.*

Proof. Use $g, g^{x'}$ to denote the input to the SSQRDH attacker. Given the SDH challenge g, g^x, g^y, the reduction can compute g, g^{xr}, g^{ys} for random $r, s \xleftarrow{\$} \mathbb{Z}_p$ and then use the SSQRDH attacker twice, on $U = g^{(xr+ys)/2}$ and $V = g^{(xr-ys)/2}$ to obtain U^2 and V^2. From that the reduction can easily compute $\left(U^2/V^2\right)^{1/(rs)} = g^{xy}$. The overall probability for this to happen is ϵ^2.

Definition 3 (Gap Assumptions). *Assume in Definitions 1 and 2 we substitute* $\mathcal{O}_x = \mathsf{DDH}(g, g^x, \cdot, \cdot)$ *by*

$$\mathcal{O}_*(\cdot, \cdot, \cdot) := \mathsf{DDH}(g, \cdot, \cdot, \cdot)$$

and denote the resulting assumptions as SSQRDH^* *and* SDH^**.*

In the literature, the SDH^* assumption is commonly known as the *Gap Diffie-Hellman* assumption and the SSQRDH^* assumption is known as the *Gap SQRDH* assumption (8). We remark that some authors define the Gap Diffie-Hellman assumption to also allow the attacker to choose the generator g when evaluating the oracle.

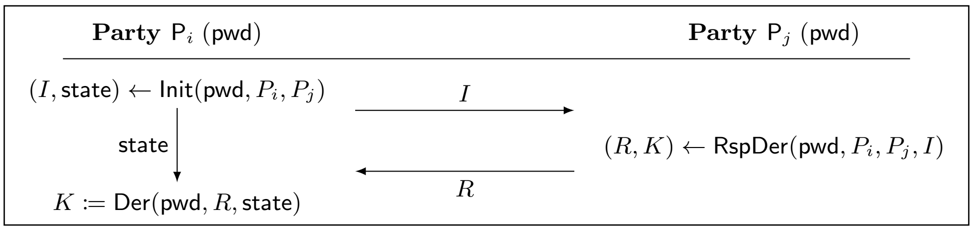
Lemma 3. *Under Turing reductions the* $(2t, \epsilon^2)$-SDH^* *assumption implies the* (t, ϵ)-SSQRDH^* *assumption.*

Proof. The proof essentially uses the reduction idea of the proof of Lemma 2. The only issue is that we now also have to show how the oracle of the SSQRDH^* assumption can be used to simulate queries to the SDH^* oracle. However, since both oracles are equal and now independent of any of the exponents in the challenge g, g^x, g^y (respectively $g, g^{x'}$), one oracle can always be directly used to simulate the other by simply relaying queries to each other and answers back.

The security of the Gap SQRDH assumption holds in the algebraic group model under the discrete logarithm assumption with a tight reduction (11). Since a full-fledged gap oracle $\mathcal{O}_*(\cdot, \cdot, \cdot)$ provides more freedom to an attacker as our more restricted DDH oracle $\mathcal{O}_x(\cdot, \cdot)$, it is obvious that the SSQRDH assumption also holds in the algebraic group model with a tight security reduction under the discrete logarithm assumption (in fact the proof in (11) holds for oracles that check general polynomial relations between the discrete logarithms of input group elements and thus directly also covers the SSQRDH assumption).

3 PAKE

A PAKE protocol $\mathsf{PAKE} = (\mathsf{Init}, \mathsf{RspDer}, \mathsf{Der})$ consists of three algorithms which are executed interactively by two parties as shown in Figure 1. Let us assume the party which initiates the session is P_i and the party which responds to the first message is P_j. We assume that at the beginning P_i and P_j have shared a password pwd uniformly random from some dictionary D of size $|D|$. The initialization algorithm Init inputs pwd, P_i, P_j and outputs a message $I \in \mathcal{M}$ and a state state. The responder's response and derivation algorithm RspDer takes as input pwd, P_i, P_j and a message I. It computes a message $R \in \mathcal{M}$ and a session key K. The initiator's derivation algorithm Der inputs pwd, a message R and a state state. It outputs a session key K.



Party P_i (pwd)
Party P_j (pwd)
$(I, \mathsf{state}) \leftarrow \mathsf{Init}(\mathsf{pwd}, P_i, P_j)$
I
state
$(R, K) \leftarrow \mathsf{RspDer}(\mathsf{pwd}, P_i, P_j, I)$
$K := \mathsf{Der}(\mathsf{pwd}, R, \mathsf{state})$
R

Fig. 1: Running a key exchange protocol between two parties.

3.1 Security Model for PAKE

In this section, we present a formal security model for two party password-based AKE protocols (PAKE) that builds on the classical model by Bellare, Pointcheval and Rogaway (12). Following the tradition of the seminal work of Bellare and Rogaway (14) (39), we provide an 'execution environment' for adversaries that emulates the attack capabilities of an active adversary.

Execution Environment. In the following let $\ell, d \in \mathbb{N}$ be positive integers (indicating the number of overall parties and overall sessions per party). In the execution environment, we fix a set of ℓ honest parties $P = \{P_1, \ldots, P_\ell\}$. In our symmetric setting, each party holds symmetric long-term passwords, each of which is shared with one of the other parties. We denote with $\mathsf{pwd}_{i,j} = \mathsf{pwd}_{j,i}$ $(i \neq j)$ the symmetric password shared between parties P_i and P_j.
Next, the execution environment manages for every $i \in [\ell]$ a vector $\mathsf{corruptionstate}_i \in \{\mathsf{corrupted}, \mathsf{uncorrupted}\}^\ell$ that is used to denote the corruption state of all long-term passwords used by P_i, each component $\mathsf{corruptionstate}_{i,j} \in \{\mathsf{corrupted}, \mathsf{uncorrupted}\}$ indicating whether $\mathsf{pwd}_{i,j}$ has been corrupted or not. Additionally, the execution environment uses vector $\mathsf{corruptionTime}_i \in [\ell \cdot d]^d$, to store the times of corruption, each component $\mathsf{corruptionTime}_{i,j}$ indicating that $\mathsf{pwd}_{i,j}$ was corrupted via the $\mathsf{corruptionTime}_{i,j}$-th query of the adversary. Moreover, the execution environment draws a global bit $b \in \{0, 1\}$, indicating whether or not the attacker will be presented with a real or a random key when making a Test-query (as defined next). The bit b will help to capture single-bit security. As argued in (32), this is the right notion of security when examining tight security reductions in the multi-user setting. For a more in-depth discussion, see Section 6.1.
Each honest party P_i can sequentially and concurrently execute the protocol multiple times. This is characterized by a collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. Oracle π_i^s behaves as party P_i, carrying out a process to execute the s-th protocol instance with some partner P_j. All oracles of P_i have access to $\mathsf{pwd}_{i,j}$ with $j \in \{1, \ldots, \ell\}$. Moreover, we assume each oracle π_i^s maintains a list of independent internal state variables as described in Table 1, each of which uses supertext s and subtext i to indicate that it is held by π_i^s. These variables are: $\mathsf{exstate}_i^s$, Pid_i^s, role_i^s, K_i^s, and T_i^s.

Variable	Description
$\mathsf{exstate}_i^s$	denotes the execution-state of oracle π_i^s, i.e. $\mathsf{exstate}_i^s \in \{\mathsf{uninitialized}, \mathsf{waiting}, \mathsf{accepted}\}$
Pid_i^s	stores the identity of the intended communication partner, i.e. $\mathsf{Pid}_i^s \in P$
role_i^s	denotes the role that P_i assumes in the protocol run: $\mathsf{role}_i^s \in \{\mathsf{uninitialized}, \mathsf{initiator}, \mathsf{responder}\}$
K_i^s	stores the session key(s) $\mathsf{K}_i^s \in \mathcal{K} \cup \{\mathsf{uninitialized}\}$
T_i^s	records the transcript of messages sent and received by oracle π_i^s in chronological order, i.e. $T \in \mathcal{M} \times \mathcal{M}$
state_i^s	is used to store the state of the initiator.

Table 1: Internal States of Oracle π_i^s

Additionally, the execution environment associates to each oracle the variable $\mathsf{keystate}_i^s$ which denotes the freshness $\mathsf{keystate}_i^s \in \{\mathsf{exposed}, \mathsf{fresh}\}$ of the session key. The remaining variables are only used by the execution environment. Among these variables, the execution environment will dedicate uniformly random bits rand_i^s to each oracle. The variables of each oracle π_i^s will be initialized as follows:
– The execution-state $\mathsf{exstate}_i^s$ is set to $\mathsf{uninitialized}$.
– The variable $\mathsf{keystate}_i^s$ is set to fresh.
– All other variables are set to only contain the special string $\mathsf{uninitialized}$.
We always have that $\mathsf{K}_i^s = \mathsf{uninitialized}$ iff π_i^s has not reached $\mathsf{accepted}$-state (yet).

Definition 4 (Partnering). *Two oracles (π_i^s, π_j^t) are said to be partnered if they share the same transcript $\mathsf{T}_i^s = \mathsf{T}_j^t$ and $\mathsf{Pid}_i^s = j$ and $\mathsf{Pid}_j^t = i$.*

Adversarial Model. An adversary $\mathcal{A}$ in our model is a probabilistic algorithm with polynomial running time (PPT), which takes as input the security parameter 1^κ and the public information, and may interact with these oracles by issuing the following queries.

– $\mathsf{SendInit}(\pi_i^s, j)$: if $\mathsf{exstate}_i^s \neq \mathsf{uninitialized}$ this query aborts. Otherwise, this query calls

$$(I, \mathsf{state}) \leftarrow \mathsf{Init}(\mathsf{pwd}_{i,j}, P_i, P_j)$$

and outputs I. Additionally, it sets
  - $\mathsf{exstate}_i^s = \mathsf{waiting}$,
  - $\mathsf{role}_i^s = \mathsf{initiator}$,
  - $\mathsf{T}_i^s = I$,
  - $\mathsf{state}_i^s = \mathsf{state}$, and
  - $\mathsf{Pid}_i^s = j$

– $\mathsf{SendRspDer}(\pi_i^s, j, I)$: if $\mathsf{exstate}_i^s \neq \mathsf{uninitialized}$ this query aborts. Otherwise, this query calls

$$(R, \mathsf{K}) \leftarrow \mathsf{RspDer}(\mathsf{pwd}_{i,j}, P_i, P_j, I)$$

and outputs R. Additionally, it sets
  - $\mathsf{exstate}_i^s = \mathsf{accepted}$,
  - $\mathsf{role}_i^s = \mathsf{responder}$,
  - $\mathsf{T}_i^s = I, R$,
  - $\mathsf{K}_i^s = \mathsf{K}$, and
  - $\mathsf{Pid}_i^s = j$

– $\mathsf{SendDer}(\pi_i^s, R)$: if $\mathsf{exstate}_i^s \neq \mathsf{waiting}$ this query aborts. Otherwise, this query calls

$$\mathsf{K} \leftarrow \mathsf{Der}(\mathsf{pwd}_{i,j}, \mathsf{state}_i^s, R).$$

Additionally, it sets
  - $\mathsf{exstate}_i^s = \mathsf{accepted}$,
  - $\mathsf{T}_i^s := \mathsf{T}_i^s, R$, and
  - $\mathsf{K}_i^s = \mathsf{K}$.

– $\mathsf{Corrupt}(i, j)$: the execution environment returns $\mathsf{pwd}_{i,j}$ and sets

$$\mathsf{corruptionstate}_{i,j} := \mathsf{corruptionstate}_{j,i} := \mathsf{corrupted}.$$

Moreover, if this was the w-th overall query of the attacker, the execution environment sets

$$\mathsf{corruptionTime}_{i,j} := \mathsf{corruptionTime}_{j,i} := w.$$

We also say that $\mathsf{pwd}_{i,j}$ and $\mathsf{pwd}_{j,i}$ is w-corrupted.

– $\mathsf{RevealKey}(\pi_i^s)$: this oracle will output key K_i^s in case that $\mathsf{exstate}_i^s = \mathsf{accepted}$ and abort otherwise. Oracle π_i^s responds to a $\mathsf{RevealKey}$-query with the contents of variable K_i^s and sets $\mathsf{keystate}_i^s = \mathsf{exposed}$. If at the point when an adversary issues this query there exists another oracle π_j^t which is partnered to π_i^s, then $\mathsf{keystate}_j^t = \mathsf{exposed}$ for π_j^t.

