Password-authenticated Key Exchange and Applications

Kristian Gjøsteen*

June 28, 2024

Abstract

We analyse a two password-authenticated key exchange protocols, a variant of CPace and a protocol related to the well-known SRP protocol. Our security results are tight. The first result gives us some information about trade-offs for design choices in CPace. The second result provides information about the security of SRP.

Our analysis is done in a new game-based security definition for password-authenticated key exchange. Our definition accomodates arbitrary password sampling methodologies. Our definition also supports modular security analysis, which we illustrate by giving two example applications of password-authenticated key exchange: password-authenticated secure channels and password-authenticated device authorisation, capturing popular applications of passwords.

1 Introduction

A *password* is a human-memorable string of symbols. Despite their well-documented weaknesses, passwords will likely be with us a while yet. Hence, we want to understand the security properties of cryptography based on passwords. In this work, we consider how to define security for key exchange authenticated using passwords, so-called *password-authenticated key exchange* (PAKE), analyse an existing and a novel protocol, and show how our definition allows compositional reasoning for password-based applications.

A password-authenticated key exchange protocol is a two-party protocol where each party takes a password and (public) associated data as input, then the parties take turns exchanging messages and finally either reject or output a session key. The associated data encodes the context for the key exchange. Our first cryptographic goal concerns authentication:

- If one party accepts, the other party knows the password and agrees on the associated data.

*kristian.gjosteen@ntnu.no, Department of Mathematical Sciences, NTNU – Norwegian University of Science and Technology, Trondheim, Norway.

It follows that any reasonable protocol acts as a password testing oracle, in the sense that an adversary can connect to the system pretending to be a user in order to test if a particular candidate password is correct or not. Such attacks are unavoidable. This gives us our second cryptographic goal, saying that this should be the only mechanism available to the adversary:

- An adversary's ability to guess passwords is bounded by the number of sessions.

Finally, note that the password's role is authentication. Confidentiality should not depend on the password. This gives us our third cryptographic goal:

- When two honest parties exchange a session key without adversarial interference, then, *even if the adversary knows the password*, the session key remains private.

Passwords are typically used to authenticate *users* of a system to the *system*. Theoretically, passwords are attractive because they are human-memorable, so passwords connect the use of a system directly to something that is only ever recorded on long-term storage inside the system and inside the heads of the users. A better explanation for the use of passwords as a system security mechanism, now and in the foreseeable future, is the low startup cost and the ease of use, primarily for the system owner, but in some sense also for the user of the system. Any analysis of password-based cryptography must therefore also consider the system owner point of view, not just the user point of view.

A user often controls their password sampling methodology, so from a user's point of view, the probability that an adversary will be able to guess their password is an interesting number. A system owner often cannot dictate their users' password sampling methodologies or their general behaviour. The probability that some adversary is able to correctly guess *some* user's password may very well be close to 1, which is not interesting. From a system owner's point of view, a more interesting number may be *how many* passwords will be guessed.

- We must work in a multi-password, multi-session setting.

1.1 Related Work

We give a somewhat limited overview of related work. Boyd *et al.* [7] gives a good overview of password-authenticated key exchange, though there has since been developments.

Bellovin and Merritt [6] first proposed password-authenticated key exchange. As discussed, an *online* password guessing attack will always be possible. The main goal is therefore to authenticate key exchange using a password without enabling a much more powerful *offline* password guessing attack. One simple instantiation of their proposal uses a password to authenticate a Diffie-Hellman key exchange by encrypting the group elements using the password as a key. There are a number of technical issues with this idea, but it has since been analysed, by among others Liu *et al.* [22].

A different approach based on Diffie-Hellman is that of SPEKE [18], where the password defines the Diffie-Hellman generator. A recent variant is CPace [4, 16] with analysis by Abdalla *et al.* [3] among others.

Other Security Properties In the introduction, we outlined the basic properties. Here we outline further desirable properties, *which we do not handle in this paper*.

EKE and SPEKE are both *symmetric*, in the sense that the two parties have essentially the same representation of the password. Commonly, password-authenticated key exchange is used between a server and a client acting on behalf of a user. The client gets a human-expressible form of the password from the user, but the server could in principle have some other representation, which gives us *asymmetric* password-authenticated key exchange. In particular, this representation need not allow impersonation of the client towards that particular server. This is obviously a useful security property, though limited.

One early approach to asymmetric password-based key exchange was SRP [29]. Unlike many other schemes, it has seen wide adoption [8], but it does not seem to have attracted much cryptographic analysis, beyond the initial analysis by the protocol designer and some attacks on early iterations. Sherman *et al.* [25] used formal methods to do a simplified analysis on an idealised version of SRP.

A server compromise allows offline attacks. If the adversary has a particular target in mind, the offline attack could be made more effective by precomputation. The interesting use case is when the adversary expects the server compromise to be quickly detected, and consequently has limited time for exploitation. Schemes like OPAQUE [19] prevent this precomputation. This is a useful security property, though quite limited.

Some protocols, such as SRP, are explicitly authenticated, in the sense that an instance will not accept unless there is a partner or the password has been misused. Some protocols, such as CPace, can be thought of as implicitly authenticated, in the sense that a session key is private, unless there is no partner and the password has been misused. We can build applications on top of implicitly authenticated key exchange, but it is easier to work with explicit authentication. Fortunately, the standard folklore upgrade technique of sending key confirmation messages works well.

Modelling Strategies Correct and useful modelling of security for password-authenticated key exchange has been a significant challenge. This is not unexpected, since even ordinary key exchange has been difficult to model. Generally, a good definition of security (a) captures the essential security properties in such a way that (b) we are able to prove some scheme secure, and (c) we can do modular security analysis of larger systems. It is also good if the definition is easy to understand and work with.

The natural approach is to draw inspiration from the modelling of ordinary key exchange using symmetric keys for authentication. Instead of sampling keys from the uniform distribution on some key set, we sample passwords from some

distribution on a set of possible passwords. This is the approach of Bellare *et al.* [5], known as the BPR framework. There are many other variants of the BPR framework. The BPR framework certainly succeeds at (b), but Shoup [27] (an extended version of [26]) suggests that it fails at (c), in particular for secure-channel protocols built on top of password-authenticated key exchange. There are variants [28] of BPR that allow for composability.

One problem is that cryptographic protocol designers will typically not know the password distribution used in practice, and there may not be a unique password distribution. We claim that this also breaks (a) for the BPR framework. To address this, we could allow the adversary to specify the distribution. We would also like to model passwords that are correlated, as well as passwords that come from both high- and low-entropy distributions. When the adversary specifies the distribution, these issues lead to a number of technical difficulties that we found hard to resolve.

A more fundamental difficulty is what happens when the adversary guesses a password. In the BPR framework, the adversary simply wins. This is even carried over to some compositional results [28]. As discussed above, this may be acceptable modelling for a single user, but it is unsatisfactory from a system owner point of view, again breaking (a).

Another approach is to ignore password sampling and instead define a notion based on simulatability, using Canetti's Universal Composability (UC) framework [9] or similar. In this approach, the password-authenticated key exchange protocol is compared to a so-called ideal functionality which defines security. Any definition in the UC framework should excel at (c).

Determining the correct ideal functionality seems to be tricky. Canetti *et al.* [10] proposed one functionality and Gentry *et al.* [14] proposed a functionality for asymmetric protocols. Abdalla *et al.* [3] found a common flaw in several earlier proposed functionalities, and therefore proposed a corrected functionality that captures the single-password, single-session implicitly authenticated case, using the standard UC theorems to get to the multi-password, multi-session case. Shoup [27] also found it necessary to modify earlier proposed functionalities in order to better capture security properties and fix flaws, and proposed a functionality that captures the multi-password, multi-session explicitly authenticated case. The system owner's point of view can be accommodated by the latter approach, but is harder to deal with in the former approach.

It is interesting to look at modelling artefacts. A draft specification [4] for one protocol recommends exchanging randomness to build session identifier, grounded in a UC security proof. Shoup [27] handles session identifiers differently and does not end up with the same recommendation, though of course for a different protocol and a different UC framework.

It is interesting that the recent analysis of TLS-OPAQUE [17] in the UC framework does not use UC functionalities for password-authenticated key exchange, because they use passwords only for authentication, not session key establishment.

1.2 Our Contribution

We claim two main contributions:

- We analyse a variant of CPace [4] and a protocol related to SRP [29], providing more information about both CPace and SRP.
- We do our analysis in a new security framework.

We also think two other minor points are worth pointing out:

- Our framework supports the usual modular security analysis.
- Our framework allows the usual folklore implicit-to-explicit design strategy, but unlike the corresponding case for ordinary key exchange [13], we do not get a tightness loss.

We discuss each claim separately.

Analysis We give (Section 3.4) a new analysis of the CPace [4, 16]. The proof techniques are based on the techniques used by Abdalla *et al.* [3], but because we work in a multi-password, multi-session setting, it is natural to emphasise tightness, which we achieve by careful use of standard rerandomisation techniques.

We consider reduction tightness and our analysis suggests that the password should be included when deriving the session key. If it is not, the reduction will not be tight and larger groups will be required for a fixed security level. Abdalla *et al.* [3] and the draft CPace specification [4] do not include the password. Our analysis therefore **gives information about the trade-offs involved in this particular design choice**. Also, our analysis suggests that the session identifier randomness suggested by the specification can be dispensed with in many settings.

We also analyse (Section 3.5) what we believe is a novel construction (Example 16). A few simple changes (Remark 17) turn this protocol into the SRP protocol [29] in such a way that SRP should be no more secure than our protocol. It seems likely that **SRP is as secure as our protocol**. SRP has been widely deployed [8], so this is an interesting result on its own.

The protocol can be made asymmetric (Remark 18). It is interesting to note that the post-reveal, our protocol degenerates to a variant of a protocol from Cohn-Gordon *et al.* [12] with one-sided authentication. This is consistent with the impossibility result for asymmetric protocols of Liu *et al.* [22].

One interesting property of our protocol is that it would be very easy to derive a mixed-authentication protocol from it (Remark 19), where the server would authenticate with the password as well as a long-term cryptographic key. Server signatures achieve the same effect in some settings (Remark 23).

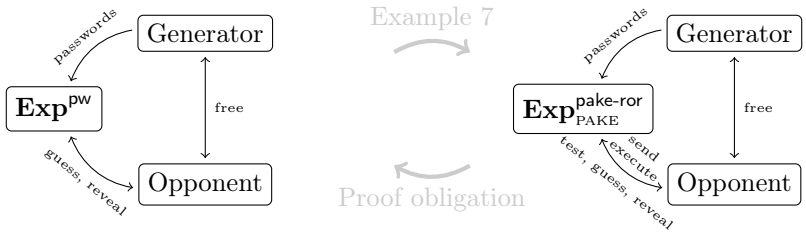

Figure 1: Security definition game structures: password guessing left, password-authenticated key exchange right.

Framework Our framework is game-based and very similar to frameworks for ordinary key exchange. The two main novelties with respect to earlier game-based frameworks for password-based key exchange are that we allow for arbitrary and varying password sampling methodologies, and that we measure the adversary's advantage in a new way.

A password guessing algorithm tries to guess one or more passwords by making *guesses*, and it may *reveal* other passwords to exploit the fact that passwords are not chosen independently. Password guessing algorithms trivially become adversaries against password-authenticated key exchange, which Example 7 demonstrates in some detail. Morally speaking, we have security if we show that every adversary essentially comes from a password guessing algorithm.

The natural password guessing game (Section 3.1) is sketched in Figure 1. A *password generator* generates passwords, while an *opponent* tries to guess passwords. An experiment facilitates the guessing, and we allow free communication between the generator and the opponent to model leakage and adversarial influence. Note that the difficulty of password guessing depends strongly on the generator, and *we make no claim or assumption about difficulty.*

The security definition for password-authenticated key exchange (Section 3.2) ought to be as similar to ordinary key exchange as possible, so we use the structure of the password guessing game and have an experiment based on ordinary key exchange that simulates the honest parties in password-based key exchange. The opponent interacts with the experiment in the usual way. This structure is also shown in Figure 1.

Recall that an adversary against key exchange wins in one of three ways: breaking authentication by some instance having no partner, breaking authentication by some instance having too many partners, or breaking confidentiality of the session keys. For password-based key exchange, the latter two are unproblematic, but the first is troublesome, since we cannot easily know when a missing partner is due to a cryptographic break, or trivially because of password misuse. We therefore define two success measures for an opponent. The *advantage* measures breaking confidentiality and breaking authentication via too many partners. The *success rate* measures breaking authentication via having no partner. We want to *bound* the advantage, and we want to *explain* the

success rate by exhibiting an opponent for the password guessing game that achieves the same success rate. For the latter argument, it is essential that this opponent works with the same generator, otherwise the opponent would be meaningless.

Informally, we say that the generator and the opponent are playing some unknown password-based game, with some experiment assisting the opponent. The security claim is that the password-authenticated key exchange experiment should be no more helpful to the opponent than the password guessing experiment.

Our definition contains the BPR framework [5] and some recent refinements [1, 2] as a special case, by considering a generator that samples passwords from a fixed distribution. Note that relative to the BPR framework, our definition is still useful for compositional designs of larger protocols. We therefore claim that **our framework improves on the BPR framework**.

The comparison with the UC framework is more complicated. We expect UC security to imply security in our model. We consider a generator-opponent pair to be an environment and use a dummy adversary which can then be simulated (by UC security) to give us an ideal UC adversary. We then turn the opponent together with the ideal UC adversary into a password guessing opponent with the same success rate and no advantage. There are a number of minor technical obstacles, and *we do not prove such a result.*

The converse result seems difficult to obtain. Syntactical issues regarding who makes which queries can probably be overcome. But the deeper reason is that a password guessing opponent only needs to explain the missing partners by guessing the passwords. There's no requirement that it duplicates any other behaviour of the opponent. This means that any ideal adversary will find it difficult to simulate the behaviour of the honest instances, allowing a suitable environment to distinguish.

It seems, then, that we propose a security definition that is implied by, but not equivalent to, UC security notions. Practically, this seems to be of little importance. Comparing security proofs in our model with UC proofs [3, 27] reveal very similar proof techniques. And indeed, the proof obligations in the two settings are very similar.

While the ideas involved in UC security are intuitive and beautiful, the working cryptologist still needs to handle the many technical issues related to UC frameworks. Our framework is much simpler than the full UC models, which is natural since we are trying to solve a much smaller smaller problem. We therefore believe that our framework makes password-based key exchange more accessible than UC security. Of course, simplicity is in some sense subjective, so our belief should perhaps not count for too much. Still, we claim that **our framework improves on UC security with respect to simplicity**.

Our definition does not capture asymmetric security. We do explain (Remark 10) how one could model asymmetry. We leave this for a future paper.

Some definitions include the option of running honest instances with incorrect passwords, modelling user typing mistakes. It is possible to design protocols that fail catastrophically in the presence of typing mistakes. We allow the gen-

erator to choose correlated passwords, so we could model typing mistakes by relaxing our context-respecting notion (see Section 3.2) in some suitable fashion. However, we are not convinced this is important.

Applications of Password-authenticated Key Exchange We define (Section 4) two password-authenticated applications, channels and device authorisation. Our main objective is to demonstrate that **our security definition allows modular analysis of composed protocols**, which the BPR framework and some early ideal functionalities would not. The applications themselves are important. Our protocols are designed mainly for simplicity, with the secure channel being essentially equivalent to the secure channel of Canetti and Krawczyk [11]. We make no claim of novelty for the protocols. One may want protocols with a few more ornamentations.

Unlike [28], we do not prove a general composition result for our security definition. While general composition results may be useful, we leave this for future work.

A mixed-authentication protocol as above combined with the natural secure channel protocol in Section 4.1 would be a straight-forward way to build something security-equivalent to TLS-OPAQUE [17]. Alternatively, we could follow the approach outlined in Remarks 23 and 24 to achieve the same result with surveillance resistance. Again, we make no claim of novelty.

Implicit-to-Explicit Design Strategy We prove (Section 3.3) the relevant folklore theorem regarding the standard implicit-to-explicit upgrade strategy. Compared to other proofs of similar theorems, we claim that **our model makes this proof particularily simple**. Note that some protocols would be insecure without key confirmation tags, which is one reason why we include the protocol in Section 3.5.

Some UC ideal functionalities define implicit authentication [3], while others define explicit authentication [27]. When the functionalities are defined in the same variant of the UC framework and are otherwise compatible, it is possible to prove a similar theorem [2].