– $\mathsf{Test}(\pi_i^s)$: if the oracle π_i^s has state $\mathsf{exstate}_i^s \neq \mathsf{accepted}$, then we abort. If the oracle π_i^s is queried for the first time, a random key $\hat{\mathsf{K}}_i^s \overset{\$}{\leftarrow} \mathcal{K}$ is drawn. Finally, K_b is returned, where K_0 is the real key K_i^s and K_1 is the random key $\hat{\mathsf{K}}_i^s$.

– $\mathsf{Execute}(\pi_i^s, \pi_j^t)$: we define this query for convenience only. This query internally simply calls

$$I \leftarrow \mathsf{SendInit}(\pi_i^s, j), \;\; R \leftarrow \mathsf{SendRspDer}(\pi_j^t, i, I),$$

and

$$\mathsf{SendDer}(\pi_i^s, R).$$

We also say that this query models a *passive* protocol run between the two oracles π_i^s, π_j^t.

Let us now capture what it means for π_i^s to be attacked passively and actively. This is important when we define security in the presence of corruptions. Intuitively, a passive attack on an oracle says that the oracle receives a message such that the attacker cannot trivially compute the session key.

Definition 5 (Passive and Active Attacker). *We say that an attacker is passive with respect to π_i^s if the following two conditions always hold:*

1. *If the ℓ-th query was $\mathsf{SendRspDer}(\pi_i^s, j, I)$ then either i) there has also been a query $I \leftarrow \mathsf{SendInit}(\pi_{j'}^t, i')$ that has output I (i.e. message I has been produced by an oracle) or ii) $\mathsf{Corrupt}(i, j)$ has not been queried before query ℓ (i.e. the password remains uncorrupted).*
2. *If the ℓ-th query was $\mathsf{SendDer}(\pi_i^s, R)$ then either i) there has also been a query $R \leftarrow \mathsf{SendRspDer}(\pi_j^t, i, \cdot)$ that has output R (i.e. message R has been produced by an oracle) or ii) $\mathsf{Corrupt}(i, \mathsf{pid}_i^s)$ has not been queried before query ℓ.*

We say that the attacker is active with respect to π_i^s if it is not passive with respect to session π_i^s, i.e. if one of the two conditions is violated.

Note that in the case of passive attacks we do not require that the message m received by π_i^s has been generated by a potential partner oracle. Rather, m could be any message that has been produced by the query $\mathsf{SendInit}$ or $\mathsf{SendRspDer}$. What this intuitively guarantees is that the implicit ephemeral key of m remains hidden from the attacker. This is slightly stronger than the notion of origin session as defined by Cremers and Feltz that implicitly defines passively secure runs by defining a prefix relation among two *oracles*: essentially, the transcript generated by one oracle should be the prefix of the transcript generated by some other oracle. In contrast, our definition allows for example the message I sent by initiator oracle π_i^s to be modified on transit by the attacker. Nevertheless, as long as π_i^s receives a message from some other oracle (as a result of a call to $\mathsf{SendRspDer}$), we always say that the attacker is passive with respect to π_i^s.

Definition 6 (Completeness). *We say that a* PAKE *protocol Π is* complete*, if for any two oracles π_i^s, π_j^t that are partnered with $\mathsf{Pid}_i^s = j$ and $\mathsf{Pid}_j^t = i$ and $\mathsf{exstate}_i^s = \mathsf{accepted}$ and $\mathsf{exstate}_j^t = \mathsf{accepted}$ it always holds that $\mathsf{K}_i^s = \mathsf{K}_j^t$.*

Definition 7 (Security Game). *We formally consider a security experiment that is played between an adversary and a challenger. In this game, the challenger implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$. First, the challenger draws a fair coin $b \in \{0,1\}$. Next, it generates long-term passwords $\mathsf{pwd}_{i,j}$ for each pair of honest parties i, j by drawing each $\mathsf{pwd}_{i,j} \xleftarrow{\$} D$. The adversary may start issuing* $\mathsf{SendInit}$*,* $\mathsf{SendRspDer}$*,* $\mathsf{SendDer}$*,* $\mathsf{Corrupt}$*,* $\mathsf{RevealKey}$*,* $\mathsf{Execute}$ *and* Test *queries. Finally, the adversary outputs a bit b' that indicates its guess of b and terminates.*

For the security definition, we need the notion of freshness of oracles.

Definition 8 (Oracle Freshness). *Let π_i^s be an accepting oracle held by party P_i with intended partner $\mathsf{Pid}_i^s = P_j$. Let π_j^t be an oracle (if it exists), such that π_i^s and π_j^t are partnered. Then the oracle π_i^s is said to be w-fresh for $w \in \mathbb{N}$ if none of the following conditions holds:*

– *the adversary has either made a $\mathsf{RevealKey}(\pi_i^s)$ query or a $\mathsf{RevealKey}(\pi_j^t)$ query, (if π_j^t exists),*
– *P_i or P_j is w'-corrupted with $w' \leq w$ and the attacker has been active with respect to π_i^s after query w',*
– *the attacker has made query $\mathsf{Test}(\pi_j^t)$.*

Observe that in the above definition, it is only a violation of freshness if the adversary sends attacker messages to π_i^s after P_i has been corrupted. However, even after the corruption of P_i, sending messages to π_i^s that have been constructed by some π_j^t is not considered a violation. This definition is essential in capturing full PFS as it helps to formalize security even if the attacker sends her own messages to π_i^s. The crucial fact is that whenever those messages have been sent to π_i^s, P_i must not be corrupted – as this would trivially break security: the adversary can easily obtain all secret information needed to compute the session key.
In the following, we provide a general security definition for PAKE protocols that captures key indistinguishability.

Definition 1 (PAKE Security Definition)**.** We say that an adversary A (t, ϵ)-*breaks* a PAKE protocol if the following three conditions are all fulfilled:

1. A runs in time t.

2. When the adversary terminates and outputs a bit b' it always holds:

$$\text{the } w\text{-th query is } \mathsf{Test}(\pi_i^s) \text{ for some oracle } \pi_i^s \quad \Rightarrow \quad \pi_i^s \text{ is } w\text{-fresh.}$$

3. The probability that b' equals b is bounded by

$$\left|\Pr[b' = b] - 1/2\right| \geq \epsilon + (q_{\mathsf{SendRspDer}} + q_{\mathsf{SendDer}})/|D|$$

where $q_{\mathsf{SendRspDer}}$ is the number of SendRspDer-queries made and q_{SendDer} is the number of SendDer-queries. If an adversary outputs b' such that $b' = b$ and the above conditions are met, then we say that the adversary *answers the* Test*-challenge correctly.* We say that the PAKE protocol is (t, ϵ)-*secure*, if there exists no adversary that (t, ϵ)-*breaks* it.

We remark that this is a very strong security model that can be regarded as a combination of the classical model of (12) with ideas from (32). We also remark that drawing passwords uniformly is just for simplicity. We support any password distribution (15). This is because in our protocol, passwords are immediately hashed and the hashed value is used throughout all the remaining computations. In the ROM, all distributions map to uniformly random values, so even if passwords are related, as long as they are distinct they will map to independent values. Jumping ahead, our theorem does not explicitly mention the password distribution at all. Of course, our protocol would thus also work if we exchange passwords for shared high-entropy keys. This makes it a very efficient tool in scenarios where parties have a shared long-term key but now would love to derive secure session keys (featuring perfect forward secrecy) to communicate with.

Perfect Forward Secrecy. We stress that our security definition models strong perfect forward secrecy. To see this, we need to show that two conditions, i) and ii), are fulfilled which grant the attacker the right to choose the oracle π that it wants to test in a very liberal way. The first condition i) says that after π finished its computations, the attacker may corrupt the password. If this condition only applies to sessions π that have received messages from other oracles (instead of adversarially generated ones) researchers often speak of weak perfect forward secrecy. However, several authors have argued that weak perfect forward secrecy (wPFS) is not enough in practice, see for example (26; 42). Therefore we immediately define the stronger notion of (strong) PFS, which allows that the attacker may also corrupt sessions π that i) have finished and ii) received a message produced by the attacker. Our security definition captures strong PFS since it does at no point require tested sessions to not have received attacker messages. In case π has received an attacker message, our freshness notion simply requires that π's password has not been corrupted before π finished and that no RevealKey query was asked to π (trivial attack).

Impossibility of RevealState *Queries.* At the same time, and like all other existing (symmetric) PAKE protocols, we do not consider state reveal attacks in our model, in contrast to models for classical AKE protocols. Although this may seem like a severe restriction, we can observe that any PAKE protocol which allows the adversary to reveal states cannot provide security of the session key (key indistinguishability). Briefly, if an attacker can obtain the ephemeral secret state through some RevealState-query for any session, she can next reveal the session key K of the same session via a RevealKey query. She can now perform an offline brute force search for the password by calling the key derivation algorithm for different passwords until the output equals K. This is possible since the key derivation algorithm is deterministic and all secret values that are input to the key derivation algorithm – except for the password – are known to the attacker. With the so obtained password, the attacker can activate a fresh session as test session by selecting a fresh ephemeral secret key esk' and computing a valid message m_1 by using pwd to an honest party (P_1) which in return responds with a valid message m'. If this session is completed, because the attacker knows the pwd and the ephemeral secret esk' contained in m_1, it can compute the same session key of this fresh session, thereby breaking the security of the session key.

4 Protocol Description

Here we present our new symmetric PAKE protocol called Protoss (short for Protocol with Tight, Optimal, Symmetric Security). It is defined in a public cyclic group G of prime order

p generated by $g \in G$ and the hash functions H and H'. We use multiplicative notation. $\mathsf{H} : \{0,1\}^* \to G$ maps passwords to group elements. The hash function $\mathsf{H}' : \{0,1\}^* \to \mathcal{K}$ outputs elements in the session key space $\mathcal{K}$. We model H, H' as random oracles in our proof. Our Protoss protocol is symmetric and can be implemented with two flows. We assume that P_i holds $\mathsf{pwd}_{i,j}$ and that P_j has $\mathsf{pwd}_{j,i}$, where $\mathsf{pwd}_{i,j} = \mathsf{pwd}_{j,i}$. The protocol construction is depicted in Figure 2. In order to establish a session key, parties P_i and P_j can use the following algorithms:

- $\mathsf{Init}(\mathsf{pwd}_{i,j}, P_i, P_j)$: this algorithm generates a uniformly random x in $\mathbb{Z}_p$ and computes $X = g^x$ and $V = \mathsf{H}(\mathsf{pwd}_{j,i})$. Then, it outputs $I = X \cdot V$ and $\mathsf{state} = (x, I, P_i, P_j, V)$.
- $\mathsf{RspDer}(\mathsf{pwd}_{i,j}, P_i, P_j, I)$: this algorithm generates a random $y \in \mathbb{Z}_p$ and computes $Y = g^y$ and $V = \mathsf{H}(\mathsf{pwd}_{j,i})$. Then, it outputs $R = Y \cdot V$. Finally, it computes $X' = I/V$, $Z = (X')^y$ and outputs $\mathsf{K} = \mathsf{H}'(Z, I, R, P_i, P_j, V)$.
- $\mathsf{Der}(\mathsf{pwd}_{i,j}, \mathsf{state}, R)$: this algorithm parses the input state as $\mathsf{state} = (x, I, P_i, P_j, V)$. Next, it computes $Y' = R/V$ and $Z = (Y')^x$ and outputs the session key K as $\mathsf{K} = \mathsf{H}'(Z, I, R, P_i, P_j, V)$.

Reducing the Reaction Time. In Figure 2, we can further optimize the reaction time by letting the initiator compute V^{-1} when she waits for the response. Key derivation will then only require a simple group operation.

Concurrent Variant of Protoss.
Observe that I and R can be precomputed for any pair of parties P_i, P_j. Since R is independent of I the messages, could also be sent concurrently if necessary, allowing for much shorter reaction times. To this end, we simply would have P_j call $\mathsf{Init}(\mathsf{pwd}_{j,i}, P_j, P_i)$ to compute R and after receiving I, compute the key via $\mathsf{Der}(\mathsf{pwd}_{j,i}, \mathsf{state}, I)$. (The key derivation $\mathsf{K} = \mathsf{H}'((X')^y, I, R, P_i, P_j, V)$ must additionally be adapted to not have the inputs I, R, P_i, P_j in chronological but lexicographic order.)