2 Background

2.1 Notation

Uppercase letters denote sets, calligraphic letters algorithms, small-caps protocols and sans-serif letters problems and constants. In protocol messages and records stored by algorithms, $-$, $\perp$ and $\top$ denote special symbols that are distinct from all other values. When we do not care about a particular value in a message, transcript or record, we denote this by a centered dot ($\cdot$).

2.2 Groups

Our schemes are based on a group G of order p with generator g. We define Diffie-Hellman-based problems. The first is for convenience, while the second is essential. The first is merely a multi-sample, multi-generator variant of Computational Diffie-Hellman (CDH). It is well known that the ordinary CDH problem reduces tightly to this variant by random self-reducibility, even when we include Gap or Strong Diffie-Hellman oracles. The second variant is a more obscure problem, but is plausibly secure.

Definition 1. Let G be a cyclic group of order p. The *Computational Diffie-Hellman* (CDH) problem provides the solver with three oracles that sample as follows:

- The first oracle samples the ith sample as $g_i \xleftarrow{r} G$.
- The second oracle samples the jth sample as $x_j \xleftarrow{r} G$.
- The third oracle samples the lth sample as $y_l \xleftarrow{r} G$.

Eventually, the solver outputs (i, j, l, z). The tuple (i, j, l, z) is an answer if (g_i, x_j, y_l, z) is a DDH tuple. A (τ, n_u, n_s)-solver against CDH uses time at most τ, makes at most n_u to the first oracle and at most n_s queries to the second and third oracle.

The *Strong Computational Diffie-Hellman* (StCDH) problem gives the adversary an additional oracle that on input of (i, j, y, z) returns 1 if (g_i, x_j, y, z) is a DDH tuple, otherwise it returns 0.

We denote the success probability of an adversary $\mathcal{B}$ by $\mathbf{Adv}^{\mathsf{CDH}}_G(\mathcal{B})$ and $\mathbf{Adv}^{\mathsf{StCDH}}_G(\mathcal{B})$, respectively.

Remark 1. We shall also use a fixed-generator variant of this problem, and consider that adversaries against this problem makes 0 queries to the first oracle. It is not trivially equivalent to sampling a single generator, since the fixed generator could be special and a DDH oracle for a special generator could be hard to simulate using only a DDH oracle for a single random generator.

Definition 2. Let G be a cyclic group of order p. The *simultaneous Diffie-Hellman tuple finder* (SDHF) problem provides the solver with two oracles that sample as follows:

- The first oracle samples the ith sample as $g_i \xleftarrow{r} G$.
- The second oracle samples the jth sample as $x_j \xleftarrow{r} G$.

Eventually, the solver outputs $(i_1, i_2, j, y, z_1, z_2)$. A tuple $(i_1, i_2, j, y, z_1, z_2)$, $y \neq 1$, is an answer if (g_{i_1}, x_j, y, z_1) and (g_{i_2}, x_j, y, z_2) are both DDH tuples. A (τ, n_u, n_s)-solver against SDHF uses time at most τ, makes at most n_u queries to the first oracle and at most n_s queries to the second oracle.

The *Strong simultaneous Diffie-Hellman tuple finder* (StSDHF) gives the adversary an additional oracle that on input of (i, j, y, z) returns 1 if (g_i, x_j, y, z) is a DDH tuple, otherwise it returns 0.

We denote the success probability of an adversary $\mathcal{B}$ by $\mathbf{Adv}_G^{\mathsf{SDHF}}(\mathcal{B})$ and $\mathbf{Adv}_G^{\mathsf{StSDHF}}(\mathcal{B})$, respectively.

Remark 2. Given a fixed generator, the strong variants of both problems could sample group elements by sampling exponents and exponentiating, making the DDH oracles easy to compute.

2.3 Block Ciphers

A *block cipher* is a pair of functions $(\pi, \pi^{-1}) : K \times S \to S$ such that for any $k \in K$, $\pi(k, \cdot)$ and $\pi^{-1}(k, \cdot)$ are inverse functions.

Sometimes, we want a block cipher on some set S_0. If we have a block cipher (π_0, π_0^{-1}) on a set S containing S_0, but not too much larger, and it is easy to recognise elements of S_0, there is a simple technique [21] to get a block cipher on S_0. A set element $s_0 \in S_0$ defines a sequence of elements $s_1, s_2, \ldots$ through the iteration rule $s_{i+1} = \pi_0(k, s_i)$. We define $\pi(k, s_0)$ to be s_i, where i is such that $s_j \notin S_0$ for $j = 1, 2, \ldots, i-1$. We define π^{-1} in the same way. There are side-channel issues, but they can be managed.

If we do not have a suitable block cipher on a sufficiently small set S, we can build one using Luby-Rackoff [23].

Using the above techniques, we can make a (somewhat slow) block cipher on the set of rational points on an elliptic curve over a finite field. We cannot use the above techniques to make a block cipher on the set of j-invariants of elliptic curves isogenous to some particular curve, since this set is very sparse in the finite field.

2.4 Symmetric Encryption

A *symmetric cryptosystem* SYM consists of a set of keys K, a set of plaintexts P, and encryption and decryption algorithms $\mathcal{E}_0$ and $\mathcal{D}_0$, such that $\forall k \in K, \forall m \in P : \Pr[\mathcal{D}_0(k, \mathcal{E}_0(k, m)) = m] = 1$.

We define multi-user security. The experiment allows the adversary to sample keys, encrypt and decrypt under those keys, and reveal keys. A challenge oracle returns an encryption of either the chosen ciphertext or a random message of the same length. The adversary may query it multiple times for multiple keys. The adversary may not both challenge and reveal a key.

The adversary against *real-or-random* must guess what the challenge oracle returns, with the usual restrictions on decryption oracle queries. The adversary breaks *ciphertext integrity* if a chosen ciphertext query succeeds for anything not previously returned by an encrypt or challenge query under that key.

A (τ, n_u, n_c, n_e)-adversary against SYM makes at most n_u sample key queries, n_c challenge queries and n_e chosen plaintext and ciphertext queries, and the runtime of the adversary and the experiment is at most τ. We denote the advantages of such an adversary $\mathcal{B}$ by $\mathbf{Adv}_{\mathrm{SYM}}^{\mathsf{ind\text{-}cca}}(\mathcal{B})$ and $\mathbf{Adv}_{\mathrm{SYM}}^{\mathsf{int\text{-}ctxt}}(\mathcal{B})$.

2.5 Random Oracle and Ideal Cipher Models

A *random oracle* models hash functions such as HMAC-x and SHA-3, while an *ideal cipher oracle* models a block cipher such as Threefish.

Our oracles sample function values only when they need to, while keeping track of previous responses.

A random oracle from S_0 to S_1 keeps a list of records (s_0, s_1) that together define a partial function $S_0 \to S_1$. On a query s_0, either a record (s_0, s_1) exists or the oracle samples $s_1 \xleftarrow{r} S_1$ and records (s_0, s_1). In either case, it returns s_1.

An ideal cipher oracle on set S with keys K keeps a list of records (k, s_0, s_1) such that the records for any $k \in K$ defines a partial permutation on S. The adversary makes queries (k, s_i, i) for $i \in \{0, 1\}$, either a record (k, s_0, s_1) exists or the oracle samples s_{1-i} and records (k, s_0, s_1). In either case, it returns s_{1-i}. Note that for the ideal cipher oracle, sampling would have to be without replacement, so some extra accounting is needed.

These oracles are not free, and their cost is not linear in the number of queries. We include this into the total run-time. When we modify oracles in security proofs, the modified oracles will have cost comparable to at most a small multiple of the original oracles.

3 Password-authenticated Key Exchange

A password is a human-memorable string of symbols. Often, it is convenient to map this string of symbols into a representation more suitable for cryptography. We do this with a *password-based key derivation function*, denoted by *pbkdf*. These are carefully designed functions, but modelled as random oracles in our context.

In client-server settings, per-user nonces or salts are often included as input in a password-based key derivation function. These are considered public since they must be communicated to any client trying to connect, who cannot otherwise derive the correct password representation. Alternatively, password-associated data such as user name and other identifying information can be included. Regardless, the effect is similar to using a per-password key derivation function.

3.1 Passwords and Password Guessing

Intuitively, a password guessing game starts by using some fixed sampling process to choose a password, then allowing the opponent to make guesses until it succeds or gives up. Unfortunately, there are some issues. We would like to model different password sampling processes. We would like to model distinct, but correlated passwords. In fact, we would like the opponent to be able to influence the sampling process. And we want to model leakage of information.

The simplest approach is to model password generation as a separate algorithm, essentially part of the adversary specification, that communicates freely with the opponent. Since we do not specify what this generator algorithm does,

The password guessing experiment proceeds as follows:

1. On the jth *password* query pw from $\mathcal{G}$, do:
   (a) Record (j, pw) and respond with j.
2. On a *guess* query (j, pw) from $\mathcal{A}$, do:
   (a) If (j, pw) is recorded, erase it and respond with $\top$. Otherwise, respond with $\bot$.
3. On a *reveal* query j from $\mathcal{A}$, do:
   (a) If (j, pw) is recorded, erase it and respond with pw.

Figure 2: Experiment $\mathbf{Exp}^{\mathsf{pw}}$ for the password guessing game, interacting with a password generator $\mathcal{G}$ and opponent $\mathcal{A}$.

we cannot say anything about how hard the resulting password guessing game is. In particular, how do we know that the generator algorithm does not simply leak passwords to the adversary? *This is not a problem.* The goal of the password guessing game is not to be difficult, but to provide a baseline for difficulty.

For later convenience, we also capture the intuitive approach.

We repeat for emphasis: The generator and the opponent play an unknown password-related game. The game may be easy, but the game may also be difficult game and the opponent strongly dependent on the available assistance. Our theorems must hold for all such games, both easy and difficult.

Definition 3. A (τ, n_u, n_s)-*password guesser* $(\mathcal{G}, \mathcal{A})$ is a pair of algorithms interacting with the experiment in Figure 2, where $\mathcal{G}$ and $\mathcal{A}$ may freely send each other messages, $\mathcal{G}$ makes at most n_u password queries (whose responses are sent to $\mathcal{A}$), and $\mathcal{A}$ makes reveal queries and at most n_s guess queries, such that the total runtime of $\mathcal{G}$, $\mathcal{A}$ and the experiment is at most τ.

The *success rate* $\mathbf{SuccR}^{\mathsf{pw}}(\mathcal{G}, \mathcal{A})$ of a password guesser $(\mathcal{G}, \mathcal{A})$ is the number of $\top$ responses.

Definition 4. Let $\mathcal{X}$ be a probability space on a set W. A *simple* (τ, n_u, n_s)-*password guesser for* $\mathcal{X}$ is an interactive algorithm $\mathcal{A}$ such that $(\mathcal{G}_{\mathcal{X}}, \mathcal{A})$ is a (τ, n_u, n_s)-password guesser, where $\mathcal{G}_{\mathcal{X}}$ is the following algorithm interacting with the password search experiment and $\mathcal{A}$. When $\mathcal{G}_{\mathcal{X}}$ receives a $\top$ query from $\mathcal{A}$, it samples $pw \xleftarrow{r} \mathcal{X}$ and sends pw to the password guessing experiment.

The success rate of a password guesser is not a number, but a probability space on the integers. When convenient, we shall identify it with the corresponding cumulative distribution function on the integers. We will need to compare the success rates of various password guessers and similar algorithms. For reasons that will become apparent, we cannot expect our algorithms to have identical or very similar success rates. Instead, we shall show that they are "almost at least as good".

Let $f, f' : S \to \mathbb{R}$ be functions. For $\delta \in \mathbb{R}$, we say that $f' \leq f + \delta$ if $f'(x) \leq f(x) + \delta$ for all $x \in S$. By thinking about probability spaces as cumulative distribution functions, we extend this notation to success rates and write $SR' \leq SR + \delta$, meaning that for all x

$$\Pr[SR' \leq x] \leq \Pr[SR \leq x] + \delta,$$

with the implication that SR' is "almost at least as good" as SR if δ is small. It may be much better. Note that the direction of the inequality is opposite of advantages.

3.2 Definition

Definition 5. A *password-authenticated key exchange (PAKE)* protocol $\text{PAKE} = (K, W, AD, \mathcal{I}, \mathcal{R})$ consists of sets of *session keys* K, passwords W and associated data AD, and two algorithms:

- The interactive *initiator* algorithm $\mathcal{I}$ takes as input associated data $ad \in AD$ and a password $pw \in W$. It alternates between sending and receiving messages. Eventually, it either outputs a session key k or $\perp$ signifying failure.
- The interactive *responder* algorithm $\mathcal{R}$ takes as input associated data $ad \in AD$ and a password $pw \in W$. It alternates between sending and receiving messages. Eventually, it either outputs a session key k or $\perp$ signifying failure.

Consider an instance of $\mathcal{I}(ad, pw)$ that sends and receives a sequence of messages $(s_1, s_2, \ldots, s_n)$. The *transcript* tr of the instance is $(ad, pw, s_1, s_2, \ldots, s_n)$. The transcript of an instance $\mathcal{R}(ad, pw)$ is defined in the same way.

We require that for any $pw \in W$ and $ad \in AD$, then for any two instances $\mathcal{I}(ad, pw)$ and $\mathcal{R}(ad, pw)$ that both output keys and have identical transcripts, they will output identical keys.

Remark 3. The above definition does not capture asymmetry. It could be captured as follows: An additional algorithm would on input of a password and some password-associated data output a server password representation and a (public) *hint* (e.g. user name, nonce, etc.). The initiator algorithm would take associated data, password and hint as input. The responder algorithm would take associated data and server password representation as input.

We base our security definition on definitions for ordinary key exchange. There are many ways to define security for ordinary key exchange in the literature, but which exact basis we use is not important. Somewhat arbitrarily, we use a definition from a recent textbook [15]. A key exchange security definition essentially consists of an experiment together with notions of partnering, authentication and freshness.

There are two design choices in our security definition that need explanation. Both are related to the standard implicit-to-explicit upgrade strategy via key confirmation tags.

The first design choice is that each password-authenticated key exchange protocol has a session key set consisting of n_p-tuples, so $K = K_1 \times \cdots \times K_{n_p}$. The idea is that each session key is actually a tuple of session keys and these session keys are somehow independent. This is important to support our design strategy.

The second design choice is to model session key confidentiality as a blend of real-or-random and one-way adversary goals. The idea is that the upgrade strategy forces the adversary to guess part of the session key, which is a one-way security goal.

The Experiment As in Section 3.1, we shall consider a two-part adversary: a password generator $\mathcal{G}$ and an opponent $\mathcal{A}$. Again, we shall assume that they are playing an unknown, password-related game and that they are free to communicate.

The passwords the experiment uses come from password queries made by $\mathcal{G}$. The adversary may reveal these long-term keys, as well as instance session keys.

The experiment's main task is to allow the opponent to interact with instances of the password-authenticated key exchange algorithms. The opponent starts instances through the execute query, and also functions as the network and sends messages to the instances through the send query. Since we work in a multi-password model, the adversary needs to specify which generated password a particular instance should use. The adversary does this simply by specifying that the jth generated password should be used. Likewise, instances are numbered in the order they are started.

As usual for key exchange, we model confidentiality of session keys as real-or-random security, with a test query that gives the adversary either session keys or random keys, as determined by a secret bit. The adversary is allowed to do multiple test queries, and the experiment uses the same secret bit for all the queries.

We shall also allow the adversary to *guess* session keys, which models one-way security. This is not generally useful for applications, but we need it for the implicit-to-explicit design strategy.

As mentioned initially, the protocol specification includes a description of the session key set as a set of tuples, $K = K_1 \times \cdots \times K_{n_p}$. The test, guess and session key reveal queries all specify which part of the key they apply to. The adversary may test one part, guess another part and reveal a third part. Note that n_p may be 1.

Remark 4. Unlike for key exchange, we cannot reveal the state and the session key at the same time. The session key is a function of the password, the randomness and the messages received. This means that if we give the adversary both the randomness and the session key, the adversary can do an offline attack. Of course, we could have opted to give the adversary the randomness or the session

key, but not both. However, for many protocols the messages are independent of the password, so this would not give the adversary any extra power. Also, it does not actually model the real world very well. So we do not bother.

Partnering We use matching conversations for partnering. Consider an instance with transcript $(ad, pw, s_1, \ldots, s_n)$ that output a session key. A second instance is a *partner* of the first if either their transcripts are equal and the second instance output a session key, or the second instance did not output a session key, its transcript is $(ad, pw, s_1, \ldots, s_{n-1})$ and the second instance sent s_{n-1}.

Partnering is not symmetric. An instance that has not output a session key may be someone's partner, but cannot have a partner.

Authentication Implicit authentication for ordinary key exchange says that an instance can have at most one partner.

Explicit authentication for ordinary key exchange requires the presence of a partner. This does not work for us. We want to distinguish between cryptographic attacks and misuse of the password. But we do not know which passwords the adversary knows.

We resolve this as follows. *Authentication* holds if no instance has more than one partner. The adversary's *explicit success rate* is the number of passwords for which there is an instance without a partner that finished while the password was unrevealed. The idea is that this success rate should be accounted for by adversarial password guessing, which we do by showing that the opponent can be turned into a pure opponent with at least the same success rate.

We also want to support an implicit-to-explicit design strategy. This is what the guess query is for. The adversary's *implicit success rate* is the number of passwords for which there is a successful guess query while the password was unrevealed. Any such successes should be accounted for by adversarial password guessing.

The design strategy idea is that in order to break explicit authentication when we use key confirmation tags, the adversary must present a key confirmation tag. But the tags are part of the session key, so any adversary that can present a correct key confirmation tag can also guess part of the session key.

We emphasise that guess queries are synthetic, in the sense that we have no applications in mind that can use the sort of security provided by the guess queries, unlike (say) the test query.

Remark 5. Note that with password-authenticated key exchange, the adversary is free to generate the same password for two distinct password queries. We have defined our partnering notion such that this does not create an immediate authentication attack, since the transcript only lists the password that was used, not which password query originated the password. However, in multi-password settings, this is usually not a desirable property, since it would imply that the system could confuse two users who happen to have the same password.

The real-or-random experiment proceeds as follows:

1. Sample $\beta, \beta'' \xleftarrow{r} \{0,1\}$.
2. On the jth *password* query pw from $\mathcal{G}$, do:
   (a) Record (j, pw) and send j.
3. On the ith *execute* query (ρ, j, ad) from $\mathcal{A}$, do:
   (a) If (j, pw) is not recorded, send $\perp$ and stop.
   (b) If $\rho = \mathcal{I}$, start the ith instance as $\mathcal{I}(ad, pw)$. When the instance sends s, send (i, s). Otherwise, start the ith instance as $\mathcal{R}(ad, pw)$ and send i.
4. On the *send* query (i, s) from $\mathcal{A}$, do
   (a) If $(i, \cdot, \cdot)$ is recorded, send $\perp$.
   (b) Otherwise, send s to the ith instance. If the ith instance outputs $\perp$, record $(i, \perp, \perp)$ and send $(i, \perp)$. If the ith instance outputs $k_0 \in K$, sample $k_1 \xleftarrow{r} K$ and record (i, k_0, k_1). If the instance sent a message s' and did not output a key, send (i, s'). If the instance sent a message s' and output a key, send $(i, \top, s')$. If the instance output a key and did not send a message, send $(i, \top)$.
5. On the *test* query (test, i) from $\mathcal{A}$, do:
   (a) If (i, s_0, s_1) is recorded, send (i, s_β).
6. On the *guess* query $(\mathsf{guess}, i, j, k_j)$ from $\mathcal{A}$, do:
   (a) If $(i, (\cdots, k_j, \cdots), \cdot)$ is recorded, send $\top$, otherwise $\perp$.
7. On the *reveal* query (x, i) from $\mathcal{A}$, do:
   (a) If $\mathsf{x} = \mathsf{session}$ (*session reveal*) and (i, s_0, s_1) is recorded, send (i, s_0).
   (b) If $\mathsf{x} = \mathsf{ltk}$ (*long-term key reveal*) and (i, pw) is recorded, send pw.
   (c) Otherwise, send $\perp$.

Eventually, the opponent $\mathcal{A}$ outputs $\beta' \in \{0,1\}$.

Figure 3: Experiment $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}$ for the real-or-random game for a password-authenticated key exchange protocol $\mathrm{PAKE} = (K, W, AD, \mathcal{I}, \mathcal{R})$ with session key set $K = K_1 \times \cdots \times K_{n_p}$. The experiment interacts with a password generator $\mathcal{G}$ and an opponent $\mathcal{A}$. The bit β'' is not used in the experiment, but is used to simplify the calculation of advantage.

We say that an adversary is *context-respecting* if it never issues an execute query for two distinct passwords (that is, the js in the queries are distinct) with the same associated data. In a single-password setting, this trivially satisfied. In a multi-password setting, this is usually a reasonable restriction, since some information must be available to uniquely identify which password is in use, so that the server can use the correct password. Another option that may achieve the same effect is to use a per-password password-based key derivation function *pbkdf* in the design of schemes, as discussed in Section 3.1.

Freshness Freshness decides which keys have been trivially compromised. If the adversary tests these keys, we do not count that as winning. However, as for authentication, the adversary may know passwords and therefore also session keys. And we cannot in advance know which passwords the adversary knows. The end result is that we shall only allow instances with partners to have fresh session keys.

Remark 6. There is no guarantee of confidentiality for session keys of instances that do not have partners. The expectation is that these session keys are trivially known to the adversary because of password misuse, and we account for this through authentication. This means that some standard confidentiality attacks fail, but since they are counted as authentication attacks, that is ok.

The instances output a tuple of independent session keys. Freshness therefore considers session keys, not instances, unlike our partnering and authentication definitions.

A session key is *trivially compromised* if a session key reveal query has been made for that session key, or both a test and a guess query has been made for that session key. A session key is *exposed* if it is trivially compromised or a test or guess query has been made for that session key. A session key is *fresh* if it is not trivially compromised and its instance has a partner instance where the corresponding session key is not exposed. An execution is *fresh* if every session key that receives a test query is fresh.

Remark 7. We only allow one guess query per instance, though see Section 3.3.

The Definition We define advantage and success rates.

Definition 6. A (τ, n_t, n_u, n_s)-*adversary* against a password-authenticated key exchange protocol PAKE is a pair of interactive algorithms $(\mathcal{G}, \mathcal{A})$ that interacts with the experiment in Figure 3, where $\mathcal{G}$ makes at most n_u password queries, $\mathcal{A}$ makes at most n_t test queries and n_s execute and guess queries, and where the runtime of the adversary and the experiment is at most τ.

The *advantage* of the adversary $(\mathcal{G}, \mathcal{A})$ against PAKE is

$$\mathbf{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{A}) = \max\{2|\Pr[E_c] - 1/2|, \Pr[E_a]\},$$

where E_c is the event that the adversary's guess equals β if the execution is fresh, or that the adversary's guess equals β'' if the execution is not fresh; and

E_a is the event that authentication does not hold for some instance that outputs a session key.

The *implicit success rate* $\mathbf{SuccR}^{\mathsf{pake\text{-}i}}_{\text{PAKE}}(\mathcal{G}, \mathcal{A})$ of the adversary $(\mathcal{G}, \mathcal{A})$ is the number of instances using distinct, password indexes for which a successful guess happens while the password is unrevealed. The *explicit success rate* $\mathbf{SuccR}^{\mathsf{pake\text{-}e}}_{\text{PAKE}}(\mathcal{G}, \mathcal{A})$ of the adversary $(\mathcal{G}, \mathcal{A})$ is the number of instances using distinct, password indexes that, without having a partner, output a session key while the password is unrevealed.

Let $\mathcal{X}$ be a probability space on W. A *simple* (τ, n_t, n_u, n_s)*-adversary for* $\mathcal{X}$ against a password-authenticated key exchange protocol PAKE is an interactive algorithm $\mathcal{A}$ such that $(\mathcal{G}_{\mathcal{X}}, \mathcal{A})$ is a (τ, n_t, n_u, n_s)-adversary against PAKE.

Remark 8. Morally speaking, we should expect that any honestly generated passwords are independent of any cryptography in use. In particular, in the random oracle model, we will not allow the password generation algorithm $\mathcal{G}$ to make any queries to any random oracles. This is just a modelling issue and does not impact adversarial power, since the generator is free to ask the opponent to evaluate hashes.

Example 7. For any password game and opponent $(\mathcal{G}, \mathcal{A})$, there exists a corresponding pure adversary $(\mathcal{G}, \mathcal{B})$ against password-authenticated key exchange. The pure opponent $\mathcal{B}$ runs a copy of $\mathcal{A}$ and forwards any messages to and from $\mathcal{G}$. When $\mathcal{A}$ makes a reveal query, $\mathcal{B}$ makes a corresponding reveal query. When $\mathcal{A}$ makes a guess query pw for the jth password, $\mathcal{B}$ privately starts an instance of $\mathcal{R}(ad, pw)$ for some ad, makes an execute query $(\mathcal{I}, j, ad)$ and lets the two instances exchange messages. If both instances output a session key, then $\mathcal{B}$ reveals the session key of the initiator instance, and if the two session keys are identical, $\mathcal{B}$ responds with $\top$ to $\mathcal{A}$. Otherwise, it responds with $\bot$.

Unless the particular password-authenticated key exchange protocol allows two instances using distinct passwords to agree on the same session key with non-trivial probability (which means that the protocol is trivially useless), it is clear that the above is a useful adversary whose success rate is essentially equal to the success rate of the password adversary.

The example shows that the class of password guessers is essentially included in the class of realistic adversaries against a password-authenticated key exchange scheme. These are the *pure* adversaries. What we want to show is that pure adversaries make up essentially the entire realistic class.

At this point, it is instructive to consider a bad password-authenticated key exchange protocol and see how the success of an adversary against that protocol cannot be accounted for by password guessing.

Example 8. Let $\text{PAKE} = (G, G, AD, \mathcal{I}, \mathcal{R})$ be the protocol where $(\mathcal{I}, \mathcal{R})$ is the ordinary Diffie-Hellman key exchange protocol.

It is easy to show that any adversary against PAKE that has non-trivial advantage leads to an adversary against DDH with essentially the same advantage (up to a factor $n_s^2 n_t$), because authentication and freshness only consider instances with partners, which is the same as unauthenticated key exchange.

Consider, however, the following simple $(\tau, 0, 1, 1)$-adversary for the uniform password distribution on G: It generates one password, runs an instance of the responder algorithm privately, makes one execute query for this password with some associated data, and then has the two instances exchange messages.

Clearly, τ is trivial. Likewise, this adversary achieves one success with probability 1. However, any simple $(\tau', 1, 1)$-password guesser for the uniform password distribution on G achieves one success with probability at most $1/|G|$, regardless of τ'.

Remark 9. The usual claims that a single test query suffices and that a single password query suffices are, in some sense, true. It is not obvious that such results are useful and *we do not prove them.*

It is important to prove these results in the correct order, which is to deal with the single test query first. The single test theorem is done through a standard hybrid argument. There is one minor difficulty, which is that an execution can be non-fresh with many test queries, but would be fresh if some of the test queries were reveal queries. However, the simulator can notice when the simulated execution would become non-fresh and output a random bit.

A single password query suffices when considering the advantage. (This changes the game between the generator and opponent.) We guess which password will be used for the advantage. The experiment deals with that password. We simulate a second experiment that deals with all the other passwords. This strategy relies crucially on there being a single test query. The success rate information is mostly lost when considering a single password.

Remark 10. Our definition does not capture the properties of asymmetric password-authenticated key exchange, where the responder has a different password representation than the client. We would model this by having two variants of the long-term key reveal, one for the initiator representation and one for the responder representation. We then change the definition of authentication to count a responder instance without a partner towards the success rate, even if the responder representation has been revealed.

For simple adversaries using high-entropy password distributions, such a protocol could be considered as an ordinary key exchange protocol, where the server representation is the public key and the password is the secret key.

To model other adversaries, we would introduce two types of guess queries into our password guessing experiment, offline and online. The adversary's success rate must then include information about what type of query was used to find a particular password. In the random oracle model, we can register adversarial password guesses whenever certain queries to *pbkdf* are made.

When [19] the offline guesses are made, namely whether they are before or after the reveal of the responder password representation, can also be interesting, and can be measured using this approach.

3.3 Implicit to Explicit

We now show that the standard design strategy of adding tags that confirm knowledge of the key is sound. The proof is straight-forward, providing evidence that our definitions are sound. The compiler we use is essentially identical to Example 10.8 in [15], but the strategy is well established [27].

Example 9. Let $\text{PAKE}_0 = (K_0, W, AD, \mathcal{I}_0, \mathcal{R}_0)$ be a password-authenticated key exchange scheme with $K_0 = K \times T^2$. The password-authenticated key exchange scheme $\text{PAKE} = (K, W, AD, \mathcal{I}, \mathcal{R})$ works as follows:

Let $\mathcal{P}$ and $\mathcal{P}_0$ denote either $\mathcal{I}$ and $\mathcal{I}_0$, or $\mathcal{R}$ and $\mathcal{R}_0$.

The interactive algorithm $\mathcal{P}$ runs the $\mathcal{P}_0$ algorithm with its own input. If $\mathcal{P}_0$ ever outputs $\perp$, $\mathcal{P}$ immediately outputs $\perp$ and stops. Any messages $\mathcal{P}_0$ sends $\mathcal{P}$ sends, and $\mathcal{P}$ forwards any received messages to $\mathcal{P}_0$, with the following modifications:

- If $\mathcal{P}_0$ sends a message s and outputs a session key $(k, t_0, t_1) \in K_0$, $\mathcal{P}$ remembers (k, t_1) and sends the pair (s, t_0).
- If $\mathcal{P}$ receives a pair (s, t_0'), it sends s to $\mathcal{P}_0$. If $\mathcal{P}_0$ sends any message, $\mathcal{P}$ outputs $\perp$. If $\mathcal{P}_0$ does not send a message and outputs a session key (k, t_0, t_1) with $t_0' = t_0$, then $\mathcal{P}$ sends t_1 and outputs k. Otherwise, $\mathcal{P}$ outputs $\perp$.
- If $\mathcal{P}_0$ outputs (k, t_0, t_1) and $\mathcal{P}$ receives a tag t_1', then it outputs k if $t_1 = t_1'$, otherwise it outputs $\perp$.

Remark 11. Constructions using PRFs or MACs are common in the literature. This is significantly more complicated.

Proposition 10. *Consider* PAKE *and* PAKE_0 *as in Example 9, with keys split into three parts. Let* $(\mathcal{G}, \mathcal{A})$ *be a* (τ, n_t, n_u, n_s)*-adversary against a password-authenticated key exchange scheme* PAKE*. Then there exists a* (τ, n_t, n_u, n_s)*-adversary* $(\mathcal{G}, \mathcal{B})$ *against* PAKE_0 *such that*

$$\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\text{PAKE}}(\mathcal{G}, \mathcal{A}) \leq \boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\text{PAKE}_0}(\mathcal{G}, \mathcal{B}) \text{ and}$$
$$\boldsymbol{SuccR}^{\mathsf{pake\text{-}i}}_{\text{PAKE}_0}(\mathcal{G}, \mathcal{B}) \leq \boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}(\mathcal{G}, \mathcal{A}).$$

If $(\mathcal{G}, \mathcal{A})$ *is context-respecting, so is* $(\mathcal{G}, \mathcal{B})$.

Proof. The adversary $\mathcal{B}$ runs a copy of $\mathcal{A}$ and a simulator **Sim** that runs a copy of the PAKE experiment, modified as follows:

- Instead of each PAKE instance running a PAKE_0 instance, **Sim** delegates the PAKE_0 instances to the PAKE_0 experiment.
- When a PAKE instance would send a pair (s, t_0), **Sim** reveals the second session key of the corresponding PAKE_0 instance to get t_0.

- When a PAKE instance receives a pair (s, t'_0) and the corresponding PAKE_0 instance outputs a session key, **Sim** guesses t'_0 for the second session key of the corresponding PAKE_0 instance. If the experiment returns $\perp$, the PAKE instance outputs $\perp$. Otherwise, **Sim** reveals the third session key t_1 of the corresponding PAKE_0 instance and has the PAKE instance send t_1. Also, **Sim** acts as if the PAKE instance has output a session key.

- When a PAKE instance receives a tag t'_1 and the corresponding PAKE_0 instance output a session key, **Sim** guesses t'_1 for the third session key of the corresponding PAKE_0 instance. If the experiment returns $\perp$, the PAKE instance outputs $\perp$. Otherwise, **Sim** acts as if the the PAKE instance output a session key.