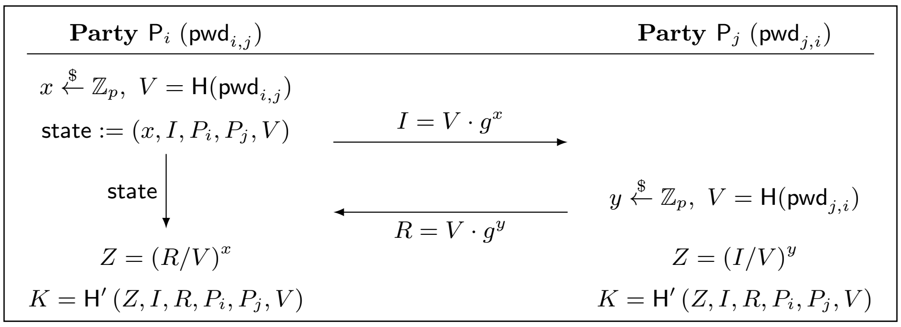

Party P_i $(\mathsf{pwd}_{i,j})$
Party P_j $(\mathsf{pwd}_{j,i})$
$x \xleftarrow{\$} \mathbb{Z}_p,\ V = \mathsf{H}(\mathsf{pwd}_{i,j})$
$\mathsf{state} := (x, I, P_i, P_j, V)$
$I = V \cdot g^x$
state
$y \xleftarrow{\$} \mathbb{Z}_p,\ V = \mathsf{H}(\mathsf{pwd}_{j,i})$
$R = V \cdot g^y$
$Z = (R/V)^x$
$Z = (I/V)^y$
$K = \mathsf{H}'(Z, I, R, P_i, P_j, V)$
$K = \mathsf{H}'(Z, I, R, P_i, P_j, V)$

Fig. 2: The Protoss Protocol.

Let us now state our main result.

Theorem 1. *Assume we have ℓ parties and at most n oracles per party. Assume the attacker makes q_{SendInit} SendInit-queries, $q_{\mathsf{SendRspDer}}$ SendRspDer-queries, q_{SendDer} SendDer-queries, q_{H} hash queries to $\mathsf{H} : \{0,1\}^* \to G$, and $q_{\mathsf{H}'}$ queries to $\mathsf{H}' : \{0,1\}^* \to \mathcal{K}$. Also, assume that we work in a group G of prime order p. Assume that $\log_2(p) = \mathsf{poly}(\kappa)$ and $\log_2(|\mathcal{K}|) = \mathsf{poly}(\kappa)$. For any attacker A that (t, ϵ)-breaks the Protoss-protocol, there is an attacker R that can break the security of the SSQRDH assumption in time t' with probability ϵ' where*

$$t' \approx t$$

and

$$\epsilon' \geq \epsilon - \left(2 + (\ell n)^2 + (q_{\mathsf{H}})^2\right)/p - (q_{\mathsf{H}'})^2/|\mathcal{K}|.$$

So, under the (t', ϵ')-SSQRDH assumption no attacker can exist that (t, ϵ)-breaks Protoss. Moreover, the proof is independent of the dictionary size $|D|$. (The inevitable loss due to guessing in online attacks is already addressed in the security definition. The definition is specifically interested in the advantage of an attacker over simple guessing.)

5 Security

Before we begin, let us describe our proof strategy intuitively.

5.1 Overview

We make heavy use of the programmability of the two random oracles. Observe that the messages I and R bear no information on the passwords whatsoever. In fact, they perfectly blind the passwords. So the only information an attacker might get on the passwords (without resulting into a trivial attack), is via the key that is derived in the final step. Now, in the centre of our strategy we will make the reduction generally output random values for each input to the random oracle H'. The only exception is when H' is queried by inputs that correspond to the correct inputs that are, by at least one oracle, used to compute the session key. In this case, we additionally need to make sure that the output is equal to the output of the $\mathsf{RevealKey}$ query for that oracle. Otherwise, the output can be truly random and, importantly, independent of all the remaining information sent. We will use the SSQRDH oracle to recognize these cases. This strategy already guarantees that an attacker can only obtain additional information by making at least one hash query (consisting of a set of correct inputs). However, to thwart offline-attacks this is not enough, since the attacker could try – via repeated hash queries – to gather information on several passwords. Let us go into more detail: consider an attacker that can solve the CDH problem. Then this attacker could easily compute $\mathsf{CDH}\left(g, I/\mathsf{H}(\mathsf{pwd}), R/\mathsf{H}(\mathsf{pwd})\right) = Z_{\mathsf{pwd}}$ for each password $\mathsf{pwd} \in D$ in an offline attack. Assume that after a $\mathsf{RevealKey}$ query this attacker has obtained key K. It can now ask $\mathsf{H}'(Z_{\mathsf{pwd}}, I, R, P_i, P_j, \mathsf{H}(\mathsf{pwd})) =: K_{\mathsf{pwd}}$ repeatedly for all password in D to the random oracle H'. Observe that if for the real key it holds that $K \neq K_{\mathsf{pwd}}$, the password pwd is wrong. This decreases the set of candidate passwords shared between P_i and P_j by one. Moreover, if $K = K_{\mathsf{pwd}}$ the password is correct. In this way, the attacker could find a password via an offline attack from a single protocol run between P_i and P_j. It can now use its knowledge of the password to correctly answer the Test query for some other session. To thwart this attack, we intuitively show that such a strategy is computationally impossible unless the attacker breaks our security assumption. In more detail, we show that if the attacker is able to send two such values Z_{pwd} and $Z_{\mathsf{pwd}'}$ for $\mathsf{pwd} \neq \mathsf{pwd}'$ to the random oracle – we call this a *critical* event – then we can break the SSQRDH assumption. Moreover, critical events are recognizable by the reduction via the SSQRDH oracle. So this means that, in absence of a critical event for any oracle, the attacker can verify (confirm or exclude) only a single password per session. This gives us the important additive contribution of $(q_{\mathsf{RspDer}} + q_{\mathsf{Der}})/|D|$ in the security theorem which captures that password tests can only be made via online-attacks where the adversary sends a message to some party. The final argument shows that if the attacker is able to distinguish the real key from a random key, it must have asked Z to the random oracle. However, this directly breaks the SSQRDH assumption.
To achieve a tight security proof, we avoid partitioning arguments that rely on guessing which sessions will be either tested (i.e the attacker obtains a key that is either random or real and the attacker has to decide which case it is) or revealed (i.e. the attacker asks to see the session key at some point). Typically, security proofs aim to follow such a partitioning argument to embed the complexity challenge into the tested sessions while behaving largely honestly in the revealed sessions. Such a guessing step would still allow for a polynomial-time security reduction but would account for a security loss in the number of sessions. Our strategy to obtain a tight security proof insists on embedding the complexity challenge into *all* messages, heavily relying on the random self-reducibility of the SSQRDH assumption. Similarly, to avoid partitioning arguments when dealing with $\mathsf{Corrupt}$-queries, we set up the reduction so that it knows *all* passwords.
Let us become more formal.

5.2 Proof of Theorem 1

As usual the proof is by contradiction developing a reduction algorithm R, which plays the role of the challenger/execution environment.

Game 0: This is the original security game. By definition, the attacker runs in time t with success probability
$$\epsilon_0 = \epsilon + (q_{\mathsf{RspDer}} + q_{\mathsf{Der}})/|D|.$$

Game 1: The reduction now aborts if at some point two distinct oracles produce the same message (I or R). Since there are only ℓn oracles in total we get

$$\epsilon_1 \geq \epsilon_0 - (\ell \cdot n)^2/p.$$

Game 2: The reduction now aborts if at some point two distinct inputs will be asked to H' that collide. We get

$$\epsilon_2 \geq \epsilon_1 - (q_{\mathsf{H}'})^2/|\mathcal{K}|.$$

Game 3: We now change how the reduction simulates the setup parameters. Assume, the reduction chooses a single random exponent $x' \xleftarrow{\$} \mathbb{Z}_p$ but $x' \neq 0$ to create a random group element $X' = g^{x'} \in G$. It can now define each output of the random oracle as a power of X'. For every query to the random oracle H the reduction draws a random $u \xleftarrow{\$} \mathbb{Z}_p$ and outputs $(X')^u$. In particular, for each pair of parties (P_i, P_j) with $i \neq j$ the reduction draws $u_{i,j} \xleftarrow{\$} \mathbb{Z}_p$ and programs the random oracle H to map to $\mathsf{H}(\mathsf{pwd}_{i,j}) = (X')^{u_{i,j}} = V_{i,j}$. As this change is only conceptual, we get

$$\epsilon_3 \geq \epsilon_2.$$

Game 4: The reduction now aborts if at some point two distinct inputs will be asked to H that collide. We get

$$\epsilon_4 \geq \epsilon_3 - (q_{\mathsf{H}})^2/p.$$

Game 5: In this game, the reduction generates the ephemeral keys produced by the oracles differently. Instead of drawing x random when computing $m = \mathsf{H}(\mathsf{pwd}) \cdot g^x$ the reduction draws $r_{i,j,s} \xleftarrow{\$} \mathbb{Z}_q$ for all parties $i, j \in [\ell]$ and all sessions $s \in [d]$ and computes

$$W_{i,j,s} = \left(X'\right)^{r_{i,j,s}}.$$

Now, each message will be computed as $m = \mathsf{H}(\mathsf{pwd}) \cdot W_{i,j,s}$ Thus, the ephemeral keys are implicitly set to $w_{i,j,s} = x' \cdot r_{i,j,s}$. This change is only conceptual and we get

$$\epsilon_5 \geq \epsilon_4.$$

Game 6: In this game, the reduction aborts whenever event $E1$ or event $E2$ happens. In the event $E1$ we have that the attacker makes two distinct random oracle queries Z, I, R, P_i, P_j, V and Z', I, R, P_i, P_j, V' to H' such that we have

1) there exists an oracle π_i^s that has accepted with transcript I, R while using password $\mathsf{pwd}_{i,j}$ (or $V = \mathsf{H}(\mathsf{pwd}_{i,j})$) to generate messages and derive the session key. (Observe that since we have excluded collisions in H we have that for any V there is a unique $\mathsf{pwd}_{i,j}$ with $V = \mathsf{H}(\mathsf{pwd}_{i,j})$.)
2) we have that the two equations hold:

$$\mathsf{DDH}(g, I/V, R/V, Z) = 1$$

and

$$\mathsf{DDH}(g, I/V', R/V', Z') = 1$$

hold for any other password $V' = \mathsf{H}(\mathsf{pwd}_{i,j'})$ with $(i, j') \neq (i, j)$.

In the event $E2$ we have that for at least one oracle π_i^s

1) there exists another oracle π_j^t that is partnered with π_i^s, and $\mathsf{T}_i^s = \mathsf{T}_j^t = (I, R)$ is the transcript of the protocol run between them and
2) the attacker has queried the random oracle on $\mathsf{H}'(Z, I, R, P_i, P_j, V)$ where $V = \mathsf{H}(\mathsf{pwd}_{i,j})$ and $\mathsf{DDH}(g, I/V, R/V, Z) = 1$.

Let us now bound the probability that

$$E = E1 \vee E2$$

will happen. To this end, we show that in the event E the reduction can break the SSQRDH assumption. In the following, let $X = g^x$ be the challenge received by the SSQRDH challenger and let $\mathcal{O}_x$ be the SSQRDH oracle that the reduction can access. The reduction now implicitly sets $X' = X$. This has major consequences for the simulation. Observe that the reduction does neither know x' nor the ephemeral keys. To nevertheless simulate correctly, it can use the SSQRDH oracle.

Simulation
Specifically, the reduction will use the following basic two-step strategy: each time the attacker asks a RevealKey-query the reduction responds with a truly random value in $\mathcal{K}$. Similarly, each time the attacker asks a H'-query, the reduction responds with a truly random value in $\mathcal{K}$. However, there is a situation in which the reduction needs to deviate from this strategy and outputs a value that it has output before to make sure the outputs of RevealKey and H' are equal. This happens if the attacker makes a query to H' of the form Z, I, R, P_i, P_j, V with $V = \mathsf{H}(\mathsf{pwd}_{i,j})$ and there exists at least one oracle π_i^s that has accepted with transcript I, R and $\mathsf{Pid}_i^s = j$ while

$$\mathsf{DDH}(g, I/V, R/V, Z) = 1.$$

In this case, we must have that $\mathsf{RevealKey}(\pi_i^s) = \mathsf{H}'(Z, I, R, P_i, P_j, V)$ to keep the simulation consistent. We also call such a situation a *consistency event* for π_i^s. Fortunately, the reduction can recognize these cases with the help of $\mathcal{O}_x$. Let us go into more detail. Let us assume that π_i^s is an initiator oracle. (The case when it is a responder oracle is analogous.) By setup we then have that

$$I/V = X^{r_{i,j,s} - u_{i,j}}.$$

Now the reduction can recognize a consistency event via the oracle query

$$\mathcal{O}_x(R/V, Z^{r_{i,j,s} - u_{i,j}}) = \mathsf{DDH}(g, I/V, R/V, Z).$$

Only if the result is 1, the reduction has encountered a consistency event. In this way, the reduction can make the simulation consistent.