- When $\mathcal{A}$ makes a test, guess or reveal query for some PAKE instance, **Sim** makes the corresponding query for the first session key of the corresponding PAKE_0 instance and forwards the response.

- When $\mathcal{A}$ makes a long-term reveal query for a password, **Sim** makes a long-term reveal query for the same password, forwarding the response.

It is straight-forward, though somewhat tedious, to verify that $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\text{PAKE}_0}$ together with **Sim** behave in identical fashion to $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\text{PAKE}}$.

If some PAKE instance has two or more partners, then trivially, the corresponding PAKE_0 instance will have the same number of partners. In other words, the probability that $(\mathcal{G}, \mathcal{A})$ breaks authentication for PAKE is at least as great as the probability that $(\mathcal{G}, \mathcal{B})$ breaks authentication for PAKE_0. We do not get equality, because it could be that the PAKE_0 instance has more than one partner, but one or more never output a session key, and so do not count as PAKE partners.

If a PAKE session key is fresh, then the instance has a partner where the session key is not exposed, which means that the corresponding PAKE_0 instance has a partner where the first session key is not exposed. Also, no test or guess queries have been made for the session key. It follows that the first PAKE_0 session key is also fresh. Conversely, if a PAKE session key is not fresh, then the corresponding first PAKE_0 session key is also not fresh. From which it follows that the probability that $(\mathcal{G}, \mathcal{B})$ guesses the correct challenge bit is identical to the probability that $(\mathcal{G}, \mathcal{A})$ guesses the correct challenge bit.

Next, suppose there is a PAKE instance without a partner. By design of **Sim**, this can only happen if **Sim** made a successful guess query for the second or third PAKE_0 session key. In other words, the implicit success rate of $(\mathcal{G}, \mathcal{B})$ at least equals the explicit success rate of $(\mathcal{G}, \mathcal{A})$. Note that the success rates need not be identical, since any successful guess query made by $\mathcal{A}$ would not be counted towards the explicit success rate, but would be counted towards its implicit success rate. □

Remark 12. When $K = K_1 \times \cdots \times K_{n_p}$, seems obvious that it should not matter if we consider the symmetric key to consist of one part or n_p parts.

It is obvious that a single-part adversary immediately becomes a multi-part adversary because test, guess and reveal queries can be simulated with queries for each part. A multi-part adversary does not immediately become a single-part adversary, however, because the multi-part adversary may succeed with a guess query for one part of the key, and there is no way to recover the entire key from the one part.

We could attempt to simulate such a result by deriving the multiple key parts from a single key part using a key derivation function that we model as a random oracle. The technical issues not-withstanding, such a construction seems redundant, so we shall not prove such a result. Instead, we work with multi-part adversaries against the underlying password-authenticated key exchange directly.

3.4 CPace Variant Analysis

The following protocol is equivalent to the CPace protocol [16, 3, 4], except that we include the password in the session key derivation.

Example 11. The protocol CPACE' $= (K, W, AD, \mathcal{I}, \mathcal{R})$, based on the group G and functions $kdf : AD \times W \times G^3 \to K$ and $pbkdf : W \to G \setminus \{1\}$, works as follows:

- The *initiator* algorithm $\mathcal{I}$ takes ad and $pw \in W$ as input, samples $a \xleftarrow{r} \{1, 2, \ldots, p-1\}$, computes $x \leftarrow pbkdf(pw)^a$ and sends x. When it receives the message $y \neq 1$, it computes $k \leftarrow kdf(ad, pw, x, y, y^a)$ and outputs k.
- The *responder* algorithm $\mathcal{R}$ takes ad and $pw \in W$ as input and samples $b \xleftarrow{r} \{1, 2, \ldots, p-1\}$. When it receives the message $x \neq 1$, it computes $y \leftarrow pbkdf(pw)^b$, $k \leftarrow kdf(ad, pw, x, y, x^b)$, sends y and outputs k.

Remark 13. Sometimes, the actual hash function $pbkdf : W \to G$ we use does not induce a uniform distribution on G. Abdalla *et al.* [3] have shown that how to resolve this using rejection sampling, under reasonable conditions. Essentially, we need a few more oracle queries of all types, and the cost of choosing the hash values increases somewhat. Adapting the proof of the following result to use rejection sampling is relatively straight-forward.

Proposition 12. *Consider the scheme* CPACE' *with* $K = K_1 \times K_2 \times K_3$, *with* $1/|K_i| \leq \delta$ *for* $i = 1, 2, 3$. *Let* $(\mathcal{G}, \mathcal{A})$ *be a context-respecting* (τ, n_t, n_u, n_s)*-adversary against* CPACE' *in the random oracle model, making at most* n_h *queries to the pbkdf and kdf oracles. Then there exists a* (τ_1', n_h, n_s, n_h)*-solver* $\mathcal{B}_1$ *for* StSDHF, *a* (τ_2', n_h, n_s, n_h)*-solver* $\mathcal{B}_2$ *for* StCDH *and a* (τ_3', n_u, n_s)*-password guesser* $(\mathcal{G}, \mathcal{B}_3)$, *with* τ_1', τ_2' *and* τ_3' *all equal to a small multiple of* τ, *such that*

$$\boldsymbol{Adv}^{\text{pake-ror}}_{\text{CPACE}'}(\mathcal{G}, \mathcal{A}) \leq \frac{n_s^2}{|G|} + \epsilon \text{ and}$$
$$\boldsymbol{SuccR}^{\text{pw}}(\mathcal{G}, \mathcal{B}_3) \leq \boldsymbol{SuccR}^{\text{pake-i}}_{\text{CPACE}'}(\mathcal{G}, \mathcal{A}) + \epsilon + n_s\delta$$

where

$$\epsilon = (n_h(n_h+1)+n_s)/|G| + \boldsymbol{Adv}_G^{\mathsf{StSDHF}}(\mathcal{B}_1) + \boldsymbol{Adv}_G^{\mathsf{StCDH}}(\mathcal{B}_2).$$

The idea is that the adversary would have to find valid DDH tuples for multiple bases in order to reject more than one password guess based on an instance session key, which solves the StSDHF problem. For instances that have a partner, the adversary would have to solve an instance of the Computational Diffie-Hellman problem for a random base. For both of these cases, we will need to recognise some DDH tuples, which we can do if we have a restricted DDH oracle, so we need the strong version of the two problems. After excluding these two cases, we can simulate the remaining interactions with the adversary without knowledge of instance passwords. The result is a password guesser, which gives us a pure opponent.

Proof of Proposition 12 We first bound the probability of E_a happening. Since we use matching conversations, this can only fail if two or more instances use the same randomness. Since the randomness is sampled from the uniform distribution and independently, the birthday paradox applies and we get the bound

$$\Pr[E_a] \leq \frac{n_s^2}{|G|}. \tag{1}$$

The bound on E_c is structured as a sequence of games. Let $E_{c,i}$ be the event that the adversary outputs the correct guess in Game i (for β if the execution is fresh, for β'' if not).

We must also keep track of any possible loss of implicit success rate throughout the sequence of games. We let SR_i denote the probability space given by the adversary's implicit success rate in Game i.

The first three games implement some technical changes. The fourth game restricts random oracle queries to prevent the adversary from deriving more than one hypothetical session key per instance, by restricting access to the random oracle. This gives us an adversary against the StSDHF problem.

The fifth game restricts random oracle queries to prevent the adversary from deriving the session key of instances that have a partner. This gives us an adversary against the StCDH problem.

Finally, we observe that unless the adversary makes oracle queries corresponding to successful password guesses, the adversary does not evaluate the hash oracle at the session key points, and consequently cannot reliably make guess queries or say anything about test queries. This gives us a password guesser that succeeds at its password guesses at least as often as the adversary succeeds at guess queries.

For a more compact notation, we denote the value $\mathit{pbkdf}(pw) \in G$ by g_{pw}.

Game 0 The initial game is the experiment interacting with the adversary $(\mathcal{G}, \mathcal{A})$. Then

$$\Pr[E_c] = \Pr[E_{c,0}] \quad \text{and} \quad \mathbf{SuccR}^{\mathsf{pake\text{-}i}}_{\text{CPACE}}(\mathcal{G}, \mathcal{A}) = SR_0. \tag{2}$$

Let $\tau_0 = \tau$.

Game 1 In this game, when honest instances sample exponents, we sample them from $\{0, 1, \ldots, p-1\}$. If any instance samples 0, the game stops.

The probability that any one instance causes the game to stop is $1/|p|$ and we get that

$$|\Pr[E_{c,0}] - \Pr[E_{c,1}]| \leq \frac{n_s}{|G|} \quad \text{and} \quad SR_1 \leq SR_0 + \frac{n_s}{|G|} \tag{3}$$

The runtime τ_1 is essentially the same as τ_0.

Remark 14. We need this modification because the honest instances cannot accept 1 as a message, but the underlying computational problems sample group elements from the uniform distributions on the group, which includes 1. We cannot change the scheme. We could change the underlying problems, but it seems easier to do this game.

Game 2 In this game, the random oracle *pbkdf* does not sample group elements as $g_{pw} \xleftarrow{r} G$, but rather as $r \xleftarrow{r} \{0, 1, \ldots, p-1\}$ and $g_{pw} \leftarrow g^r$, for some fixed generator g. We also stop if $g_{pw} = g_{pw'}$ for $pw \neq pw'$.

The birthday paradox applies and we get that

$$|\Pr[E_{c,1}] - \Pr[E_{c,2}]| \leq \frac{n_h^2}{|G|} \quad \text{and} \quad SR_2 \leq SR_1 + \frac{n_h^2}{|G|} \tag{4}$$

This game requires some extra accounting for each *pbkdf* hash query. Since the number of hash queries is bounded by τ, the runtime bound τ_2 is at most a small multiple of τ_1.

Remark 15. Note that when either x or y were sent by an honest instance with password pw, the experiment knows the discrete logarithm to base g_{pw} and can now decide if $(g_{pw'}, x, y, z)$ is a DDH tuple or not, even if $g_{pw} \neq g_{pw'}$, as long as $g_{pw'}$ was output by the *pbkdf* oracle.

Game 3 In this game, we stop if when an instance computes a message x, the random oracle *kdf* has seen a query of the form $(\cdot, \cdot, x, \cdot, \cdot)$ or $(\cdot, \cdot, \cdot, x, \cdot)$.

Since x is sampled from the uniform distribution on G and there have been at most n_h queries to *kdf*, we get that

$$|\Pr[E_{c,2}] - \Pr[E_{c,3}]| \leq \frac{n_h}{|G|} \quad \text{and} \quad SR_3 \leq SR_2 + \frac{n_h}{|G|}. \tag{5}$$

This game requires some extra accounting for each hash query. Since the number of hash queries is bounded by τ, the runtime bound τ_3 is at most a small multiple of τ_2.

Game 4 In this game, we introduce some extra accounting into the random oracle *kdf*. For each query (ad, pw, x, y, z), pw queried to the *pbkdf* oracle, that results in a response k, the oracle stores a tuple $(ad, pw, x, y, z, k, ind)$, where *ind* is $\perp$ if the query is made by $\mathcal{A}$ and $\top$ otherwise.

The changes in this game are not observable, so

$$\Pr[E_{c,4}] = \Pr[E_{c,3}] \quad \text{and} \quad SR_4 = SR_3. \tag{6}$$

The runtime bound τ_4 is essentially equal to τ_3.

Remark 16. Note that in the games, the last flag denotes who made the hash oracle query, either the adversary ($\perp$) or the experiment ($\top$). In the reductions, the flag will typically denote a DDH tuple (or at least an intended DDH tuple) or a non-DDH tuple.

Game 5 In this game, we stop if the random oracle *kdf* ever records two tuples $(\cdot, pw_1, x, y, z_1, \cdot, \perp)$ and $(\cdot, pw_2, x, y, z_2, \cdot, \perp)$, where $pw_1 \neq pw_2$, pw_1 and pw_2 have both been queried the *pbkdf* oracle, some honest instance sent x or y, and (g_{pw_1}, x, y, z_1) and (g_{pw_2}, x, y, z_2) are both DDH tuples.

Let F_i be the event that this happens in Game i, $i = 4, 5$. Since the games proceed identically until F_i happens, we get that $\Pr[F_5] = \Pr[F_4]$, and that

$$|\Pr[E_{c,5}] - \Pr[E_{c,4}]| \leq \Pr[F_5] \quad \text{and} \quad SR_5 \leq SR_4 + \Pr[F_5]. \tag{7}$$

This game requires some extra accounting for each hash query. Since the number of hash queries is bounded by τ, the runtime bound τ_5 is at most a small multiple of τ_4.

Lemma 13. *There exists a* (τ_1', n_h, n_s, n_h)*-solver* $\mathcal{B}_1$ *for* StSDHF*, with* τ_1' *essentially equal to* τ_5*, such that*

$$\boldsymbol{Adv}_G^{\mathsf{StSDHF}}(\mathcal{B}_1) = \Pr[F_5].$$

Proof. The adversary $\mathcal{B}_1$ runs a copy of $\mathcal{A}$ and a simulation **Sim** of the random oracles *pbkdf* and *kdf* interacting with a copy of the experiment $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\mathrm{CPACE}'}$, modified as follows:

- When a query to *pbkdf* is made and the oracle would sample a response, **Sim** makes a query to its first sampling oracle and uses the response as the sampled value.
- When an instance starts, **Sim** makes a query to its second sampling oracle and uses the response as its message.
- When an instance with transcript (ad, pw, x, y) wants to compute its session key, **Sim** first checks if there is a tuple $(ad, pw, x, y, \cdot, k, \top)$ recorded. If there is, it uses k as its session key. If not, it samples $k \xleftarrow{r} K$ and records $(ad, pw, x, y, -, k, \top)$.

- When $\mathcal{A}$ makes a new query (ad, pw, x, y, z) to the random oracle *kdf* such that pw was returned by the *pbkdf* oracle and some instance has sent either x or y, then **Sim** uses its third oracle to decide if (g_{pw}, x, y, z) is a DDH tuple. If it is not, **Sim** samples $k \xleftarrow{r} K$, records $(ad, pw, x, y, z, k, \bot)$ and returns k. If it is a DDH tuple and there is a record $(ad, pw, x, y, -, k, \top)$, **Sim** updates the record to $(ad, pw, x, y, z, k, \top)$ and returns k. Otherwise, **Sim** samples $k \xleftarrow{r} K$, records $(ad, pw, x, y, z, k, \top)$ and returns k.

If two tuples $(\cdot, pw_1, x, y, z_1, \cdot, \top)$ and $(\cdot, pw_2, x, y, z_2, \cdot, \top)$ are recorded, with $pw_1 \neq pw_2$ and some instance sent x or y, $\mathcal{B}_1$ outputs the corresponding indexes and group elements and stops. Otherwise, $\mathcal{B}_1$ ends when $\mathcal{A}$ outputs its bit.

Simulating *pbkdf* using the first oracle of $\mathcal{B}_1$ does not introduce any changes, since the oracle samples its values from the uniform distribution on G, which the *pbkdf* random oracle would also have done.

The main simulation difficulty is that the underlying game knows the discrete logarithms of the messages sent by instances, but the simulation does not. In particular, this means that the simulation does not know z for its own instances.

This difficulty is resolved by using lazy evaluation of the hash oracle input combined with ensuring consistency in query responses by using the Strong Diffie-Hellman oracle provided by the StSDHF problem. Lazy evaluation is done in the third bullet point where we choose a value for the session key without knowing the entire hash input. Consistent evaluation is ensured by the fourth bullet point. We already forbid instances from using messages that had previously been queried to *kdf*. This simplifies the accounting in the reduction.

It therefore follows that until $\mathcal{B}_1$ stops, it perfectly simulates Game 5 (and Game 4). It only stops if the event F_5 happens. When this event happens, $\mathcal{B}_1$ stops and its output corresponds to two tuples that satisfy the StSDHF problem.

The adversary $\mathcal{B}_1$ makes at most as many queries to its first oracle as there are hash queries, it makes as many queries to its second oracle as there are sessions, and it makes at most as many queries to its Strong Diffie-Hellman oracle as there are hash queries.

Finally, the simulation requires some extra accounting, but the runtime of $\mathcal{B}_1$ and its experiment is essentially the same as the runtime of Game 5. □

Game 6 In this game, we stop if the adversary makes a *kdf* query (ad, pw, x, y, z) such that pw has been queried to *pbkdf*, some responder instance has transcript $(ad, \cdot, x, y)$, some initiator instance has transcript $(ad, \cdot, x, \cdot)$ and (g_{pw}, x, y, z) is a DDH tuple.

Let F_i' be the event that this happens in Game i, $i = 5, 6$. Since the games proceed identically until F_i' happens, we get that $\Pr[F_6'] = \Pr[F_5']$, and that

$$|\Pr[E_{c,6}] - \Pr[E_{c,5}]| \leq \Pr[F_6'] \quad \text{and} \quad SR_6 \leq SR_5 + \Pr[F_6']. \tag{8}$$

This game requires some extra accounting for each hash query. Since the number of hash queries is bounded by τ_5, the runtime bound τ is at most a small multiple of τ_5.

Lemma 14. *There exists a* (τ_2', n_h, n_s, n_h)*-solver* $\mathcal{B}_2$ *for* StCDH*, with* τ_2' *essentially equal to* τ_6*, such that*

$$\boldsymbol{Adv}_G^{\mathsf{StCDH}}(\mathcal{B}_2) = \Pr[F_6'].$$

Proof. The adversary $\mathcal{B}_1$ runs a copy of $\mathcal{A}$ and a simulation **Sim** of the random oracles *pbkdf* and *kdf* interacting with a copy of the experiment $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\text{CPACE'}}$, modified as follows:

- When a query to *pbkdf* is made and the oracle would sample a response, **Sim** makes a query to its first sampling oracle and uses the response as the sampled value.
- When an initiator instance starts, **Sim** makes a query to its second sampling oracle and uses the response as its message.
- When a responder instance receives a message x from an initiator instance with the same associated data ad and same password pw, **Sim** makes a query to its third sampling oracle and uses the response as its message. Then it samples $k \xleftarrow{r} K$ as its session key and records $(ad, pw, x, y, -, k, \top)$.
- When an initiator instance with transcript (ad, pw, x, y) wants to compute its session key, then if there is a record $(ad, pw, x, y, \cdot, k, \top)$, it uses k as its session key. Otherwise, it samples $k \xleftarrow{r} K$ as its session key and records $(ad, pw, x, y, -, k, \top)$.
- When $\mathcal{A}$ makes a new query (ad, pw, x, y, z) to the random oracle *kdf* such that pw has been queried to the *pbkdf* oracle, some initiator instance sent x and some responder instance has transcript $(\cdot, \cdot, x, y)$, then **Sim** uses its fourth oracle to decide if (g_{pw}, x, y, z) is a DDH tuple. If it is a DDH tuple, **Sim** outputs the corresponding indices of g_{pw}, x and y, as well as z. Otherwise, **Sim** samples $k \xleftarrow{r} K$, records $(ad, pw, x, y, z, \bot)$ and returns k.
- When $\mathcal{A}$ makes a new query (ad, pw, x, y, z) to the random oracle *kdf* such that pw has been queried to the *pbkdf* oracle, some initiator instance sent x and no responder instance has transcript $(\cdot, \cdot, x, y)$, then **Sim** uses its fourth oracle to decide if (g_{pw}, x, y, z) is a DDH tuple. If it is not, **Sim** samples $k \xleftarrow{r} K$, records $(ad, pw, x, y, z, k, \bot)$ and returns k. Otherwise, if there is a record $(ad, pw, x, y, -, k, \top)$, the record is updated to $(ad, pw, x, y, z, k, \top)$ and k is returned. Otherwise, **Sim** samples $k \xleftarrow{r} K$, records $(ad, pw, x, y, z, k, \top)$ and returns k.

Simulating *pbkdf* using the first oracle of $\mathcal{B}_2$ does not introduce any changes, since the oracle samples its values from the uniform distribution on G, which the *pbkdf* random oracle would also have done.

Again, the main difficulty is that the underlying game knows the discrete logarithms of the messages sent by instances, but the simulation does not. In particular, this means that the simulation does not know z for its own instances.

This difficulty is resolved by using lazy evaluation of the hash oracle input combined with ensuring consistency in query responses by using the Strong Diffie-Hellman oracle provided by the StCDH problem. Lazy evaluation is done in the third and fourth bullet points, where we choose consistent values for the session key without knowing the entire hash input. We do not know the entire hash input because we have embedded CDH challenges in the password and the protocol messages.

The fourth and fifth bullet points ensure consistent evaluation. There are two cases. The first originates with initiator instances that do not have a partner. We can recognize these tuples using the Strong Diffie-Hellman oracle. The second originates with responder instances that have potential partners, which is exactly F_6'. In this case, we have embedded a complete CDH problem in the password and the protocol messages, so if the adversary makes such a hash query, we get the solution to the CDH problem. Here too, we can recognise such hash queries using the Strong Diffie-Hellman oracle. Again, forbidding instances from using messages that have already been queried to *kdf* simplifies accounting in the reduction.

It therefore follows that until $\mathcal{B}_2$ stops, it perfectly simulates Game 6 (and Game 5). It only stops if the event F_6' happens. When this event happens, $\mathcal{B}_2$ stops and its output is a solution to the CDH challenge.

Since $\mathcal{B}_2$ makes queries to the Strong Diffie-Hellman oracle only when $\mathcal{A}$ makes queries to *kdf*, this bounds the number of Strong Diffie-Hellman queries. Since $\mathcal{B}_2$ makes a query to its first sampling oracle only to respond to *pbkdf* queries, we can bound the number of sampling queries by the number of hash queries. Finally, since $\mathcal{B}_2$ makes queries to the second and third sampling oracles only to sample messages for instances, and each instance samples at most one message, we can bound the number of sampling queries by the number of instances.

Finally, the simulation requires some extra accounting, but the runtime of $\mathcal{B}_2$ and its experiment is essentially the same as the runtime of Game 6. □

Conclusion Observe that in the final game, the adversary cannot query the *kdf* oracle at the value used for session key derivation for instances with partners, so a test query for any session key of such an instance cannot reveal any information about the challenge bit. Therefore

$$\Pr[E_{c,6}] = 1/2. \tag{9}$$

For any instance with transcript (ad, pw, x, y), where either x or y were adversary-generated, then we say that a *kdf* query (ad, pw, x, y, z) where (g_{pw}, x, y, z) is a DDH tuple is *password-revealing*. Let $\mathcal{Y}$ denote the number of distinct passwords that such password-revealing queries happen for.

Unless the adversary makes a password-revealing query, it cannot distinguish the instance session key from a random key, which means that it cannot make a successful guess query for that instance, except with probability at most δ. Therefore, unless a password-revealing query happens for a particular password,

that password cannot see a successful guess query except with probability $n'_s\delta$, where n'_s is the number of instances started with that particular password. Hence, except with probability $n_s\delta$, the number of distinct passwords that will see successful guess queries is upperbounded by $\mathcal{Y}$, so SR_6 is $n_s\delta$-near $\mathcal{Y}$.

$$\mathcal{Y} \leq SR_6 + n_s\delta. \tag{10}$$

Proposition 12 then follows from equations (1)–(10) and Lemmas 13, 14 and 15.

Lemma 15. *There exists a (τ'_3, n_u, n_s)-password guesser $(\mathcal{G}, \mathcal{B}_3)$, with τ'_3 essentially equal to τ_6, such that*

$$\boldsymbol{SuccR}^{\mathsf{pw}}(\mathcal{G}, \mathcal{B}_3) = \mathcal{Y}.$$

Proof. The opponent $\mathcal{B}_3$ runs a copy of $\mathcal{A}$ and a simulation **Sim** of the random oracles *pbkdf* and *kdf* interacting with a copy of the experiment in Game 6, modified as follows:

- When **Sim** is notified about the jth password, **Sim** records $(j, \perp)$.
- When **Sim** makes a guess query (j, pw) and gets a response $\top$, it replaces the record $(j, \perp)$ by (j, pw).
- Initiator instances compute their messages as $x \leftarrow g^a$, while responder instances compute their messages as $y \leftarrow g^b$.

  Note that for any $g_{pw} = g^r$ returned by *pbkdf*, any initiator instance and any y, **Sim** can compute a DDH tuple (g_{pw}, x, y, z) by computing $z \leftarrow y^{ar^{-1}}$. A similar formula applies for responder instances.
- Suppose an instance with transcript $(ad, \cdot, x, y)$ uses the jth password and wants to compute its session key.
  - If (j, pw) is recorded, **Sim** computes the correct z, samples $k \xleftarrow{r} K$ and records $(ad, pw, x, y, z, k, \top)$.
  - If $(j, \perp)$ and $(ad, pw, x, y, z, k, \top)$ are recorded, then **Sim** makes a guess query (j, pw). If the result is $\top$, it returns k. If the result is $\perp$, it samples $k' \xleftarrow{r} K$ and records $(ad, -, x, y, -, k', j)$.
  - If $(j, \perp)$ and $(ad, -, x, y, -, k, j)$ are recorded, then **Sim** uses k. (Note that unless $(\mathcal{G}, \mathcal{A})$ is context-respecting, this could cause inconsistent results.)
  - If $(j, \perp)$ is recorded, but no record $(ad, \cdot, x, y, \cdot, k, \top)$ or $(ad, -, x, y, -, k, j)$ exists, then **Sim** samples $k \xleftarrow{r} K$ and records $(ad, -, x, y, -, k, j)$.
- When $\mathcal{A}$ makes a new query (ad, pw, x, y, z) to the random oracle *kdf* such that pw has been queried to the *pbkdf* oracle and either x or y was sent by an honest instance, then **Sim** checks if (g_{pw}, x, y, z) is a DDH tuple.

– If it is not, **Sim** samples $k \xleftarrow{r} K$, records $(ad, pw, x, y, z, k, \bot)$ and returns k.

– If it is a DDH tuple, some responder instance has transcript $(ad, \cdot, x, y)$ and some initiator instance has transcript $(ad, \cdot, x, \cdot)$, **Sim** stops.

– If it is a DDH tuple, $(ad, pw', x, y, z', k', \top)$ with $pw' \neq pw$ is recorded, and some instance has transcript $(ad, \cdot, x, \cdot)$ or $(ad, \cdot, \cdot, y)$, **Sim** stops.

– If it is a DDH tuple and $(ad, -, x, y, -, k, j)$ is recorded (that is, there is a unique instance with transcript $(ad, \cdot, x, y)$ that uses the jth password), **Sim** makes a guess query of (j, pw). If the result is $\top$, the record is updated to $(ad, pw, x, y, z, k, \top)$. Otherwise, **Sim** samples $k \xleftarrow{r} K$ and records $(ad, pw, x, y, z, k, \top)$.

– Otherwise, **Sim** samples $k \xleftarrow{r} K$ and records $(ad, pw, x, y, z, k, \top)$.

We must show that $\mathbf{Exp}^{\mathsf{pw}}$ together with **Sim** perfectly simulate the experiment and the random oracles in Game 6. The only difficulty is that some simulated instances do not know their passwords. Their messages are independent of the password, so the messages are simulated perfectly. If an instance password becomes known, we can recreate the instance DDH tuple correctly.

Until the password is known, we cannot know the input to the session key derivation, which we deal with using lazy evaluation. The adversary may evaluate the hash at DDH tuples (which correspond to password guesses), but this is forbidden for instances with partners and allowed at most once for other instances. Sometimes, the adversary evaluates the hash at a DDH tuple before the instance derives its session key, which **Sim** notices and turns it into a password guess. Other times, the adversary evaluates the hash at a DDH tuple after the instance derives its session key. Again, **Sim** notices and turns it into a password guess. If the guess is correct, any records of lazy evaluation are updated to keep the simulated hash oracle consistent. Since we prohibit multiple password guesses per instance, we know that we will make at most one password guess per instance.

It follows that the simulation is perfect, that the password guesser $(\mathcal{G}, \mathcal{B}_3)$ makes at most n_u password queries, and at most one password guess per instance.

Finally, any hash query made by $\mathcal{A}$ that causes $\mathcal{Y}$ to increase corresponds exactly to a hash query that makes $\mathcal{B}_3$ guess a password, so $\mathbf{SuccR}^{\mathsf{pw}}(\mathcal{G}, \mathcal{B}_3) = \mathcal{Y}$ and the lemma follows. □

3.5 SRP-related Protocol

As we shall explain, the following protocol is related to the well-known SRP protocol.

Example 16. Let (π, π^{-1}) be a block cipher on G with key set G. The protocol SRP' $= (K, W, AD, \mathcal{I}, \mathcal{R})$, based on the group G, (π, π^{-1}) and functions $kdf : AD \times G^5 \to K \times T^2$ and $pbkdf : W \to \{0, 1, \ldots, p-1\}$, works as follows:

- The *initiator* algorithm $\mathcal{I}$ takes ad and $pw \in W$ as input, samples $a \xleftarrow{r} \{1, 2, \ldots, p-1\}$, computes $g_{pw} \leftarrow g^{pbkdf(pw)}$ and $x \leftarrow g^a$, and sends x. When it receives the message y, it computes

$$(k, t_1, t_2) \leftarrow kdf(ad, pw, x, y, \pi^{-1}(g_{pw}, y)^a, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)})$$

  and sends t_1. When it eventually receives the message t_2', it outputs k if $t_2' = t_2$. Otherwise, it outputs $\perp$.

- The *responder* algorithm $\mathcal{R}$ takes ad and $pw \in W$ as input, computes $g_{pw} \leftarrow g^{pbkdf(pw)}$ and samples $b \xleftarrow{r} \{0, 1, \ldots, p-1\}$. When it receives the message $x \neq 1$, it computes $y \leftarrow \pi(g_{pw}, g^b)$ and $(k, t_1, t_2) \leftarrow kdf(ad, pw, x, y, x^b, g_{pw}^b)$ and sends y. When it receives the message t_1', it sends t_2 and outputs k if $t_1' = t_1$. Otherwise, it outputs $\perp$.

Remark 17. We describe two changes that give us a scheme equivalent to SRP [29]. First, a slight speedup is possible. Instead of computing $\pi^{-1}(g_{pw}, x)^b$ and g_{pw}^b separately, we can compute a random linear combination as in MQV-like protocols. Second, we may not need a full-blown block cipher. For particular groups, a very simple algebraic function seems to work well, in the sense that nobody has come up with a useful attack on SRP which uses that block cipher.

It is not obvious how to give a tight proof of security after the first change, though techniques like Kiltz *et al.*[20] would be one likely approach. We have no idea how to prove security in any way after the second change.

Remark 18. The protocol SRP' can be made asymmetric by letting the responder take g_{pw} as input instead of pw, and using g_{pw} instead of pw when deriving the session key. If pw was sampled from a high-entropy distribution and g_{pw} is known by the adversary, this scheme reduces to ordinary key exchange with one-sided authentication (only the initiator is authenticated).

Remark 19. Another direction we could develop this scheme is as a mixed-authentication scheme, where the server authenticates with the password and a long-term cryptographic key.

The server has a secret key v and a public key $u = g^v$. The initiator takes u as input, while the server takes (u, v) as input. The input to *kdf* will be

$$(ad, g_{pw}, u, x, y, x^b, g_{pw}^b, x^v) = (ad, g_{pw}, u, x, y, \pi^{-1}(g_{pw}, y)^a, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}, u^a).$$

Remark 20. If we remove the exchange of tags in SRP', we get an insecure protocol, not just a protocol with implicit authentication. In particular, note that the tags are sent in the opposite order of what Example 9 would do, resulting in a protocol with four messages instead of three.

Proposition 17. *Consider the scheme SRP' from Example 16. Let $(\mathcal{G}, \mathcal{A})$ be a context-respecting (τ, n_t, n_u, n_s)-adversary against SRP' in the random oracle and the ideal cipher models, making at most n_h queries to the pbkdf, kdf*

and ideal cipher oracles. Then there exists a $(\tau_1', 0, n_s + n_h, n_h)$*-solver* $\mathcal{B}_1$*, a* (τ_2', n_h, n_s, n_h)*-solver* $\mathcal{B}_2$ *and a* (τ_3', n_u, n_s)*-password guesser* $(\mathcal{G}, \mathcal{B}_3)$*, with* τ_1'*,* τ_2' *and* τ_3' *all equal to a small multiple of* τ*, such that*

$$\begin{aligned}\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{SRP'}(\mathcal{G}, \mathcal{A}) &\leq \epsilon + \frac{n_s^2}{|G|} \text{ and} \\ \boldsymbol{SuccR}^{\mathsf{pw}}(\mathcal{G}, \mathcal{B}_3) &\leq \boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}_{SRP'}(\mathcal{G}, \mathcal{A}) + \epsilon,\end{aligned}$$

where

$$\begin{aligned}\epsilon = {} & (n_h(3n_h + 1) + n_s(n_s + n_h))/|G| + (n_h^2 + n_s)/|T| \\ & + \boldsymbol{Adv}^{\mathsf{StCDH}}_G(\mathcal{B}_1) + \boldsymbol{Adv}^{\mathsf{StCDH}}_G(\mathcal{B}_2).\end{aligned}$$

We need to extract password guesses from interactions with both the initiator and responder. The key confirmation tag derivation involves the password, which commits the adversary to a particular password guess which we can identify. But we need to identify a particular password guess. When interacting with a responder instance, unless the adversary can find two or more passwords that result in the same tag, any query made is uniquely identified by the tag. When interacting with an initiator instance, unless the adversary can find particular collisions in the block cipher (seen as a function from keys and blocks to blocks), the adversary's first message commits to a password guess, and any other guess forces the adversary to solve CDH problems.