Extraction in Case E2
Now let us show how we can extract a solution to the SSQRDH problem in the event $E2$. Recall that in this case we have that 1) there is an oracle π_i^s that also has a partner oracle π_j^t and $\mathsf{T}_i^s = \mathsf{T}_j^t = (I, R)$ is the transcript of their protocol run, and 2) the attacker has queried the random oracle on $\mathsf{H}'(Z, I, R, P_i, P_j, V)$ with $V = \mathsf{H}(\mathsf{pwd}_{i,j})$ and we have $\mathsf{DDH}(g, I/V, R/V, Z) = 1$ for some password pwd. However, since

$$R/V = X^{r_{j,i,t} - u_{i,j}} \wedge I/V = X^{r_{i,j,s} - u_{i,j}}$$

we must have that

$$Z = g^{x^2 (r_{j,i,t} - u_{i,j})(r_{i,j,s} - u_{i,j})}$$

from which we can easily compute

$$g^{x^2} = Z^{1/((r_{j,i,t} - u_{i,j})(r_{i,j,s} - u_{i,j}))}.$$

This does not work if

$$(r_{j,i,t} - u_{i,j})(r_{i,j,s} - u_{i,j}) = 0$$

but since the $w_{i,j,s}, u_{i,j}$ are drawn uniformly random, the probability for such a failure is only $2/|G|$.

Extraction in Case E1
Let us now show how to extract a solution in the event E1. Recall that in this event the attacker makes two random oracle queries Z, I, R, P_i, P_j, V and Z', I, R, P_i, P_j, V' to H' with $V = \mathsf{H}(\mathsf{pwd}_{i,j})$ and $V' = \mathsf{H}(\mathsf{pwd}_{i,j'})$ for $(i, j') \neq (i, j)$ such that we have
- there exists one oracle π_i^s that has accepted with transcript I, R.
- we have that the equations

$$\mathsf{DDH}(g, I/V, R/V, Z) = 1 \wedge \mathsf{DDH}(g, I/V', R/V', Z') = 1$$

  hold for two distinct passwords $\mathsf{pwd}_{i,j} \neq \mathsf{pwd}_{i,j'}$.

Let us again assume that π_i^s is an initiator oracle. In case it is a responder oracle, the arguments are analogous. If π_i^s is an initiator oracle, we have that

$$I/\mathsf{H}(\mathsf{pwd}_{i,j}) = X^{r_{i,j,s} - u_{i,j}} \wedge I/\mathsf{H}(\mathsf{pwd}_{i,j'}) = X^{r_{i,j,s} - u_{i,j'}}.$$

Moreover, we have that

$$\mathsf{H}(\mathsf{pwd}_{i,j'})/\mathsf{H}(\mathsf{pwd}_{i,j}) = X^{u_{i,j'}-u_{i,j}}.$$

At the same time, it holds that

$$Z^{1/(r_{i,j,s}-u_{i,j})} = \mathsf{CDH}(g, X, R/\mathsf{H}(\mathsf{pwd}_{i,j}))$$

and

$$Z'^{1/(r_{i,j',s}-u_{i,j'})} = \mathsf{CDH}(g, X, R/\mathsf{H}(\mathsf{pwd}_{i,j'})).$$

Let us now consider

$$\begin{aligned} &Z^{1/(r_{i,j,s}-u_{i,j})}/Z'^{1/(r_{i,j',s}-u_{i,j'})} \\ &= \mathsf{CDH}(g, X,\ (R/\mathsf{H}(\mathsf{pwd}_{i,j}))/(R/\mathsf{H}(\mathsf{pwd}_{i,j'}))\ ) \end{aligned}$$

which implies

$$Z^{1/(r_{i,j,s}-u_{i,j})}/Z'^{1/(r_{i,j',s}-u_{i,j'})} = \mathsf{CDH}(g, X,\ X^{u_{i,j'}-u_{i,j}}\ ).$$

This shows that

$$g^{x^2} = \left(Z^{1/(r_{i,j,s}-u_{i,j})}/Z'^{1/(r_{i,j',s}-u_{i,j'})}\right)^{1/(u_{i,j'}-u_{i,j})}$$

gives a solution to the SSQRDH problem unless

$$u_{i,j'} - u_{i,j} = 0,\ r_{i,j',s} - u_{i,j'} = 0,\ \text{or } (r_{i,j,s} - u_{i,j}) = 0.$$

However, each of these events only happens with statistically small probability $1/|G|$.
The reduction can easily identify the two queries that exist by assumption in this case. To this end it sends each random oracle query that has been sent to the H'-oracle to the DDH oracle by evaluating

$$\mathcal{O}_x(R/V, Z^{r_{i,j,s}-u_{i,j}})$$

(when appropriately parsed). The result of this evaluation is stored. Now, for any two queries Z, I, R, P_i, P_j, V and Z', I, R, P_i, P_j, V' where the result of this oracle call is positive while we have at the same time that both $V = \mathsf{H}(\mathsf{pwd}_{i,j})$ and $V' = \mathsf{H}(\mathsf{pwd}_{i',j'})$ for some distinct passwords $\mathsf{pwd}_{i,j}, \mathsf{pwd}_{i',j'}$ can be used by the reduction to break the security assumption.
To sum up, in either event E1 or E2 the reduction can simulate the oracles correctly and also extract a solution to the SSQRDH assumption. Taking the worst case of both events,

$$\epsilon_6 \geq \epsilon_5 - \epsilon' - 2/p.$$

Final Analysis. The remaining event we have to analyse is

$$\bar{E} = \bar{E}1 \wedge \bar{E}2.$$

Recall that in the event $\bar{E}2$, we have that for every oracle π_i^s it holds that either 1) π_i^s has no partner oracle or 2) π_i^s has a partner oracle π_j^t with common transcript $\mathsf{T}_i^s = \mathsf{T}_j^t = (I, R)$ but the attacker has never queried the random oracle on $\mathsf{H}'(Z, I, R, P_i, P_j)$ such that $\mathsf{DDH}(g, I/\mathsf{H}(\mathsf{pwd}), R/\mathsf{H}(\mathsf{pwd}), Z) = 1$ for some password pwd. Observe that in case of 2) it is easy to see that, because of the random oracle, the attacker gains no additional information on the session key of π_i^s. Crucially observe that messages I and R do not bear any information on passwords since the ephemeral keys are random group elements and thus act as one-time keys in a one-time pad. Moreover, since there are no consistency events for π_i^s, queries made to H' do not bear any information on the passwords and essentially every output is an independently drawn random value in the keyspace. So, the attacker simply passively observes a protocol run without gaining any information on the session key (since it does not query the random oracle on meaningful inputs in the sense of $\mathsf{DDH}(g, I/\mathsf{H}(\mathsf{pwd}), R/\mathsf{H}(\mathsf{pwd}), Z) = 1$).
In case 1) we must have that one of the messages has been generated by the attacker. Let us again assume π_i^s is an initiator oracle (and R has been generated by the attacker). Again, in case it is a responder oracle the arguments are analogous. With the previous game we have also guaranteed that for any π_i^s with transcript I, R and $\mathsf{Pid}_i^s = j$ there has only been at most a single query Z, I, R, P_i, P_j such that

$$\mathcal{O}_x(R/\mathsf{H}(\mathsf{pwd}_{i,j}), Z^{r_{i,j,s}-u_{i,j}}) = 1.$$

Furthermore, observe that for an attacker to be successful, we must have that $\mathsf{Corrupt}(i, j)$ has only been asked to π_i^s after it accepted, so R has been computed independently of this query.
The only values that the attacker can obtain that depend on the password is the key output by $\mathcal{K}_i^s = \mathsf{RevealKey}(\pi_i^s)$. However, the key has in fact been chosen truly random and otherwise bears no structural information. So, the only way for the attacker to test for candidate passwords is to ask H' queries and check whether they equal the output of $\mathsf{RevealKey}(\pi_i^s)$. Now observe that we have guaranteed that for each oracle π_i^s with $\mathsf{Pid}_i^s = j$ and transcript I, R there is only one input Z, I, R, P_i, P_j to H' with

$$\mathsf{DDH}(g, I/\mathsf{H}(\mathsf{pwd}_{i,j}), R/\mathsf{H}(\mathsf{pwd}_{i,j}), Z) = 1.$$

Now, any test that the attacker can make for a password must consist of such a query, as any other input to H' can never result in an output that equals $\mathcal{K}_i^s$ (since we have excluded collisions in H') – independent of whether the computations involve the correct password or not. So 1) since the attacker can only gain information on the password via these special Z, I, R, P_i, P_j queries to H' and 2) since there is only a single such query per oracle π_i^s and 3) because each query can only be used to test at most a single password (since the query itself only depends on a single password), we get that in total the attacker can only test a single password for any message that it sent to an oracle message. Since the overall number of messages that the attacker can send to oracles is at most $q_{\mathsf{SendRspDer}} + q_{\mathsf{SendDer}}$ and since we have now exhaustively covered all the events E and $\bar{E}$ we get that

$$\epsilon_6 = (q_{\mathsf{SendRspDer}} + q_{\mathsf{SendDer}})/|D|, \text{ and thus}$$

$$\begin{aligned}&(q_{\mathsf{SendRspDer}} + q_{\mathsf{SendDer}})/|D| \\ &\geq \epsilon + (q_{\mathsf{SendRspDer}} + q_{\mathsf{SendDer}})/|D| - (\ell \cdot n)^2/p - (q_{\mathsf{H}})^2/p \\ &\quad -(q_{\mathsf{H}'})^2/|\mathcal{K}| - \epsilon' - 2/p\end{aligned}$$

which concludes the proof. Overall the reduction makes at most $O(q_{\mathsf{H}'})$ queries to the DDH-oracle.

6 Security Loss

Our main theorem shows that the security loss $L = t'/\epsilon' \cdot \epsilon/t$, does not lose a multiplicative factor. Indeed ϵ' is statistically close to ϵ. A crude bound is $\epsilon' \geq \epsilon/2$. Since $t' \approx t$ we so obtain that the security reduction is tight. To better explain the nature of our reduction and the consequences for parameter sizes we argue similar to (10) (Appendix A). When instantiating parameters to achieve k-bit security in the sense that any successful attacker must have runtime $t \geq 2^k$ and advantage $2^{-k} \geq \epsilon$ we can now setup the system parameters such that any R has success probability at most $2^{-k-1} \geq \epsilon'$ when running in time at most $t' \geq 2^k$. In this way, the reduction only loses one bit of security and it is simple to compensate for that with an appropriate parameter choice.

$$2^{-k-1} \geq \epsilon' \geq \epsilon/2 \quad \Rightarrow \quad 2^{-k} \geq \epsilon.$$

6.1 Tight Composability

Via the choice of our security model we believe that direct compositions of Protoss and (tightly secure) symmetric primitives (without a prior key confirmation step) can immediately be shown with a security loss that does not exceed that of Protoss. The reasons for that are that we 1) consider several Test queries and 2) a single, global bit b that decides whether we are in the real or random case. The authors in (43; 32) call this multi-challenge single-bit security and show that this notion can guarantee that we can avoid hybrid arguments when we combine symmetric primitives with such a key exchange protocol. Roughly, the intuition is that all the (challenge) keys of all the symmetric primitives can be exchanged for random keys at once without the attacker noticing. In our reduction, this step does only account for a constant loss in security. Classical techniques, however, often need to apply a hybrid argument to exchange the keys one by one that incurs a considerable security loss in the number of instances that need to be exchanged.

Protocol	$\lvert I\rvert+\lvert R\rvert$	Msg.	Key Der.	Overall	Tight Proof?	Assumpt.	Trusted Setup?
TBPEKE (38) (Theorem 2)	320	2 FB	1 VB	2 FB + 1 VB $= (\alpha\beta + 2\alpha)$ HG	n (GB)	GSDH ROM	y
EKE2 (12) (Theorem 1)	320	1 E + 1 FB	1 D + 1 VB	1 E + 1 D + 1 FB +1 VB $= (\alpha\beta + \alpha + \gamma)$ HG	n (GB)	CDH Ideal Cipher	n
SPEKE (30)	320	1 HG + 1 VB	1 VB	1 HG + 2 VB $= (2\alpha\beta + 1)$ HG	–	–	n
SPAKE1 (7) (Theorem 4.1)	320	2 FB	1 FB + 1 VB	3 FB + 1 VB $= (\alpha\beta + 3\alpha)$ HG	n (GB)	CDH ROM	y
SPAKE2 (7) (Theorem 5.1)	320	2 FB	1 FB + 1 VB	3 FB + 1 VB $= (\alpha\beta + 3\alpha)$ HG	n (GB)	CDH ROM	y
CPace (6) (Theorem 5.1)	320	1 HG + 1 VB	1 VB	1 HG + 2 VB $= (2\alpha\beta + 1)$ HG	n (UC)	sSDH ROM	n
PPK (20)	320	2 HG + 1 FB	1 VB	2 HG + 1 FB + 1 VB $= (\alpha\beta + \alpha + 2)$ HG	n (SB)	DDH ROM	n
Protoss (this work)	320	1 HG + 1 FB	1 VB	1 HG + 1 FB + 1 VB $= (\alpha\beta + \alpha + 1)$ HG	y (GB)	SSQRDH ROM	n

Table 2: Efficiency of Symmetric Two-Move PAKE Protocols with Optimal Communication Complexity. We count hash-to-group operations (HG), fixed-base exponentiations (FB), variable-base exponentiations (VB). symmetric encryptions (E), and symmetric decryptions (D). We assume that neither passwords nor their hashes will be stored on computing devices after a protocol run. We assume 1 FB = α HG, 1 VB= β FB, and 1 E +1 D=γ HG. The entry '–' indicates that no formal proof exists. GSDH indicates the Gap Simultaneous Diffie-Hellman Assumption while sSDH stands for the Strong Simultaneous CDH assumption. GB indicates proofs in game-based security models, SB indicates proofs in simulation-based security model, while UC refers to proofs in the universal composability framework. The proof for CPace is tight for a single session only. Invoking the composition theorem of the UC framework will introduce a hybrid argument and a loss in the number of sessions.

6.2 Performance Analysis

Assume we implement Protoss in groups over elliptic curves with prime order p such that $\log_2(p) = 160$ and group element representation of around 160 bits. To compute a message $m = \mathsf{H}(\mathsf{pwd}) \cdot g^x = V \cdot g^x$ we, essentially require 1 hash-to-group (HG) operation to compute $V = \mathsf{H}(\mathsf{pwd})$ and one fixed-base exponentiation (FB) g^x. The cost for the key derivation $(m/V)^y$ essentially accounts for a variable-base exponentiation (VB). For more generality, let us assume that the costs are related such that 1 FB=α HG, 1 VB=β FB=$\alpha\beta$ HG, and 1 E + 1 D=γ HG. Table 2 shows that our protocol is computationally more efficient than the existing ROM-based ones with optimal message size in case two assumptions hold. These assumptions are i) that variable-base exponentiations are computationally more expensive than fixed-base exponentiations ($\beta \geq 1$) and that ii) the hash-to-group operation is more efficient than fixed-base exponentiation ($\alpha \geq 1$). In practice these assumptions are generally fulfilled, the first because of the computational advantage of using pre-computed public ladders for fixed-base exponentiations as detailed in the introduction, and the second because hashing is generally more efficient than exponentiation in groups (19).
The table also counts the costs of the general EKE2 protocol when it's based on symmetric encryption ciphers. The general EKE2 protocol requires one symmetric encryption and one decryption. For dedicated ciphers these operations usually have comparable computational costs. We stress that, the decryption operation must injectively map to the underlying group that we work in, thus standard symmetric ciphers like AES cannot be used. This requirement seems generally more challenging (or equivalent) than what we require from our hash-to-group operation which we allow to be compressing. As compared to the general EKE2 protocol, Protoss can also make use of an important synergy between message generation and key derivation as compared to the general EKE2 protocol. This synergy revolves around the fact that we do not need to compute with the password twice. Instead, we hash it once and obtain the value V that can be used in message generation and key derivation as well. Importantly, once we have V each computation then only accounts for a single multiplication only. This efficiency improvement does not generically apply to ciphers where encryption and decryption do not share state information. We thus assume that in general 1 E + 1 D $\geq$ 1 HG, and $\gamma \geq 1$. We also note that there is a crucial conceptual difference between Protoss and the traditional EKE approach. The cipher in the EKE encrypts the ephemeral key and so guarantees that it is confidential. The key used in this encryption routine is the password. Since this encryption operation is applied several times, each time a pair of communication partners exchange messages anew, we require relative strong security properties of the cipher (multi-use security). Our approach however, can be thought of as reversing this relation. We encrypt the hashed password with a fresh one-time key (the ephemeral public key) in a statistically secure way. Since we only work with a one-time pad the security requirements are in general less demanding than for multi-use ciphers and constructions are simpler. Our encryption guarantees that over the several runs of the protocol the messages do not reveal any information on the hashed password. Conversely, under the assumption that the hashed password remains secret, all ephemeral keys remain secret as well. Our proof shows that there are no algebraic attacks.
Finally, we note that although the ideal cipher model is known to imply the random oracle model, we have no conclusive evidence on whether the other direction is true (12). In this sense, relying on ideal ciphers for the proof seems like a stronger assumption than relying on the random oracle model. To give some concrete values, we have experimentally computed $\alpha = 2.5$ and $\beta = 3.33$ on a standard laptop computer for Curve25519 using Elligator2 (Elligator2Hash) (19) (11th Gen Intel(R) Core(TM) i7-1165G7 @2.80GHz, Java, BouncyCastle).

Acknowledgements

Emanuele Di Giandomenico and Sven Schäge were supported by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme in the scope of the CONFIDENTIAL6G project under Grant Agreement 101096435. The contents of this publication are the sole responsibility of the authors and do not in any way reflect the views of the EU.

Bibliography

[1] Abdalla, M.: Password-based authenticated key exchange: An overview. In: Chow, S.S.M., Liu, J.K., Hui, L.C.K., Yiu, S. (eds.) Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8782, pp. 1–9. Springer (2014). https://doi.org/10.1007/978-3-319-12475-9_1, https://doi.org/10.1007/978-3-319-12475-9_1

[2] Abdalla, M., Barbosa, M.: Perfect forward security of SPAKE2. IACR Cryptol. ePrint Arch. p. 1194 (2019), https://eprint.iacr.org/2019/1194

[3] Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part I. Lecture Notes in Computer Science, vol. 12170, pp. 278–307. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56784-$2_1$0

[4] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) Topics in Cryptology – CT-RSA 2001. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 8–12, 2001). https://doi.org/10.1007/3-540-45353-$9_1$2

[5] Abdalla, M., Fouque, P., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3386, pp. 65–84. Springer (2005). https://doi.org/10.1007/978-3-540-30580-4_6, https://doi.org/10.1007/978-3-540-30580-4_6

[6] Abdalla, M., Haase, B., Hesse, J.: Security analysis of cpace. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13093, pp. 711–741. Springer (2021). https://doi.org/10.1007/978-3-030-92068-5_24, https://doi.org/10.1007/978-3-030-92068-5_24

[7] Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3376, pp. 191–208. Springer (2005). https://doi.org/10.1007/978-3-540-30574-3_14, https://doi.org/10.1007/978-3-540-30574-3_14

[8] Alwen, J., Blanchet, B., Hauck, E., Kiltz, E., Lipp, B., Riepel, D.: Analysing the HPKE standard. In: Canteaut, A., Standaert, F. (eds.) Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 87–116. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_4, https://doi.org/10.1007/978-3-030-77870-5_4

[9] Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015: 12th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 9014, pp. 629–658. Springer, Heidelberg, Germany, Warsaw, Poland (Mar 23–25, 2015). https://doi.org/10.1007/978-3-662-46494-$6_2$6

[10] Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 273–304. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-$5_1$0

[11] Bauer, B., Fuchsbauer, G., Loss, J.: A classification of computational assumptions in the algebraic group model. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 121–151.

Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_5, https://doi.org/10.1007/978-3-030-56880-1_5

[12] Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 139–155. Springer, Heidelberg, Germany, Bruges, Belgium (May 14–18, 2000). https://doi.org/10.1007/3-540-45539-$6_1$1

[13] Bellare, M., Ranjan, R., Riepel, D., Aldakheel, A.: The concrete security of two-party computation: Simple definitions, and tight proofs for PSI and oprfs. IACR Cryptol. ePrint Arch. p. 1476 (2024), https://eprint.iacr.org/2024/1476

[14] Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 773, pp. 232–249. Springer (1993). https://doi.org/10.1007/3-540-48329-2_21, https://doi.org/10.1007/3-540-48329-2_21

[15] Bellare, M., Shea, L.: Flexible password-based encryption: Securing cloud storage and provably resisting partitioning-oracle attacks. In: Rosulek, M. (ed.) Topics in Cryptology - CT-RSA 2023 - Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24-27, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13871, pp. 594–621. Springer (2023). https://doi.org/10.1007/978-3-031-30872-7_23, https://doi.org/10.1007/978-3-031-30872-7_23

[16] Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992). https://doi.org/10.1109/RISP.1992.213269

[17] Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 4-6, 1992. pp. 72–84. IEEE Computer Society (1992). https://doi.org/10.1109/RISP.1992.213269, https://doi.org/10.1109/RISP.1992.213269

[18] Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. IACR Cryptol. ePrint Arch. p. 833 (2013), http://eprint.iacr.org/2013/833

[19] Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013. pp. 967–980. ACM (2013). https://doi.org/10.1145/2508859.2516734, https://doi.org/10.1145/2508859.2516734

[20] Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 156–171. Springer, Heidelberg, Germany, Bruges, Belgium (May 14–18, 2000). https://doi.org/10.1007/3-540-45539-$6_1$2

[21] Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 404–421. Springer, Heidelberg, Germany, Aarhus, Denmark (May 22–26, 2005). https://doi.org/10.1007/$11426639_2$4

[22] Cash, D., Kiltz, E., Shoup, V.: The twin diffie-hellman problem and applications. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4965, pp. 127–145. Springer (2008). https://doi.org/10.1007/978-3-540-78967-3_8, https://doi.org/10.1007/978-3-540-78967-3_8

[23] Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 767–797. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26954-$8_2$5

[24] Diemert, D., Gellert, K., Jager, T., Lyu, L.: More efficient digital signatures with tight multi-user security. In: Garay, J. (ed.) PKC 2021: 24th International Conference on

Theory and Practice of Public Key Cryptography, Part II. Lecture Notes in Computer Science, vol. 12711, pp. 1–31. Springer, Heidelberg, Germany, Virtual Event (May 10–13, 2021). https://doi.org/10.1007/978-3-030-75248-4$_1$

[25] Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 129–147. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40084-1$_8$

[26] Gennaro, R., Krawczyk, H., Rabin, T.: Okamoto-tanaka revisited: Fully authenticated diffie-hellman with minimal overhead. In: Zhou, J., Yung, M. (eds.) Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6123, pp. 309–328 (2010). https://doi.org/10.1007/978-3-642-13708-2_19, `https://doi.org/10.1007/978-3-642-13708-2_19`

[27] Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) Advances in Cryptology – CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 142–159. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2006). https://doi.org/10.1007/11818175$_9$

[28] Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 95–125. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0$_4$

[29] Hao, F., Ryan, P.Y.A.: Password authenticated key exchange by juggling. In: Christianson, B., Malcolm, J.A., Matyas, V., Roe, M. (eds.) Security Protocols XVI. pp. 159–171. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[30] Hao, F., Shahandashti, S.F.: The SPEKE protocol revisited. In: Chen, L., Mitchell, C.J. (eds.) Security Standardisation Research - First International Conference, SSR 2014, London, UK, December 16-17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8893, pp. 26–38. Springer (2014). https://doi.org/10.1007/978-3-319-14054-4_2, `https://doi.org/10.1007/978-3-319-14054-4_2`

[31] Jablon, D.P.: Strong password-only authenticated key exchange. Comput. Commun. Rev. **26**(5), 5–26 (1996). https://doi.org/10.1145/242896.242897, `https://doi.org/10.1145/242896.242897`

[32] Jager, T., Kiltz, E., Riepel, D., Schäge, S.: Tightly-secure authenticated key exchange, revisited. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 117–146. Springer, Heidelberg, Germany, Zagreb, Croatia (Oct 17–21, 2021). https://doi.org/10.1007/978-3-030-77870-5$_5$