Proof of Proposition 17 We first bound the probability of E_a happening. Since we use matching conversations, this can only fail if two or more instances send the same group element. Since the group elements are sampled from the uniform distribution (and in the responder's case, have a permutation applied to it) and independently, the birthday paradox applies and we get the bound

$$\Pr[E_a] \leq \frac{n_s^2}{|G|}. \tag{11}$$

The bound on E_c is structured as a sequence of games. Let $E_{c,i}$ be the event that the adversary outputs the correct guess in Game i (for β if the execution is fresh, for β'' if not).

We must also keep track of any possible loss of implicit success rate throughout the sequence of games. We let SR_i denote the probability space given by the adversary's implicit success rate in Game i.

The first five games implement several technical changes. The sixth game has responders reject if the adversary has not evaluated *kdf* at the appropriate point, forcing the adversary to commit to a password guess and preventing more than one password guess per responder instance. The seventh game has initiators reject if the adversary evaluates *kdf* at multiple points, preventing more than one password guess per initiator instance unless the adversary solves the StCDH problem. The eight game prevents the adversary from evaluating *kdf* at certain

points, preventing the adversary from learning anything about fresh session keys, unless the adversary solves the StCDH problem. After excluding these cases, we can simulate the remaining interactions with the adversary without knowledge of instances passwords. The result is a password guesser, which gives us a pure opponent.

For a more compact notation, we denote the value $g^{pbkdf(pw)} \in G$ by g_{pw}.

Game 0 The initial game is the experiment interacting with the adversary $(\mathcal{G}, \mathcal{A})$. Then

$$\Pr[E_c] = \Pr[E_{c,0}], \quad \text{and} \quad \mathbf{SuccR}^{\mathsf{pake\text{-}i}}_{\mathrm{SRP}'}(\mathcal{G}, \mathcal{A}) = SR_0. \tag{12}$$

Let $\tau_0 = \tau$.

Game 1 In this game, when the (inverse) block cipher oracle receives a query for which no corresponding record exists, it samples its response from the uniform distribution on G, without rejecting already used group elements.

We may assume that this does not change the time bound on the game, so $\tau_1 = \tau_0$.

This change is only observable if the oracle ever records a collision. Since elements are sampled independently from the uniform distribution, the birthday bound applies and we get

$$|\Pr[E_{c,1}] - \Pr[E_{c,0}]| \leq \frac{n_h^2}{|G|} \quad \text{and} \quad SR_1 \leq SR_0 + \frac{n_h^2}{|G|}. \tag{13}$$

Game 2 In this game, we stop if two instances ever send the same group element, or the *kdf* has been queried at that group element before, or the block cipher has ever returned the group element of a responder instance before the responder instance made its query.

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_2 is essentially equivalent to τ_1 plus at most a small multiple of τ.

Since the group elements are sampled from the uniform distribution (and in the responder's case, have a permutation applied to it) and independently, the usual analysis applies and we get

$$\begin{aligned} |\Pr[E_{c,2}] - \Pr[E_{c,1}]| &\leq \frac{n_s(n_s + n_h) + n_h}{|G|} \text{ and} \\ SR_2 &\leq SR_1 + \frac{n_s(n_s + n_h) + n_h}{|G|}. \end{aligned} \tag{14}$$

Game 3 In this game, we stop if there is ever a collision in *pbkdf*.

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_3 is essentially equivalent to τ_2 plus at most a small multiple of τ.

The birthday bound applies, giving us

$$|\Pr[E_{c,3}] - \Pr[E_{c,2}]| \leq \frac{n_h^2}{|G|} \quad \text{and} \quad SR_3 \leq SR_2 + \frac{n_h^2}{|G|}. \tag{15}$$

Game 4 Consider an block cipher query (g_{pw}, y) where the oracle must sample a response, that is, the block cipher has not seen this query before, nor has the inverse block cipher responded with y to some query with the key g_{pw}. We say that such a query is *colliding* if the same block cipher value is recorded for a distinct block cipher query. (Note that the previous query may have come from a block cipher query or an inverse block cipher query.) In this game, we stop if we ever observe a colliding query.

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_4 is essentially equivalent to τ_3 plus at most a small multiple of τ.

The worst case is if the ideal cipher is evaluated at most once for any key, in which case we consider at most n_h elements sampled independently from the uniform distribution, giving us

$$|\Pr[E_{c,4}] - \Pr[E_{c,3}]| \leq \frac{n_h^2}{|G|} \quad \text{and} \quad SR_4 \leq SR_3 + \frac{n_h^2}{|G|}. \tag{16}$$

Game 5 In this game, we stop if two distinct queries to *kdf* ever return identical results in the second coordinate (the initiator's tag).

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_5 is essentially equivalent to τ_4 plus at most a small multiple of τ.

Since the random oracle samples values from the uniform distribution and independently, the birthday paradox applies and we get

$$|\Pr[E_{c,5}] - \Pr[E_{c,4}]| \leq \frac{n_h^2}{|T|} \quad \text{and} \quad SR_5 \leq SR_4 + \frac{n_h^2}{|T|}. \tag{17}$$

A responder instance receives a tag which now uniquely identifies a particular *kdf* query. By the changes in the previous game, this query must contain the instance password.

Remark 21. While we will need a certain amount of collision resistance in *kdf*, the above bounds are likely an overestimate. The issue is that an adversary can use collisions in *kdf* and a responder instance to make multiple password guesses. Since the collisions can be found without interaction with the experiment, this breaks security. But the adversary must see the responder's group element before finding the collisions, which means that the collisions must match a particular pattern, and the only thing that can vary is the password. It is also likely that the responder will time out, strictly limiting the available resources to find collisions. The above details are easy to describe, but somewhat tricky to model, so we do not.

Game 6 In this game, we shall have certain instances reject. Consider an instance with associated data ad using password pw that sent and received x, y and t_1, and for initiator instances, also t_2, and that no instance with the same associated data and password sent the group element the instance first received. If the adversary did not query *kdf* at (ad, pw, x, y, z_1, z_2), where $(g, x, \pi^{-1}(g_{pw}, y), z_1)$ and $(g, g_{pw}, \pi^{-1}(g_{pw}, y), z_2)$ are DDH tuples, then the instance rejects, regardless of the value of the relevant tag.

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_6 is essentially equivalent to τ_5 plus at most a small multiple of τ.

By an earlier change, instances send unique messages. If a particular instance cannot have a partner, no other instance will evaluate *kdf* at the point this instance will. For any particular instance, since the adversary has not queried *kdf*, it has no information about the correct value of the tag. (Note that we *stop* after certain collisions, but the adversary cannot know if we will stop, so this does not give the adversary any information.) The adversary is therefore correct with probability at most $1/|T|$, giving us

$$|\Pr[E_{c,6}] - \Pr[E_{c,5}]| \leq \frac{n_s}{|T|} \quad \text{and} \quad SR_6 \leq SR_5 + \frac{n_s}{|T|}. \tag{18}$$

This change means that before any instance accepts, there will be a particular *kdf* query containing the instance password. We now need to make sure this query is unique.

Game 7 In this game, we stop if for any initiator instance with associated data ad that sent x and t_1 and received y, the adversary queries *kdf* at two points $(ad, pw, x, y, z_1, \cdot)$ and $(ad, pw', x, y, z_1', \cdot)$, $pw \neq pw'$, such that $(g, x, \pi^{-1}(g_{pw}, y), z_1)$ and $(g, x, \pi^{-1}(g_{pw'}, y), z_1')$ are both DDH tuples.

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_7 is essentially equivalent to τ_6 plus at most a small multiple of τ.

We get that for any honest initiator instance, the adversary made at most one *kdf* query related to the instance. By the changes in the previous games, if the instance accepts there must be such a query.

Let F_i be the event that this happens in Game i, $i = 6, 7$. Since the games proceed identically until F_i happens, we get that $\Pr[F_7] = \Pr[F_6]$, and that

$$|\Pr[E_{c,7}] - \Pr[E_{c,6}]| \leq \Pr[F_7] \quad \text{and} \quad SR_7 \leq SR_6 + \Pr[F_7]. \tag{19}$$

Lemma 18. *There exists a* $(\tau_1', 0, n_s + n_h, n_h)$*-solver* $\mathcal{B}_1$ *for* StCDH*, with* τ_1' *essentially equal to* τ_7*, such that*

$$\boldsymbol{Adv}_G^{\mathsf{StCDH}}(\mathcal{B}_1) = \Pr[F_7].$$

Proof. The adversary $\mathcal{B}_1$ runs a copy of $\mathcal{A}$ and a simulation **Sim** of the ideal cipher and the random oracles *pbkdf* and *kdf* interacting with a copy of the experiment $\mathbf{Exp}_{\text{SRP}'}^{\mathsf{pake\text{-}ror}}$, all as in Game 7, further modified as follows:

- Initiator instances use the second CDH oracle to get x.
- When the block cipher oracle receives a query (g_{pw}, h) for which no record $(g_{pw}, h, y, \cdot)$ exists, it samples $y \xleftarrow{r} G$, records $(g_{pw}, h, y, \bot)$ and responds with y.
- When the inverse block cipher oracle receives a query (g_{pw}, y) for which no record (g_{pw}, h, y) exists, the block cipher oracle queries the third CDH oracle to get h, records $(g_{pw}, h, y, \top)$ and responds with h.
- When an initiator oracle with associated data ad and password pw that sent x and received y would query the *kdf* oracle, if a record $(ad, pw, x, y, \cdot, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}, k, t_1, t_2, \top)$ exists, we use (k, t_1, t_2). Otherwise, we sample $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, record $(ad, pw, x, y, -, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}, k, t_1, t_2, \top)$ and use (k, t_1, t_2).
- When the adversary makes a new *kdf* query (ad, pw, x, y, z_1, z_2), with $z_2 = \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}$), and some initiator instance has transcript $(ad, \cdot, x, \dots)$, **Sim** uses its Strong Diffie-Hellman oracle to determine if $(g, x, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}, z_1)$ is a DDH tuple. If it is not a DDH tuple, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \bot)$ and responds with (k, t_1, t_2). If it is a DDH tuple and $(ad, pw, x, y, -, z_2, k, t_1, t_2, \top)$ is recorded, **Sim** updates the record to include z_1 and responds with (k, t_1, t_2). Otherwise, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \top)$ and responds with (k, t_1, t_2).
- Suppose **Sim** ever makes two records $(ad, pw, x, y, z_1, \cdot, \cdot, \cdot, \cdot, \top)$ and $(ad, pw', x, y, z_1', \cdot, \cdot, \cdot, \cdot, \top)$. Then for some h either $(g_{pw}, h, y, \top)$ or $(g_{pw'}, h, y, \top)$ are recorded, so $\mathcal{B}_1$ outputs (g, x, h, z_1).

The main difficulty with the simulation is that the underlying game knows the discrete logarithms of the messages sent by initiator instances, while the simulation does not. In particular, this means that the simulation does not know z_1 for its own instances.

This difficulty is resolved by using lazy evaluation of the hash oracle input combined with ensuring consistency in query responses by using the Strong Diffie-Hellman oracle provided by the StCDH problem. Lazy evaluation is done in the fourth and fifth bullet points, where we choose consistent values for the session key without knowing the entire hash input. We do not know the entire hash input because we have embedded CDH challenges in the protocol messages.

The only interesting case is the initiator instances that have sent a message that we do not know the discrete logarithm of. We use a blank value for the Diffie-Hellman secret, and then we need to recognise when the adversary queries the oracle at the point the instance would have queried it. Since our message came from the second CDH oracle, we can use the Strong Diffie-Hellman oracle to recognize the correct query.

(Note that we do not need any oracles for responder oracles, since we know both the password and the discrete logarithm of the message.)

It therefore follows that until $\mathcal{B}_1$ stops, it perfectly simulates Game 7 (and Game 6). It only stops if the event F_7 happens. When this event happens, $\mathcal{B}_1$ stops and its output is a solution to a CDH challenge.

Since $\mathcal{B}_1$ makes queries to the Strong Diffie-Hellman oracle only when queries to *kdf* are made, this bounds the number of Strong Diffie-Hellman queries by n_h. Since $\mathcal{B}_1$ queries its sampling oracles either for initiator instances or in response to block cipher oracle queries, we can bound the number of sampling queries by $n_s + n_h$.

Finally, the simulation requires some extra accounting, but the runtime of $\mathcal{B}_1$ and its experiment is essentially the same as the runtime of Game 7. □

Game 8 In this game, we stop if the adversary makes a *kdf* query $(ad, pw, x, y, z_1, \cdot)$ such that some responder instance has transcript $(ad, \cdot, x, y, \dots)$, some initiator instance has transcript $(ad, \cdot, x, y, \dots)$, and $(g, x, \pi^{-1}(g_{pw}, y), z_1)$ is a DDH tuple.

This game requires some extra accounting for each oracle query. Since the number of hash queries is bounded by τ, the runtime bound τ_8 is essentially equivalent to τ_7 plus at most a small multiple of τ.

Let F'_i be the event that this happens in Game i, $i = 7, 8$. Since the games proceed identically until F'_i happens, we get that $\Pr[F'_8] = \Pr[F'_7]$, and that

$$|\Pr[E_{c,8}] - \Pr[E_{c,7}]| \leq \Pr[F'_8] \quad \text{and} \quad SR_8 \leq SR_7 + \Pr[F'_8]. \tag{20}$$

Lemma 19. *There exists a (τ'_2, n_h, n_s, n_h)-solver $\mathcal{B}_2$ for* StCDH*, with τ'_2 essentially equal to τ_8, such that*

$$\boldsymbol{Adv}_G^{\mathsf{StCDH}}(\mathcal{B}_2) = \Pr[F'_8].$$

Proof. The adversary $\mathcal{B}_2$ runs a copy of $\mathcal{A}$ and a simulation **Sim** of the ideal cipher oracle and the random oracles *pbkdf* and *kdf* interacting with a copy of the experiment $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\mathrm{SRP}'}$, all as in Game 8, further modified as follows:

- Initiator instances use the second CDH oracle to get x. Responder instances receiving x sent by an initiator instance use the third CDH oracle to get h and use the block cipher with key g_{pw} to get y.
- When an initiator or responder instance with associated data ad, password pw and first two messages x and y would query the *kdf* oracle, if a record $(ad, pw, x, y, \cdot, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}, k, t_1, t_2, \top)$ exists, we use (k, t_1, t_2). Otherwise, we sample $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, record $(ad, pw, x, y, -, \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}, k, t_1, t_2, \top)$ and use (k, t_1, t_2).
- When the adversary makes a new *kdf* query (ad, pw, x, y, z_1, z_2), with $z_2 = \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}$, and some responder instance has transcript $(ad, \cdot, x, y, \dots)$ and some initiator instance has transcript $(ad, \cdot, x, y, \dots)$, **Sim** uses its Strong Diffie-Hellman oracle to check if $(g, x, \pi^{-1}(g_{pw}, y), z_1)$ is a DDH tuple. If so, $\mathcal{B}_2$ outputs z_1. Otherwise, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \bot)$ and returns (k, t_1, t_2).

- When the adversary makes a new *kdf* query (ad, pw, x, y, z_1, z_2), with $z_2 = \pi^{-1}(g_{pw}, y)^{pbkdf(pw)}$, and some initiator instance sent x or some responder instance with password pw sent y, but not both, *simulator* uses its Strong Diffie-Hellman oracle to check if $(g, x, \pi^{-1}(g_{pw}, y), z_1)$ is a DDH tuple. If it is not, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \bot)$ and returns (k, t_1, t_2). If it is and a record $(ad, pw, x, y, -, z_2, k, t_1, t_2, \top)$ exists, **Sim** updates the record to include z_1 and responds with (k, t_1, t_2). Otherwise, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \top)$ and returns (k, t_1, t_2).