[33] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 456–486. Springer (2018). https://doi.org/10.1007/978-3-319-78372-7_15, `https://doi.org/10.1007/978-3-319-78372-7_15`

[34] Kudla, C., Paterson, K.G.: Modular security proofs for key agreement protocols. In: Roy, B.K. (ed.) Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3788, pp. 549–565. Springer (2005). https://doi.org/10.1007/11593447_30, `https://doi.org/10.1007/11593447_30`

[35] Liu, X., Liu, S., Han, S., Gu, D.: Eke meets tight security in the universally composable framework. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography – PKC 2023. pp. 685–713. Springer Nature Switzerland, Cham (2023)

[36] Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996). https://doi.org/10.1201/9781439821916, `http://cacr.uwaterloo.ca/hac/`

[37] Pointcheval, D.: Password-based authenticated key exchange. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography - PKC 2012 - 15th Inter-

national Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7293, pp. 390–397. Springer (2012). https://doi.org/10.1007/978-3-642-30057-8_23, `https://doi.org/10.1007/978-3-642-30057-8_23`

[38] Pointcheval, D., Wang, G.: VTBPEKE: Verifier-based Two-Basis Password Exponential Key Exchange. In: ASIA CCS'17. Abu Dhabi, United Arab Emirates (Apr 2017). https://doi.org/10.1145/3052973.3053026, `https://hal.inria.fr/hal-01471737`

[39] Pointcheval, D., Wang, G.: VTBPEKE: verifier-based two-basis password exponential key exchange. In: Karri, R., Sinanoglu, O., Sadeghi, A., Yi, X. (eds.) Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017. pp. 301–312. ACM (2017). https://doi.org/10.1145/3052973.3053026, `https://doi.org/10.1145/3052973.3053026`

[40] Santos, B.F.D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13276, pp. 127–156. Springer (2022). https://doi.org/10.1007/978-3-031-07085-3_5, `https://doi.org/10.1007/978-3-031-07085-3_5`

[41] Schäge, S.: Tight security for signature schemes without random oracles. Journal of Cryptology **28**(3), 641–670 (Jul 2015). https://doi.org/10.1007/s00145-013-9173-6

[42] Schäge, S.: TOPAS: 2-pass key exchange with full perfect forward secrecy and optimal communication complexity. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 1224–1235. ACM (2015). https://doi.org/10.1145/2810103.2813683, `https://doi.org/10.1145/2810103.2813683`

[43] Skrobot, M., Lancrenon, J.: On composability of game-based password authenticated key exchange. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018. pp. 443–457. IEEE (2018). https://doi.org/10.1109/EuroSP.2018.00038, `https://doi.org/10.1109/EuroSP.2018.00038`

[44] Zhao, Z., Dong, Z., Wang, Y.: Security analysis of a password-based authentication protocol proposed to ieee 1363. Theoretical Computer Science **352**(1), 280–287 (2006). https://doi.org/https://doi.org/10.1016/j.tcs.2005.11.038, `https://www.sciencedirect.com/science/article/pii/S0304397505008844`

A UC-Proof of Protoss

We will now present a security proof in the UC framework. We rely on the multi-user, multi-session model presented by (35). In contrast to models like (6), it inherently accounts for several instances of the protocol run. The (6) model focusses on only a single session. Then it invokes the UC theorem to show security in a more dynamic setting with several parties and sessions. However, this is essentially a hybrid argument that will account for a security loss in the number of implicit applications of the UC-theorem. Though this number is polynomial, it still accounts for a tightness loss overall. We thus use the model by (35) which is specifically suited to keep track of the tightness loss in a dynamic setting.

A.1 Security Model

For better reference, we here recall the security model of (35). We take the model almost in verbatim from (35) adapted to our notation. Moreover, we use parties more generally instead of clients and servers. Intuitively we can think of clients and servers in (35) to correspond to initiators and servers in our setting. Similarly sessions π_i^s will be denoted as instances (P_i, iid_i). The original paper provides detailed intuition for the rest of the model.
Moreover, as emphasized in (35) this PAKE framework implicitly also deals with static corruptions of parties, i.e., the adversary can corrupt some parties and get their passwords before the protocol execution. Almost all UC frameworks (21; 27) for PAKE are defined in the way of static corruptions. Also as in (35) the corruption process is not explicitly modeled. We

specifically use the security model in (35) that models lazy extraction as introduced in (3). As indicated in (3), it seems necessary for protocols like SPEKE, SPAKE2, or TBPEKE in the UC model.

A.2 Theorem in the UC Framework

Theorem 2 (Protoss **is UC secure).** *If the* SSQRDH *assumption holds in G and if* H *and* H′ *are modelled as a random oracles, then* Protoss *securely emulates $\mathcal{F}_{\mathsf{lePAKE}}$. More precisely, for any PPT environment $\mathcal{Z}$ and real world adversary $\mathcal{A}$ which has access to random oracles* H *and* H′*, there exist a PPT simulator* Sim*, which has access to the ideal functionality $\mathcal{F}_{\mathsf{lePAKE}}$, and algorithm $\mathcal{B}$ s.t. the advantage of environment $\mathcal{Z}$ in distinguishing the real world running with $\mathcal{A}$ and the ideal world running with* Sim *is bounded by*

$$\mathsf{Adv}_{\mathsf{Protoss},\mathcal{Z}} \leq \mathsf{Adv}^{\mathsf{SSQRDH}}_{G,\mathcal{B}} + \frac{(q_{\mathsf{H}})^2 + (\ell \cdot n)^2}{p} + \frac{(q_{\mathsf{H}'})^2}{|\mathcal{K}|}$$

where q_{H} and $q_{\mathsf{H}'}$ denote the maximum number of random oracle queries to H *and* H′.

Proof. Fundamentally, the proof is very similar to our game-based proof. Observe that in our game-based proof, the first five game transitions are either conceptually only or they only introduce statistically low error terms to exclude collisions. In this proof, to not repeat the arguments, we intuitively move to this point immediately via the first game transition. This will be formally described in our first intermediate simulator Sim in Figure 5. This already uses programming of H. In the next game, we will transition to the next simulator Sim in Figure 6 that also programs the random oracle H′. Finally, we will show what happens when we introduce the ideal functionality $\mathcal{F}_{\mathsf{lePAKE}}$ in Figure 7. As in our game-based proof, we will exploit the analysis of events $E1$ and $E2$ to conclude that the two games are indistinguishable. In particular, this will ultimately show that no offline attack can be successful (event E1) and that the attacker is not able to compute non-trivial Z values while remaining passive (E2). As a consequence, we have that all session keys are truly random. Being able to exclude events $E1$ and $E2$ allows us to make the final game transition towards the ideal scenario, where keys are always drawn uniformly random via FreshKey queries. Also, since we now know that for any key exchange the attacker can perform only a single password check (since we exclude event E1), this can easily be emulated by the simulator via a TestPW query.
We use some of the conventions used in (35).

- Good/Bad party instance. We call a party instance (P_i, iid_i) a good (resp. bad) instance, if the password pwd used in this instance equals (resp. differs from) the correct password $\hat{\mathsf{pwd}}$ shared between P_i and its intended partner P_j.
- Linked instances. We say that an instance (P_j, iid_j) is linked to another instance (P_i, iid_i) if I generated by (P_i, iid_i) is received by one instance (P_j, iid_j) of its intended partner P_j. Similarly, we say that an instance (P_i, iid_i) is linked to another instance (P_j, iid_j) if R generated by (P_j, iid_j) is received by one instance (P_i, iid_i) of its intended partner P_i. If two instances are linked to each other, they are referred to as linked instances.

The central aim of the proof is to develop a PPT simulator Sim, which has access to $\mathcal{F}_{\mathsf{lePAKE}}$ and interacts with the environment $\mathcal{Z}$, and simulates the real world Protoss protocol interactions among the adversary $\mathcal{A}$, parties, and the environment $\mathcal{Z}$. With this goal in mind, Sim has to simulate honestly generated messages from real parties, respond to adversarial messages, and simulate the random oracles H and H′. The functionality $\mathcal{F}_{\mathsf{lePAKE}}$ provides information to Sim through interfaces including TestPW, NewInstance, FreshKey, CopyKey, CorruptKey, as defined in Fig. 3. Recall that Sim has no secret inputs (i.e. passwords).
The full description of Sim is given in Fig. 5. Let $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$ be the real experiment where environment $\mathcal{Z}$ interacts with real parties and adversary $\mathcal{A}$, and $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$ be the ideal experiment where $\mathcal{Z}$ interacts with simulator Sim. We prove that $|\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} \Rightarrow 1]|$ is negligible via a series of games **Game 0-3**, where **Game 0** is $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$, **Game 3** is equivalent to the ideal game $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$, and argue that two adjacent games are indistinguishable from $\mathcal{Z}$'s point of view.

Game 0. This is the real experiment $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$. In this experiment, Z initializes a password for each pair of party, observes the interactions among the parties and the adversary $\mathcal{A}$, and also obtains the corresponding session keys of protocol instances. Here $\mathcal{A}$ may implement attacks as view, modify, insert, or drop messages over the network. We have

$$\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] = \Pr[\mathbf{Game\ 0} \Rightarrow 1].$$

Game 1. (Move to **Game 5** of the Game-based Proof.) In **Game 1**, Sim (depicted in Figure 5) still needs to take passwords as inputs. With the help of passwords, it perfectly simulates the executions in $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$, except that the ROs H, H' are simulated in a collision-free way and H is programmed as $\mathsf{H}(\mathsf{pwd}_{i,j}) = (g^{x'})^{u_{i,j}}$. Moreover, ephemeral group values produced by instance iid_i are computed as $(g^{x'})^{w_{i,j,\mathsf{iid}_i}}$. Also, collisions among the messages are excluded. Meanwhile, Sim also necessarily keeps all these corresponding exponent values in $DL(g)$ and $DL(X')$.
We have

$$|\Pr[\mathbf{Game\ 1} \Rightarrow 1] - \Pr[\mathbf{Game\ 0} \Rightarrow 1]| \leq \frac{(q_{\mathsf{H}'})^2}{|\mathcal{K}|} + \frac{(q_{\mathsf{H}})^2 + (\ell \cdot n)^2}{p},$$

where q_{H} ($q_{\mathsf{H}'}$) denotes the maximum number of queries to H (H').

Game 2. We now exploit that the event $E1 \vee E2$ (as defined in the game-based security proof) will result in a break of the SSQRDH assumption. So the only remaining real attack behavior that the simulator should simulate is conditioned on the event $\bar{E}1 \wedge \bar{E}2$. We only concentrate on the changes to the previous game. Recall events $E1$ and $E2$ as defined in the game-based proof. These events are independent of queries of the security model except for queries to the random oracles H and H'.

$$|\Pr[\mathbf{Game\ 2} \Rightarrow 1] - \Pr[\mathbf{Game\ 1} \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{SSQRDH}}_{G,\mathcal{B}}.$$

Game 3. (Use $\mathcal{F}_{\mathsf{lePAKE}}$ interfaces.) In this game, depicted in Figure 7, we introduce the ideal functionality $\mathcal{F}_{\mathsf{lePAKE}}$ (Figure 3). By querying $\mathcal{F}_{\mathsf{lePAKE}}$, the simulator Sim can perfectly simulate **Game 2** via the queries $\mathsf{FreshKey}, \mathsf{CopyKey}, \mathsf{CorruptKey}, \mathsf{TestPW}, \mathsf{RegisterTest}$, and $\mathsf{LateTestPW}$.

$$\Pr[\mathbf{Game\ 3} \Rightarrow 1] = \Pr[\mathbf{Game\ 2} \Rightarrow 1].$$

Next, observe that the simulator in **Game 3** is indistinguishable from the experiment in the ideal world. Therefore, we have that $\Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} \Rightarrow 1] = \Pr[\mathbf{Game\ 3} \Rightarrow 1]$.

Functionality $\mathcal{F}_{\mathsf{lePAKE}}$
The functionality $\mathcal{F}_{\mathsf{lePAKE}}$ is parameterized by security parameter κ. It interacts with a simulator Sim and a set of parties via the queries described next. Moreover, it manages lists L, L', L'' that are initially empty.

- The list L holds entries of the format $(\mathsf{file}, P_i, P_j, \mathsf{pwd})$.
- Entries in L' have format $(P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, z, t)$ where we have that $z \in \{\mathsf{fresh}, \mathsf{completed}, \mathsf{compromised}, \mathsf{interrupted}\}$ and $t \in \{-, \mathsf{sid}\}$.
- Entries in L'' have format $(P_i, P_j, \mathsf{sid}, K)$.

Password Storage

Upon receiving a query $(\mathsf{StorePWFile}, P_i, P_j, \mathsf{pwd})$ from party P_i:
If there already exists some record $R = (\mathsf{file}, P_i, P_j, \mathsf{pwd}') \in L$ with some pwd', ignore this query. Otherwise, add $R_{i,j} = (\mathsf{file}, P_i, P_j, \mathsf{pwd})$ and $R_{j,i} = (\mathsf{file}, P_j, P_i, \mathsf{pwd})$ to L and send $(\mathsf{StorePWFile}, P_i, P_j)$ to Sim.

Sessions

Upon receiving a query $(\mathsf{NewInstance}, \mathsf{iid}_i, P_j, \mathsf{pwd})$ from P_i:
Retrieve the entry $R_{i,j} = (\mathsf{file}, P_i, P_j, \mathsf{pwd}')$ from L. Set $b = 1$ in case $\mathsf{pwd}' = \mathsf{pwd}$ and $b = 0$ otherwise. Send $(\mathsf{NewInstance}, P_i, \mathsf{iid}_i, P_j, b)$ to Sim.
Add $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{fresh}, -)$ to L'. We also say that P_j is the intended partner of (P_i, iid_i).

Upon receiving a query $(\mathsf{NewInstance}, \mathsf{iid}_i, P_j)$ from P_i:
Retrieve the entry $R_{i,j} = (\mathsf{file}, P_i, P_j, \mathsf{pwd})$ from L. Send $(\mathsf{NewInstance}, P_j, \mathsf{iid}_j, P_i)$ to Sim. Add $(P_j, \mathsf{iid}_j, P_i, \mathsf{pwd}, \mathsf{fresh}, -)$ to L'.

We also say that P_i is the intended partner of (P_j, iid_j) and that R' is (marked as) fresh. Two instances (P_i, iid_i) and (P_j, iid_j) are said to be partnered if there are two entries in L', $(P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, z, t)$ and $(P_j, \mathsf{iid}_j, P_i, \mathsf{pwd}', z', t')$, with $\mathsf{pwd} = \mathsf{pwd}'$.

Active Session Attacks

Upon receiving a query $(\mathsf{TestPW}, P, \mathsf{iid}, \mathsf{pwd}')$ **from** Sim:
If there is a fresh entry $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{fresh}, -)$ in L' for some $(P_i, \mathsf{iid}_i) = (P, \mathsf{iid})$ do the following:

- If $\mathsf{pwd}' = \mathsf{pwd}$ overwrite $R' := (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{compromised}, -)$ and reply to Sim with "correct guess". We also say that R' is compromised.
- If $\mathsf{pwd}' \neq \mathsf{pwd}$ overwrite $R' := (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{interrupted}, -)$ and reply to Sim with "wrong guess". We also say that R' is interrupted.

Upon receiving a query $(\mathsf{RegisterTest}, P, \mathsf{iid})$ **from** Sim: If there is a fresh entry $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{fresh}, -)$ in L' for some $(P_i, \mathsf{iid}_i) = (P, \mathsf{iid})$ then overwrite it to $R' := (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{interrupted}, -)$ mark it as $\mathsf{interrupted}$ and flag it tested.

Upon receiving a query $(\mathsf{LateTestPW}, P, \mathsf{iid}, \mathsf{pwd}')$ **from** Sim:
If there is a entry $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{interrupted}, \mathsf{sid})$ in L' for some $(P_i, \mathsf{iid}_i) = (P, \mathsf{iid})$ that is flagged tested and a corresponding entry $(P_i, P_j, \mathsf{sid}, K)$ in L'' remove the flag tested and do the following:

- If $\mathsf{pwd}' = \mathsf{pwd}$ output K
- If $\mathsf{pwd}' \neq \mathsf{pwd}$ output a random $K \in \mathcal{K}$

Fig. 3: The PAKE functionality $\mathcal{F}_{\mathsf{lePAKE}}$.

Session Key Generation

Upon receiving a query $(\mathsf{FreshKey}, P, \mathsf{iid}, \mathsf{sid})$ **from** Sim:
If 1) there is a fresh or interrupted entry $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, z, -)$ in L' with $z \in \{\mathsf{fresh}, \mathsf{interrupted}\}$ for some $i, j \in [n]$ such that $(P, \mathsf{iid}) = (P_i, \mathsf{iid}_i)$; and 2) there is no other entry $(P_i, P_j, \mathsf{sid}, K')$ in L'' with $\mathsf{iid}_i \neq \mathsf{iid}'_i$:

– pick a new random key $K \in \mathcal{K}$, overwrite $R' := (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{completed}, \mathsf{sid})$ in L', send $(\mathsf{iid}, \mathsf{sid}, K)$ to P_i, and add $R'' = (P_i, P_j, \mathsf{sid}, K)$ to L''.

Upon receiving a query $(\mathsf{CopyKey}, P, \mathsf{iid}, \mathsf{sid})$ **from** Sim:
If 1) there is a fresh record $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{fresh}, -)$ in L' and a completed record $\tilde{R}' = (P_j, \mathsf{iid}_j, P_i, \mathsf{pwd}, \mathsf{completed}, \mathsf{sid})$ s.t. (P_i, iid_i) and (P_j, iid_j) are partnered; and 2) there is no other entry $(P_i, P_j, \mathsf{sid}, K')$ in L'' with $\mathsf{iid}_i \neq \mathsf{iid}'_i$: and 3) there is exactly one entry $(P_j, P_i, \mathsf{sid}', K)$ in L'' with $\mathsf{sid}' = \mathsf{sid}$:

– Retrieve the entry $(P_j, P_i, \mathsf{sid}, K)$ from L'' and overwrite
$R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{completed}, \mathsf{sid})$ and send $(\mathsf{iid}, \mathsf{sid}, K)$ to P.

Upon receiving a query $(\mathsf{CorruptKey}, P, \mathsf{iid}, \mathsf{sid}, K)$ Sim:
If 1) there is a compromised record $R' = (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{compromised}, -)$ in L' with $(P_i, \mathsf{iid}_i) = (P, \mathsf{iid})$ for some i; and 2) there is no other entry $R'' = (P_i, P_j, \mathsf{sid}, K')$ in L'' with $\mathsf{iid}_i \neq \mathsf{iid}'_i$:

– overwrite $R' := (P_i, \mathsf{iid}_i, P_j, \mathsf{pwd}, \mathsf{completed}, \mathsf{sid})$ and send $(\mathsf{iid}, \mathsf{sid}, K)$ to P.

Fig. 4: Continuation of Fig. 3.

Sim maintains group element X' and sets L_{H}, $L_{\mathsf{H}'}$, T, $DL(g)$, $DL(X')$, CR that are initially empty in the simulation.

- Sim maintains internally a global group element X' that is computed as $g^{x'}$ for some uniformly random $x' \leftarrow \mathbb{Z}_p$.
- $L_{\mathsf{H}}, L_{\mathsf{H}'}$: store entries $(x, \mathsf{H}(x))$ and $(x, \mathsf{H}'(x))$ w.r.t random oracles H and H'.
- T: stores messages M sent by instances (P_i, iid_i) of parties in the format (P_i, iid_i, M) where M is typically denoted as I (initiator message) or R (responder message).
- $DL(g)$: stores pairs (X, x) of group elements and corresponding discrete logarithms with respect to basis g.
- $DL(X')$: stores pairs (X, x) of group elements and corresponding discrete logarithms with respect to basis X'.
- Q: stores hashed passwords $V_{i,j}$ shared between parties P_i, P_j in the format $(P_i, P_j, V_{i,j})$
- CR: stores triples (P_i, iid_i, b) where $b \in \{0, 1\}$ with $b = 1$ indicating that the provided password was correct.
- Similarly to (35), we also assume Sim has received all $(\mathsf{StorePWFile}, P_i, P_j)$ queries for all the pairs of parties P_i, P_j.

PAKE Sessions

on $(\mathsf{NewInstance}, P_i, \mathsf{iid}_i, P_j, b)$ from $\mathcal{F}_{\mathsf{lePAKE}}$:

- Draw $w_{i,j,\mathsf{iid}_i} \leftarrow \mathbb{Z}_p$ and compute $X = (X')^{w_{i,j,\mathsf{iid}_i}}$ and $I = XV_{i,j} = (X')^{(w_{i,j,\mathsf{iid}_i} + u_{i,j})}$ where we get $u_{i,j}$ via entries $(P_i, P_j, V_{i,j}) \in Q$ and $(V_{i,j}, u_{i,j}) \in DL(X')$. Set $T := T \cup \{(P_i, \mathsf{iid}_i, I)\}$ and send I from P_i to $\mathcal{A}$. Also add (P_i, iid_i, b) to CR, $(X, w_{i,j,\mathsf{iid}_i})$ to $DL(X')$.
- Add $(X, x' w_{i,j,\mathsf{iid}_i})$ to $DL(g)$.

on $(\mathsf{NewInstance}, P_j, \mathsf{iid}_j, P_i)$ from $\mathcal{F}_{\mathsf{lePAKE}}$ and I from $\mathcal{A}$ as a party message from P_i to (P_j, iid_j):

- Draw $w_{j,i,\mathsf{iid}_j} \leftarrow \mathbb{Z}_p$ and compute $X = (X')^{w_{j,i,\mathsf{iid}_j}}$ and $R = XV_{j,i} = (X')^{(w_{j,i,\mathsf{iid}_j} + u_{j,i})}$. Set $T := T \cup \{(P_i, \mathsf{iid}_i, R)\}$ and send R from P_j to $\mathcal{A}$. Also add (P_j, iid_j, b) to CR, $(X, w_{j,i,\mathsf{iid}_j})$ to $DL(X')$.
- Add $(X, x' w_{j,i,\mathsf{iid}_j})$ to $DL(g)$.
- Compute a new key as $K = \mathsf{H}'(Z, I, R, P_i, P_j, V_{i,j})$ where $Z = (I/V_{i,j})^{x' w_{j,i,\mathsf{iid}_j}}$.
- Set $\mathsf{sid}_{j,i,\mathsf{iid}_j} := P_i || P_j || I || R$.

on R from $\mathcal{A}$ as a message from P_j to (P_i, iid_i):

- Retrieve $(P_i, \mathsf{iid}_i, I) \in T$, $\mathsf{sid}_{i,j,\mathsf{iid}_i} := P_i || P_j || I || R$.
- Set $\mathsf{sid}_{i,j,\mathsf{iid}_i} := P_i || P_j || I || R$.
- Compute a new key as $K = \mathsf{H}'(Z, I, R, P_i, P_j, V_{i,j})$ where $Z = (R/V_{i,j})^{x' w_{i,j,\mathsf{iid}_i}}$.

On Random Oracles

on $\mathsf{H}(\mathsf{pwd})$ from $\mathcal{A}$:

If $\exists (\mathsf{pwd}, V) \in L_{\mathsf{H}}$: return V.
Otherwise draw $u \leftarrow \mathbb{Z}_p$ and compute $V = (X')^u$. Add $L_{\mathsf{H}} := L_{\mathsf{H}} \cup \{(\mathsf{pwd}, V)\}$ and $DL := DL \cup \{(V, u)\}$, and return V.

on $\mathsf{H}'\,(Z, I, R, P_i, P_j, V)$ from $\mathcal{A}$:

If $\exists\,(Z, I, R, P_i, P_j, V, K) \in L_{\mathsf{H}'}$: return K.
Otherwise draw $K \xleftarrow{\$} \mathcal{K}$, set $L_{\mathsf{H}'} := L_{\mathsf{H}'} \cup \{(Z, I, R, P_i, P_j, V, K)\}$, and return K.