The main difficulty with the simulation is that the underlying game knows the discrete logarithms of the messages sent by instances, while the simulation does not. In particular, this means that the simulation does not know z_1 for its own instances.

This difficulty is resolved by using lazy evaluation of the hash oracle input combined with ensuring consistency in query responses by using the Strong Diffie-hellman oracle provided by the StCDH problem. Lazy evaluation is done in the third and fourth bullet points, where we choose consistent values for the session key without knowing the entire hash input. We do not know the entire hash input because we have embedded CDH challenges in the protocol messages.

When instances evaluate *kdf*, we use a blank value for the Diffie-Hellman secret, and then we need to recognise when the adversary queries the oracle at the point the instance would have queried it. Since our messages come from the second and third StCDH oracles, we can use the Strong Diffie-Hellman oracle to recognize the correct query.

It therefore follows that until $\mathcal{B}_2$ stops, it perfectly simulates Game 8 (and Game 7). It only stops if the event F_8' happens. When this event happens, $\mathcal{B}_2$ stops and its output is a solution to a CDH challenge.

Since $\mathcal{B}_2$ makes queries to the Strong Diffie-Hellman oracle only when queries to *kdf* are made, this bounds the number of Strong Diffie-Hellman queries by n_h. Since $\mathcal{B}_2$ queries its sampling oracles either for initiator instances or in response to block cipher oracle queries, we can bound the number of sampling queries by $n_s + n_h$.

Finally, the simulation requires some extra accounting, but the runtime of $\mathcal{B}_2$ and its experiment is essentially the same as the runtime of Game 8. □

Conclusion We now observe that in the final game, the adversary has no information about the session key, so

$$\Pr[E_{c,8}] = 1/2. \tag{21}$$

Proposition 17 then follows from equations (11)–(21) and Lemmas 18, 19 and 20.

Lemma 20. *There exists a (τ_3', n_u, n_s)-password guesser $(\mathcal{G}, \mathcal{B}_3)$, with τ_3' essentially equal to τ_8, such that*

$$\boldsymbol{SuccR}^{\mathsf{pw}}(\mathcal{G}, \mathcal{B}_3) = SR_8.$$

Proof. The adversary $\mathcal{B}_3$ runs a copy of $\mathcal{A}$ and a simulation **Sim** of the ideal cipher oracle and the random oracles *pbkdf* and *kdf* interacting with a copy of the experiment $\mathbf{Exp}^{\mathsf{pake\text{-}ror}}_{\mathrm{SRP}'}$, all as in Game 8, further modified as follows:

- When $\mathcal{B}_3$ is notified about the jth password, **Sim** records $(j, \bot)$.
- When $\mathcal{B}_3$ makes a guess query (j, pw) and gets a response $\top$, it replaces the record $(j, \bot)$ by (j, pw).
- When the block cipher receives a query (g_{pw}, h) for which no record $(g_{pw}, h, \cdot, \cdot)$ exists, the block cipher oracle $r \xleftarrow{r} \{0, 1, \ldots, p-1\}$ and computes $y \xleftarrow{r} g^r$. If some responder instance ever sent y, **Sim** stops. Otherwise, **Sim** records $(g_{pw}, h, y, 0)$ and responds with y.
- When the inverse block cipher oracle receives a query (g_{pw}, y) for which no record $(g_{pw}, h, y, \cdot)$ exists, the block cipher oracle samples $r \xleftarrow{r} \{0, 1, \ldots, p-1\}$, computes $h \xleftarrow{r} g^r$, records $(g_{pw}, h, y, 1)$ and responds with h.
- Initiator instances sample their messages as usual. Responder instances sample $y \xleftarrow{r} G$. If the ideal cipher oracle already has a record $(\cdot, \cdot, y, \cdot)$, **Sim** stops.
- Suppose an instance with associated data ad, using the jth password and with messages x and y wants to compute its session key.
  - If (j, pw) is recorded, **Sim** computes the correct Diffie-Hellman secrets z_1 and z_2. If $(ad, pw, x, y, \cdot, \cdot, k, t_1, t_2, \top)$ is recorded, **Sim** uses (k, t_1, t_2). Otherwise, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \top)$ and uses (k, t_1, t_2).
  - If $(j, \bot)$ and $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \top)$ are recorded, $\mathcal{B}_3$ makes a guess query (j, pw). If the result is $\top$, it uses (k, t_1, t_2). If the result is $\bot$, it samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, -, x, y, -, -, k, t_1, t_2, j)$ and uses (k, t_1, t_2).
  - If $(j, \bot)$ and $(ad, -, x, y, -, -, k, t_1, t_2, j)$ are recorded, then **Sim** uses (k, t_1, t_2). (Note that unless $(\mathcal{G}, \mathcal{A})$ is context-respecting, this could cause inconsistent results.)
  - If $(j, \bot)$ is recorded, but no record $(ad, \cdot, x, y, \cdot, \cdot, \cdot, \cdot, \top)$ or $(ad, \cdot, x, y, \cdot, \cdot, \cdot, \cdot, j)$ exists, then **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, -, x, y, -, -, k, t_1, t_2, j)$ and uses (k, t_1, t_2).
- When $\mathcal{A}$ makes a new query (ad, pw, x, y, z_1, z_2) to the random oracle *kdf* such that either x or y was sent by an honest instance, then **Sim** checks if both $(g, x, \pi^{-1}(g_{pw}, y), z_1)$ and $(g, g_{pw}, \pi^{-1}(g_{pw}, y), z_2)$ are DDH tuples.
  - If either is not a DDH tuple, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \bot)$ and returns (k, t_1, t_2).
  - If both are DDH tuples, some responder instance has transcript $(ad, \cdot, x, y, \ldots)$ and some initiator instance has transcript $(ad, \cdot, x, y, \ldots)$, **Sim** stops.

– If both are DDH tuples, $(ad, pw', x, y, z'_1, z'_2, \cdot, \cdot, \cdot, \top)$ is recorded with $pw' \neq pw$, and some initiator instance has transcript $(ad, \cdot, x, y, \dots)$, **Sim** stops.

– If both are DDH tuples and $(ad, -, x, y, -, -, k, t_1, t_2, j)$ is recorded (that is, there is a unique instance with transcript $(ad, \cdot, x, y)$ that uses the jth password), **Sim** makes a guess query of (j, pw). If the result is $\top$, the record is updated to $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \top)$. Otherwise, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \bot)$ and returns (k, t_1, t_2).

– Otherwise, **Sim** samples $(k, t_1, t_2) \xleftarrow{r} K \times T^2$, records $(ad, pw, x, y, z_1, z_2, k, t_1, t_2, \bot)$ and returns (k, t_1, t_2).

We must show that $\mathbf{Exp}^{\mathsf{pw}}$ together with **Sim** perfectly simulate the experiment and the oracles in Game 8. The only difficulty is that some simulated instances do not know their passwords. Their initial messages are independent of the password, so these can be simulated perfectly, though some care is needed with respect to responder messages, both to avoid collisions in the block cipher and ensure that we know the discrete logarithm of the requisite group elements. (We rely on exceptions instituted in earlier games to reject these cases.) If an instance password becomes known, we can recreate the instance DDH tuples correctly.

Until the password is known, we cannot know the input to the session key derivation, which we deal with using lazy evaluation. The adversary may evaluate the hash at DDH tuples (which correspond to password guesses), but this is forbidden for instances with partners and allowed at most once for initiator instances. For responder instances, the adversary will have to point to a particular query which gives us a guess.

Sometimes, the adversary evaluates the hash at DDH tuples before the instance derives its session key, which **Sim** notices and turns into a password guess. Other times, the adversary evaluates the hash at DDH tuples after the instance derives its session key. Again, **Sim** notices and turns it into a password guess. If the guess is correct, any records of lazy evaluation are updated to keep the simulated hash oracle consistent. The end result is that we make at one password guess per instance.

It follows that the simulation is perfect, that the password guesser $(\mathcal{G}, \mathcal{B}_3)$ makes at most n_u password queries, and at most one password guess per instance.

Finally, any hash query made by $\mathcal{A}$ that allows SR_8 to increase corresponds exactly to a hash query that makes $\mathcal{B}_3$ guess a password, so $\mathbf{SuccR}^{\mathsf{pw}}(\mathcal{G}, \mathcal{B}_3) = SR_8$ and the lemma follows. □

4 Applications

In order to see the utility of our security definition, we want to investigate two applications of password-authenticated key exchange.

In the interests of simplicity, we do not go for a fully general definitions for either application, nor will we use fully compositional constructions. We choose these particular protocols because they are simple to explain and easy to analyse, not because they have particularily strong security properties or are otherwise desirable. Other protocols achieve stronger or different security notions. We make several remarks along these lines.

Security definitions and proof sketches are in Appendix A.

4.1 Secure Channels

A *secure channel* protocol is a protocol that provides two-party conversations. While these may be actual human-to-human conversations, they are more likely to be computer-to-computer conversations. TLS [24] is one example of such a protocol. We shall only consider the simplest class of such protocols, where each plaintext message sent corresponds to a single network message sent. Any such protocol has three tasks: Establishing a state for a conversation, encrypting messages and decrypting messages.

Our modelling of secure channels captures establishing a conversation, done by an initiator and a responder, and then encryption and decryption of messages. A secure channel is asynchroneous, as usual. We make no attempt to model the actual transmission of messages. Instead, the network is the adversary. As earlier, we base our definition on a secure channel definition from a recent textbook [15], but avoid most of the somewhat baroque complexity in that presentation.

Definition 21. A *password-authenticated secure channel* protocol $\text{PBMSG} = (P, AD, W, \mathcal{I}, \mathcal{R}, \mathcal{E}, \mathcal{D})$ consists of sets of plaintexts P, associated data AD and passwords W, and four algorithms:

- The interactive initiator and responder algorithms $\mathcal{I}$ and $\mathcal{R}$ take as input a password $pw \in W$ and associated data $ad \in AD$ and output either a state st or $\perp$. They alternate between sending and receiving messages, with $\mathcal{I}$ sending the first message and $\mathcal{R}$ receiving the first message.
- The encryption algorithm $\mathcal{E}$ takes as input a state st and a plaintext m and outputs a new state st' and a ciphertext c.
- The decryption algorithm $\mathcal{D}$ takes as input a state st and a ciphertext c and outputs either $\perp$, or a new state st' and a message m.

An *instance* of the secure channel protocol first runs either the initiator or the responder algorithms to get a state, and then runs the encryption and decryption algorithms multiple times, each time updating the state. The *role* of an instance is 0 if it ran the initiator algorithm, 1 if it ran the responder algorithm. An instance is *accepting* if the initiator or responder algorithm output a state.

Our example protocol for password-authenticated secure channels is the classical cryptographic design, composing password-authenticated key exchange

with symmetric key encryption. We use symmetric encryption with associated data, which allow us to keep track of which direction messages are sent and in which order. This construction is very simple, e.g. encryption and decryption are independent processes and there is no attempt to communicate how many messages have been decrypted.

Example 22. Let PAKE $= (K, W, AD, \mathcal{I}_0, \mathcal{R}_0)$ be a password-authenticated key exchange protocol and let SYM be a symmetric cryptosystem with plaintext set P, key set K, associated data set $\{0,1\} \times \mathbb{Z}$, encryption algorithm $\mathcal{E}_0$ and decryption algorithm $\mathcal{D}_0$. The password-authenticated secure channel protocol PBMSG $= (P, AD, W, \mathcal{I}, \mathcal{R}, \mathcal{E}, \mathcal{D})$ has the following algorithms:

- The initiator and responder algorithms $\mathcal{I}$ and $\mathcal{R}$ take as input a password $pw \in W$ and $ad \in AD$ and run $\mathcal{I}_0$ and $\mathcal{R}_0$, respectively. If the output is k, the initiator and responder algorithm output the state $(\rho, k, 0, 0)$, where $\rho = 0$ for $\mathcal{I}$ and $\rho = 1$ for $\mathcal{R}$. If the output is $\perp$, the initiator and responder algorithm output $\perp$.
- The encryption algorithm $\mathcal{E}$ takes as input a state (ρ, k, j_0, j_1) and a plaintext m. It computes $c \leftarrow \mathcal{E}_0(k, (\rho, j_\rho), m)$ and outputs the new state $(\rho, k, j_0 + (1-\rho), j_1 + \rho)$ and the ciphertext c.
- The decryption algorithm $\mathcal{D}$ takes as input a state (ρ, k, j_0, j_1) and a ciphertext c. It computes $m \leftarrow \mathcal{D}_0(k, (1-\rho, j_{1-\rho}), c)$. If the decryption fails, $\mathcal{D}$ outputs $\perp$. Otherwise, the algorithm outputs the new state $(\rho, k, j_0 + \rho, j_1 + (1-\rho))$ and the message m.

It is worth emphasising the plain meaning of the security definition. The intention is that we give traditional cryptographic proofs to bound the advantage, while we bound the success rate using a password guesser. This provides the following guarantees (cf. C1, C2 and C3 from Section 1):

- For any conversation started, the other party knows the password and agrees on the associated data.
- An adversary's ability to guess passwords is bounded by the number of sessions.
- When two honest parties successfully start a conversation with each other, then, *even if the adversary knows the password*, they agree on the content and order of messages, up to the last sent messages not arriving, and the conversation is private, up to the length of the messages.

Proposition 23. *Consider the scheme* PBMSG *from Example 22 based on a password-authenticated key exchange scheme* PAKE *and a symmetric cryptosystem* SYM*. Let $(\mathcal{G}, \mathcal{A})$ be a context-respecting $(\tau, n_c, n_u, n_s, n_e)$-adversary against* PBMSG*. Then there exists a context-respecting $(\tau_1', 0, n_u, n_s)$-adversary $(\mathcal{G}, \mathcal{B}_1)$ against* PAKE*, a context-respecting $(\tau_2', 0, n_u, n_s)$-adversary $(\mathcal{G}, \mathcal{B}_2)$ against* PAKE*,*

context-respecting (τ_3', n_s, n_u, n_s)*-adversaries* $(\mathcal{G}, \mathcal{B}_3)$ *and* $(\mathcal{G}, \mathcal{B}_3')$ *against* PAKE*, a multi-key* $(\tau_4', n_u, n_e+n_c, n_e+n_c)$*-adversary* $\mathcal{B}_4$ *against ciphertext integrity for* SYM *and a multi-key* $(\tau_5', n_u, n_c, n_e+n_c, n_e)$*-adversary* $\mathcal{B}_5$ *against real-or-random for* SYM*, with* τ_1'*,* τ_2'*,* τ_3'*,* τ_4' *and* τ_5' *essentially equal to* τ*, such that*

$$\begin{aligned}\boldsymbol{Adv}^{\mathsf{ror}}_{PBMSG}(\mathcal{G}, \mathcal{A}) \leq\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_2) + \\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_3) + 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_3') + \\ & 2\boldsymbol{Adv}^{\mathsf{int\text{-}ctxt}}_{SYM}(\mathcal{B}_4) + \boldsymbol{Adv}^{\mathsf{ind\text{-}cca}}_{SYM}(\mathcal{B}_5) \text{ and,} \\ \boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}(\mathcal{G}, \mathcal{B}_1) \leq\ & \boldsymbol{SuccR}^{\mathsf{pbmsg}}_{PBMSG}(\mathcal{G}, \mathcal{A}).\end{aligned}$$

Remark 22. There are a number of ways to build password-authenticated secure channel protocols with stronger security properties. If we use *evolving session keys*, we can allow state reveals after a challenge query. We could also use two-round unauthenticated key exchange to allow challenge queries after a state reveal, once the instance has recovered. (We do not *want* to use unauthenticated key exchange, but since authentication would have to be password-authenticated, the only practicable option seems to be unauthenticated key exchange.)

Remark 23. We emphasise that the protocols we study in this section are chosen to be simple examples of how our model works, and not suggestions for how to do password-authenticated cryptography in the real world.

Typically, a password-authenticated secure channel is used between a client and a server. If the adversary learns the users's password, pure cryptography cannot help the server (though things like confirmation messages sent through independent channels may to some extent work). But the adversary should not be able to impersonate the server.