Fig. 5: Intermediate simulator Sim for $\mathsf{Protoss}$ in the proof of Theorem 2

PAKE Sessions (conditioned on the event $\bar{E}1 \wedge \bar{E}2$)

on $(\mathsf{NewInstance}, P_i, \mathsf{iid}_i, P_j, b)$ from $\mathcal{F}_{\mathsf{lePAKE}}$:

- Draw $w_{i,j,\mathsf{iid}_i} \leftarrow \mathbb{Z}_p$ and compute $X = (X')^{w_{i,j,\mathsf{iid}_i}}$ and $I = XV_{i,j} = (X')^{(w_{i,j,\mathsf{iid}_i} + u_{i,j})}$ where we get $u_{i,j}$ via entries $(P_i, P_j, V_{i,j}) \in Q$ and $(V_{i,j}, u_{i,j}) \in DL(X')$. Set $T := T \cup \{(P_i, \mathsf{iid}_i, I)\}$ and send I from P_i to $\mathcal{A}$. Also add (P_i, iid_i, b) to CR, $(X, w_{i,j,\mathsf{iid}_i})$ to $DL(X')$.

on $(\mathsf{NewInstance}, P_j, \mathsf{iid}_j, P_i)$ from $\mathcal{F}_{\mathsf{lePAKE}}$ and I from $\mathcal{A}$ as a party message from P_i to (P_j, iid_j):

- Draw $w_{j,i,\mathsf{iid}_j} \leftarrow \mathbb{Z}_p$ and compute $X = (X')^{w_{j,i,\mathsf{iid}_j}}$ and $R = XV_{j,i} = (X')^{(w_{j,i,\mathsf{iid}_j} + u_{j,i})}$. Set $T := T \cup \{(P_i, \mathsf{iid}_i, R)\}$ and send R from P_j to $\mathcal{A}$. Also add (P_j, iid_j, b) to CR, $(X, w_{j,i,\mathsf{iid}_j})$ to $DL(X')$.
- Compute a new key K as follows: Sim checks whether there is a password pwd' with $(\mathsf{pwd}', V) \in L_{\mathsf{H}}$, $(P_i, P_j, I) \in T$, and $(I/V, x) \in DL(X')$. (This means that $X = I/V$ has been generated by some instance beforehand). If such a pwd' exists, use the SSQRDH decision oracle to check whether there is $(Z', P_i, P_j, I, R, V, K') \in L_{\mathsf{H}'}$ with Z' being equal to $Z = \mathsf{CDH}(g, I/V, R/V)$. In this case, use this $K = K'$. Otherwise, draw a random $K \leftarrow \mathcal{K}$. (If at some later point (Z, P_i, P_j, I, R, V) is asked to H', we keep the responses consistent by outputting K as well).
- Set $\mathsf{sid}_{j,i,\mathsf{iid}_j} := P_i || P_j || I || R$.

on R from $\mathcal{A}$ as a message from P_j to (P_i, iid_i):

- Retrieve $(P_i, \mathsf{iid}_i, I) \in T$, $\mathsf{sid}_{i,j,\mathsf{iid}_i} := P_i || P_j || I || R$.
- Set $\mathsf{sid}_{i,j,\mathsf{iid}_i} := P_i || P_j || I || R$.
- Sim checks whether there is a password pwd' with $(\mathsf{pwd}', V) \in L_{\mathsf{H}}$, (P_j, P_i, R), and $(R/V, x) \in DL(X')$ (This means that $X = R/V$ has been generated by some instance beforehand). If such a pwd' exists, compute the key K as follows: use the SSQRDH decision oracle to determine if there is $(Z', P_i, P_j, I, R, V, K') \in L_{\mathsf{H}'}$ with Z' being equal to $Z = \mathsf{CDH}(g, I/V, R/V)$. In this case use this $K = K'$. Otherwise, draw a random $K \leftarrow \mathcal{K}$.(Again, if at some later point (Z, P_i, P_j, I, R, V) is asked to H', we keep the responses consistent by outputting K as well).

On Random Oracles

on $\mathsf{H}(\mathsf{pwd})$ from $\mathcal{A}$:

If $\exists(\mathsf{pwd}, V) \in L_{\mathsf{H}}$: return V.
Otherwise draw $u \leftarrow \mathbb{Z}_p$ and compute $V = (X')^u$. Add $L_{\mathsf{H}} := L_{\mathsf{H}} \cup \{(\mathsf{pwd}, V)\}$ and $DL := DL \cup \{(V, u)\}$, and return V.

on $\mathsf{H}'\,(Z, I, R, P_i, P_j, V)$ from $\mathcal{A}$:

If $\exists\,(Z, I, R, P_i, P_j, V, K) \in L_{\mathsf{H}'}$: return K.
Otherwise draw $K \xleftarrow{\$} \mathcal{K}$, set $L_{\mathsf{H}'} := L_{\mathsf{H}'} \cup \{(Z, I, R, P_i, P_j, V, K)\}$, and return K.

Fig. 6: Next simulator Sim for $\mathsf{Protoss}$ in the proof of Theorem 2

PAKE Sessions

on $(\mathsf{NewInstance}, P_i, \mathsf{iid}_i, P_j, b)$ from $\mathcal{F}_{\mathsf{lePAKE}}$:

- Draw $w_{i,j,\mathsf{iid}_i} \leftarrow \mathbb{Z}_p$ and compute $X = (X')^{w_{i,j,\mathsf{iid}_i}}$ and $I = XV_{i,j} = (X')^{(w_{i,j,\mathsf{iid}_i} + u_{i,j})}$ where we get $u_{i,j}$ via entries $(P_i, P_j, V_{i,j}) \in Q$ and $(V_{i,j}, u_{i,j}) \in DL(X')$. Set $T := T \cup \{(P_i, \mathsf{iid}_i, I)\}$ and send I from P_i to $\mathcal{A}$. Also add (P_i, iid_i, b) to CR, $(X, w_{i,j,\mathsf{iid}_i})$ to $DL(X')$.

on $(\mathsf{NewInstance}, P_j, \mathsf{iid}_j, P_i)$ from $\mathcal{F}_{\mathsf{lePAKE}}$ and I from $\mathcal{A}$ as a party message from P_i to (P_j, iid_j):

- Draw $w_{j,i,\mathsf{iid}_j} \leftarrow \mathbb{Z}_p$ and compute $X = (X')^{w_{j,i,\mathsf{iid}_j}}$ and $R = XV_{j,i} = (X')^{(w_{j,i,\mathsf{iid}_j} + u_{j,i})}$. Set $T := T \cup \{(P_i, \mathsf{iid}_i, R)\}$ and send R from P_j to $\mathcal{A}$. Also add (P_j, iid_j, b) to CR, $(X, w_{j,i,\mathsf{iid}_j})$ to $DL(X')$.
- Add $(X, x' w_{j,i,\mathsf{iid}_j})$ to $DL(g)$.
- Set $\mathsf{sid}_{j,i,\mathsf{iid}_j} := P_i || P_j || I || R$.
- The computation of the key K now depends on whether certain conditions are fulfilled or not.
  - If (P_j, iid_j) is linked to a good instance (P_i, iid_i), the key K of (P_j, iid_j) will be computed via query $(\mathsf{FreshKey}, P_j, \mathsf{sid}_{j,i,\mathsf{iid}_j})$ to $\mathcal{F}_{\mathsf{lePAKE}}$. (According to the definition of the $\mathsf{FreshKey}$ query this does not change the distribution).
  - Otherwise, Sim checks whether there is a password pwd' with $(\mathsf{pwd}', V) \in L_{\mathsf{H}}$, $(P_i, P_j, I) \in T$, and $(I/V, x) \in DL(X')$. (This means that $X = I/V$ has been generated by some instance beforehand). If such a pwd' exists query $(\mathsf{TestPW}, P_j, \mathsf{iid}_j, \mathsf{pwd}')$ to $\mathcal{F}_{\mathsf{lePAKE}}$ to learn whether pwd' is equal to the correct password pwd used in instance (P_j, iid_j).
    - * If such a pwd' exists and the response of $\mathcal{F}_{\mathsf{lePAKE}}$ indicates $\mathsf{pwd}' = \mathsf{pwd}$ (correct guess), compute the key K as follows: use the SSQRDH decision oracle to determine if there is $(Z', P_i, P_j, I, R, V, K) \in L_{\mathsf{H}'}$ with Z' being equal to $Z = \mathsf{CDH}(g, I/V, R/V)$. In this case use this K. Otherwise, draw a random $K \leftarrow \mathcal{K}$.[a] Moreover, we make a $(\mathsf{CorruptKey}, P_j, \mathsf{iid}_j, \mathsf{sid}_{j,i,\mathsf{iid}_j}, K)$ to $\mathcal{F}_{\mathsf{lePAKE}}$. (This does not change the view of $\mathcal{Z}$).
    - * If such a pwd' does not exist or if $\mathcal{F}_{\mathsf{lePAKE}}$ returns wrong guess, query $(\mathsf{FreshKey}, P_j, \mathsf{iid}_j, \mathsf{sid}_{j,i,\mathsf{iid}_j})$ to $\mathcal{F}_{\mathsf{lePAKE}}$. (This does not change the view of $\mathcal{Z}$).

on R from $\mathcal{A}$ as a message from P_j to (P_i, iid_i):

- Retrieve $(P_i, \mathsf{iid}_i, I) \in T$,
- Set $\mathsf{sid}_{j,i,\mathsf{iid}_j} := P_i || P_j || I || R$.
- The computation of the key K now depends on whether certain conditions are fulfilled or not.
  - If (P_i, iid_i) and an instance (P_j, iid_j) of P_j are linked to each other and $(P_i, \mathsf{iid}_i, 1) \in CR$, then a random key K must have been assigned to (P_j, iid_j). Now Sim sets $\mathsf{sid}_{i,j,\mathsf{iid}_i} = \mathsf{sid}_{j,i,\mathsf{iid}_j}$ and query $(\mathsf{CopyKey}, P_i, \mathsf{iid}_i, \mathsf{sid}_{i,j,\mathsf{iid}_i})$ to $\mathcal{F}_{\mathsf{lePAKE}}$ (According to the definition of the $\mathsf{CopyKey}$ query this does not change the distribution from $\mathcal{Z}$'s point of view).
  - Otherwise, Sim checks whether there is a password pwd' with $(\mathsf{pwd}', V) \in L_{\mathsf{H}}$, (P_j, P_i, R), and $(R/V, x) \in DL(X')$ (This means that $X = R/V$ has been generated by some instance beforehand). If such a pwd' exists, query $(\mathsf{TestPW}, P_i, \mathsf{iid}_i, \mathsf{pwd}')$ to $\mathcal{F}_{\mathsf{lePAKE}}$ to learn whether pwd' is equal to the correct password pwd used in instance (P_i, iid_i).
    - * If such a pwd' exists and the response of $\mathcal{F}_{\mathsf{lePAKE}}$ indicates $\mathsf{pwd}' = \mathsf{pwd}$ (correct guess), compute the key K as follows: use the SSQRDH decision oracle to determine if there is $(Z', P_i, P_j, I, R, V, K) \in L_{\mathsf{H}'}$ with Z' being equal to $Z = \mathsf{CDH}(g, I/V, R/V)$. In this case use this K. Otherwise, draw a random $K \leftarrow \mathcal{K}$.[b] Moreover, we make a $(\mathsf{CorruptKey}, P_i, \mathsf{iid}_i, \mathsf{sid}_{i,j,\mathsf{iid}_i}, K)$ to $\mathcal{F}_{\mathsf{lePAKE}}$. (This does not change the view of $\mathcal{Z}$).
    - * If such a pwd' does not exist or if $\mathcal{F}_{\mathsf{lePAKE}}$ returns wrong guess, query $(\mathsf{FreshKey}, P_i, \mathsf{iid}_i, \mathsf{sid}_{i,j,\mathsf{iid}_i})$ to $\mathcal{F}_{\mathsf{lePAKE}}$ (This does not change the view of $\mathcal{Z}$).

[a] If at some later point (Z, P_i, P_j, I, R, V) is asked to H', we keep the responses consistent by outputting K as well.

[b] Again, if at some later point (Z, P_i, P_j, I, R, V) is asked to H', we keep the responses consistent by outputting K as well.

Fig. 7: Simulator Sim for $\mathsf{Protoss}$ in the proof of Theorem 2 that is equivalent to the ideal game.

On Random Oracles

```
on $\mathsf{H}(\mathsf{pwd})$ from $\mathcal{A}$:
If $\exists(\mathsf{pwd}, V) \in L_\mathsf{H}$ : return $V$.
Otherwise draw $u \leftarrow \mathbb{Z}_p$ and compute $V = (X')^u$. Add $L_\mathsf{H} := L_\mathsf{H} \cup \{(\mathsf{pwd}, V)\}$ and
$DL := DL \cup \{(V, u)\}$, and return $V$.

on $\mathsf{H}'(Z, I, R, P_i, P_j, V)$ from $\mathcal{A}$:
If $\exists (Z, I, R, P_i, P_j, V, K) \in L_{\mathsf{H}'}$ : return $K$.
Otherwise draw $K \overset{\$}{\leftarrow} \mathcal{K}$, set $L_{\mathsf{H}'} := L_{\mathsf{H}'} \cup \{(Z, I, R, P_i, P_j, V, K)\}$, and return $K$.
```

Fig. 8: Continuation of Fig. 7.