The obvious approach is to give servers signing keys. The client and server then run password-authenticated key exchange, and the server signs the associated data and the protocol messages.

In the model, we would need to add key generation queries for server key pairs as well as long term key reveals for both secret keys and passwords. For authentication, we would require a partner for initiator instances, but not for server instances. For freshness, we would still require a partner for server instances, but we need not require partner instances for client instances.

For the security proof, we add another game where we reject if a client does not have a partner. If this changes the game behaviour, it leads to an attack on the signature scheme.

An alternative approach is to use a mixed-authentication scheme, as discussed in Remark 19. We leave this for future work.

Remark 24. An added complication is that we want surveillance-resistant cryptography, in the sense that a passive adversary cannot deduce the identities of communicating partners. Our current formulation does not allow for this, because agreeing on which password to use (and the related associated data) happens out of band, which for common use cases means that the information must be sent in the clear.

The standard way to approach this problem is to first establish an unauthenticated secure channel. Identifying information is then exchanged, after which the password authenticated key exchange protocol is run, possibly augmented as in the previous remark to provide one-sided authentication for the server that does not depend on the password. The main trick is that the password-authenticated key exchange protocol needs to be bound to the unauthenticated secure channel, which is done easiest through associated data. It is also natural to mix the established session keys, but as Hesse *et al.* [17] show, this is not necessary.

For the security proof, we would insert a game where we replace the session key established by the unauthenticated key exchange protocol by random keys for instances with partners.

It is an interesting question if the mixed-authentication version of SRP' from Remark 19 can be modified to support surveillance-resistant cryptography.

4.2 Device Authorisation

Our second application captures the popular paradigm of using a password to *authorise* devices to access some resource, which the devices then access at will. It is common practice not to store a user's password on their devices in case a device is lost or otherwise compromised. There are many, many ways to do this, and we shall describe a very simple method.

Our modelling consists of an authorisation process follows by connecting to the service. Authorisation is an interactive process between an initiator and a responder and results in each party getting a token that will later be used to connect to the service. Connecting to the service is modelled as having a new conversation with the service. In order to simplify the presentation, we model starting a new conversation as a non-interactive process. This requires a fairly strong assumption to achieve meaningful security.

Definition 24. A *password-authenticated device authorisation* scheme PBAUTH $= (P, AD, W, \mathcal{I}, \mathcal{R}, \mathcal{H}, \mathcal{E}, \mathcal{D})$ consists of sets of plaintexts P, associated data AD and passwords W, and algorithms:

- The initiator and responder interactive algorithms $\mathcal{I}$ and $\mathcal{R}$ take as input a password $pw \in W$ and $ad \in AD$ and output either a *token* tok or $\perp$.
- The handshake algorithm $\mathcal{H}$ takes as input a token tok and associated data $ad \in AD$ and output a state st.
- The encrypt algorithm $\mathcal{E}$ takes as input a state st and a message m and outputs a ciphertext c and a state st'.
- The decrypt algorithm $\mathcal{D}$ takes as input a state st and a ciphertext c and outputs either $\perp$, or a message m and a state st'.

An instance of the authorisation protocol runs either the initiator or the responder algorithm to get a token. The *role* of an instance running the initiator

algorithm is 0, and 1 for an instance running the responder algorithm. The token output by an instance has the same role as the instance.

We say that an adversary is *nonce-respecting* if it does at most one execute query for each token-associated data pair.

Our construction uses password-authenticated key exchange to get a session key which becomes the token. Session keys for new connections are derived from the token using a key derivation function. Each connection uses a symmetric cryptosystem with associated data just as in Example 22.

Example 25. Let PAKE $= (K, W, AD, \mathcal{I}_0, \mathcal{R}_0)$ be a password-authenticated key exchange protocol and let SYM be a symmetric cryptosystem with key set K_0, plaintext set P, associated data set $AD \times \{0,1\} \times \mathbb{Z}$ and encryption and decryption algorithms $\mathcal{E}_0$ and $\mathcal{D}_0$, and let $kdf : K \times AD \to K_0$ be a key derivation function. The password-authenticated device authorisation scheme PBAUTH $= (P, AD, W, \mathcal{I}, \mathcal{R}, \mathcal{H}, \mathcal{E}, \mathcal{D})$ has the following algorithms:

- The initiator and responder algorithms $\mathcal{I}$ and $\mathcal{R}$ take as input a password $pw \in W$ and $ad \in AD$ and run $\mathcal{I}_0$ and $\mathcal{R}_0$, respectively. If the output is k, the initiator and responder algorithm output the state (ρ, k), where $\rho = 0$ for $\mathcal{I}$ and $\rho = 1$ for $\mathcal{R}$. If the output is $\perp$, the initiator and responder algorithm output $\perp$.
- The handshake algorithm $\mathcal{H}$ takes as input a token (ρ, k) and associated data $ad \in AD$ and outputs a state $(\rho, kdf(k, ad), 0, 0)$.
- The encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$ are exactly as in Example 22.

Remark 25. This mechanism is also used for *session resumption* in secure channel systems, with varying degrees of elaboration.

It is worth emphasising the plain meaning of the security definition. The intention is that we give traditional cryptographic proofs to bound the advantage, while we bound the success rate using a password guesser. This provides the following guarantees (cf. C1, C2 and C3 from Section 1):

- For any connection started without either partner token being revealed, the other party knows the password and agrees on the associated data.
- An adversary's ability to guess passwords is bounded by the number of authorisation sessions.
- When two honest parties successfully start a connection with each other, then, *even if the adversary knows the password*, they agree on the content and order of messages, up to the last sent messages not arriving, and the conversation is private, up to the length of the messages.

Proposition 26. *Consider the scheme* PBAUTH *from Example 25 based on a password-authenticated key exchange scheme* PAKE*, a symmetric cryptosystem* SYM *and a key derivation function kdf. Let* $(\mathcal{G}, \mathcal{A})$ *be a context-respecting* $(\tau, n_c, n_u, n_s, n_e)$*-adversary against* PBAUTH*, with kdf modelled as a random oracle. Then there exists a context-respecting* $(\tau'_1, 0, n_u, n_s)$*-adversary* $(\mathcal{G}, \mathcal{B}_1)$ *against* PAKE*, a context-respecting* $(\tau'_2, 0, n_u, n_s)$*-adversary* $(\mathcal{G}, \mathcal{B}_2)$ *against* PAKE*, context-respecting* (τ'_3, n_s, n_u, n_s)*-adversaries* $(\mathcal{G}, \mathcal{B}_3)$ *and* $(\mathcal{G}, \mathcal{B}'_3)$ *against* PAKE*, a context-respecting multi-key* $(\tau'_4, n_u, n_e + n_c, n_e + n_c)$*-adversary* $\mathcal{B}_4$ *against ciphertext integrity for* SYM *and a context-respecting multi-key* $(\tau'_5, n_u, n_c, n_e + n_c, n_e)$*-adversary* $\mathcal{B}_5$ *against real-or-random for* SYM*, with* τ'_1*,* τ'_2*,* τ'_3*,* τ'_4 *and* τ'_5 *essentially equal to* τ*, such that*

$$\begin{aligned}\boldsymbol{Adv}^{\mathsf{ror}}_{\mathit{PBAUTH}}(\mathcal{G}, \mathcal{A}) \leq\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathit{PAKE}}(\mathcal{G}, \mathcal{B}_2) + \\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathit{PAKE}}(\mathcal{G}, \mathcal{B}_3) + 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathit{PAKE}}(\mathcal{G}, \mathcal{B}'_3) + \\ & 2\boldsymbol{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathit{SYM}}(\mathcal{B}_4) + \boldsymbol{Adv}^{\mathsf{ind\text{-}cca}}_{\mathit{SYM}}(\mathcal{B}_5) \text{ and,} \\ \boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}(\mathcal{G}, \mathcal{B}_1) \leq\ & \boldsymbol{SuccR}^{\mathsf{pbauth}}_{\mathit{PBAUTH}}(\mathcal{G}, \mathcal{A}).\end{aligned}$$

Remark 26. There are a number of ways to build authorisation protocols with stronger security properties. If the handshake algorithm is replaced by a pair of interactive algorithms, we can get confidentiality even after tokens are revealed, for instance by using an ordinary key exchange algorithm authenticated using symmetric cryptography. We can even put public and private key pairs into the tokens (the initiator and responder need not output identical tokens) so that compromising the initiator token does not allow an adversary to impersonate the responder.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments.

References

[1] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. Cryptology ePrint Archive, Report 2020/320, 2020. `https://eprint.iacr.org/2020/320`.

[2] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020.

[3] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021,*

Part IV, volume 13093 of *LNCS*, pages 711–741. Springer, Heidelberg, December 2021.

[4] Michel Abdalla, Björn Haase, and Julia Hesse. CPace, a balanced composable PAKE. Internet-Draft draft-irtf-cfrg-cpace-07, Internet Engineering Task Force, January 2023. Work in Progress.

[5] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

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

[7] C. Boyd, A. Mathuria, and D. Stebila. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer Berlin Heidelberg, 2019.

[8] Daniel De Almeida Braga, Pierre-Alain Fouque, and Mohamed Sabt. PARASITE: PAssword recovery attack against srp implementations in ThE wild. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2497–2512. ACM Press, November 2021.

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `https://eprint.iacr.org/2000/067`.

[10] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

[11] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

[12] Katriel Cohn-Gordon, Cas Cremers, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. Highly efficient key exchange protocols with optimal tightness. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 767–797. Springer, Heidelberg, August 2019.

[13] Kai Gellert, Kristian Gjøsteen, Håkon Jacobsen, and Tibor Jager. On optimal tightness for key exchange with full forward secrecy via key confirmation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023,*

Part IV, volume 14084 of *LNCS*, pages 297–329. Springer, Heidelberg, August 2023.

[14] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 142–159. Springer, Heidelberg, August 2006.

[15] Kristian Gjøsteen. *Practical Mathematical Cryptography*. CRC Press, 2022.

[16] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/7384`.

[17] Julia Hesse, Stanislaw Jarecki, Hugo Krawczyk, and Christopher Wood. Password-authenticated TLS via OPAQUE and post-handshake authentication. Cryptology ePrint Archive, Report 2023/220, 2023. `https://eprint.iacr.org/2023/220`.

[18] David P. Jablon. Strong password-only authenticated key exchange. *Comput. Commun. Rev.*, 26(5):5–26, 1996.

[19] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018.

[20] Eike Kiltz, Jiaxin Pan, Doreen Riepel, and Magnus Ringerud. Multi-user CDH problems and the concrete security of NAXOS and HMQV. In Mike Rosulek, editor, *Topics in Cryptology - CT-RSA 2023 - Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24-27, 2023, Proceedings*, volume 13871 of *Lecture Notes in Computer Science*, pages 645–671. Springer, 2023.

[21] Jack Levine and J. V. Brawley. Some cryptographic applications of permutation polynomials. *Cryptologia*, 1(1):76–92, 1977.

[22] Xiangyu Liu, Shengli Liu, Shuai Han, and Dawu Gu. EKE meets tight security in the universally composable framework. Cryptology ePrint Archive, Report 2023/170, 2023. `https://eprint.iacr.org/2023/170`.

[23] Michael Luby and Charles Rackoff. How to construct pseudo-random permutations from pseudo-random functions (abstract). In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, page 447. Springer, Heidelberg, August 1986.

[24] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[25] Alan T. Sherman, Erin Lanus, Moses Liskov, Edward Zieglar, Richard Chang, Enis Golaszewski, Ryan Wnuk-Fink, Cyrus J. Bonyadi, Mario Yaksetig, and Ian Blumenfeld. *Formal Methods Analysis of the Secure Remote Password Protocol*, pages 103–126. Springer International Publishing, Cham, 2020.

[26] Victor Shoup. Security analysis of itSPAKE2+. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 31–60. Springer, Heidelberg, November 2020.

[27] Victor Shoup. Security analysis of SPAKE2+. Cryptology ePrint Archive, Report 2020/313, 2020. `https://eprint.iacr.org/2020/313`.

[28] Marjan Skrobot and Jean Lancrenon. 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*, pages 443–457. IEEE, 2018.

[29] Thomas D. Wu. The secure remote password protocol. In *NDSS'98*. The Internet Society, March 1998.

A Applications: Details and Sketches

In these sections, we sometimes denote tuples $(s_1, s_2, \ldots, s_n)$ by a bold-face $\mathbf{s}$.

A.1 Secure Channels: Details and Sketches

Security for a secure channel protocol, as defined, is fairly straight-forward. We want confidentiality, integrity and authentication. We model confidentiality as real-or-random security, in the sense that the adversary cannot tell if a ciphertext contains a message of their choice or a randomly chosen message of the same length. Integrity is ciphertext integrity, which essentially states that an instance will only accept the ciphertexts sent by its partner, in the same order they were sent. Authentication says essentially that whenever we establish a conversation, the password is known to the partner. We could have included authentication into the integrity notion, but it seems appropriate to discuss the two separately.

Definition 27. Consider an instance of role ρ where the initiator or responder algorithm using password pw sent and received a sequence of network messages $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, it encrypted a sequence of messages $\mathbf{m}_\rho = (m_{\rho,1}, \ldots, m_{\rho,n_\rho})$ to ciphertexts $\mathbf{c}_\rho = (c_{\rho,1}, \ldots, c_{\rho,n_\rho})$ and decrypted a sequence of ciphertexts $\mathbf{c}_{1-\rho} = (c_{1-\rho,1}, \ldots, c_{1-\rho,n_{1-\rho}})$ to messages $\mathbf{m}_{1-\rho} = (m_{1-\rho,1}, \ldots, m_{1-\rho,n_{1-\rho}})$. The *message transcript* of an instance is $(\rho, ad, pw, \mathbf{m}_0, \mathbf{m}_1)$. The *network transcript* of an instance is $(\rho, ad, pw, \mathbf{s}, \mathbf{c}_0, \mathbf{c}_1)$. (Note that the encryptions and decryptions may be interleaved arbitrarily, but the transcripts do not record this information.) The network transcript is *accepting* if the instance was accepting.

The real-or-random experiment proceeds as follows:

1. Sample $\beta, \beta'' \xleftarrow{r} \{0,1\}$.
2. On the jth *password* query pw, do:
   (a) Record (j, pw) and send j.
3. On the ith *execute* query (ρ, j, ad), do:
   (a) If (j, pw) is not recorded, send $\perp$ and stop.
   (b) If $\rho = 0$, start the ith instance as $\mathcal{I}(ad, pw)$. When the instance sends s, send (i, s). Otherwise, start the ith instance as $\mathcal{R}(ad, pw)$ and send i.
4. On the *send* query (i, s), do
   (a) If (i, st) is recorded, send $\perp$.
   (b) Otherwise, send s to the ith instance. If the ith instance outputs $\perp$, record $(i, \perp)$ and send $(i, \perp)$. If the ith instance outputs st, record $(i, st, 0)$. If the instance sent a message s' and did not output a state, send (i, s'). If the instance sent a message s' and output a state, send $(i, \top, s')$. If the instance output a state and did not send a message, send $(i, \top)$.
5. On the *chosen plaintext* / *chosen ciphertext* / *challenge* query (cp, i, m) / (cc, i, c) / (ror, i, m_0), do:
   (a) If (i, st, l) is not recorded, send $\perp$ and stop.
   (b^{cp}) Compute $(st', c) \leftarrow \mathcal{E}(st, m)$ and send c.
   (b^{cc}) Compute $(st', m) \leftarrow \mathcal{D}(st, c)$, record (msg, i, l, m) and send $\top$. If the decryption failed, update the record to $(i, \perp)$, send $\perp$ and stop.
   ($\mathrm{b}^{\mathrm{ror}}$) Sample $m_1 \xleftarrow{r} \{m \in P \mid |m| = |m_0|\}$, compute $(st', c) \leftarrow \mathcal{E}(st, m_\beta)$ and send c.
   (c) Update the record to $(i, st', l+1)$.
6. On the *reveal* query (x, π), do:
   (a) If $\mathsf{x} = \mathsf{msg}$, $\pi = (i, l)$ (*message reveal*) and (msg, i, l, m) is recorded, send (i, s_0).
   (b) If $\mathsf{x} = \mathsf{ltk}$, $\pi = i$ (*long-term key reveal*) and (i, pw) is recorded, send pw.
   (c) Otherwise, send $\perp$.

Eventually, the adversary outputs $\beta' \in \{0,1\}$.

Figure 4: Experiment $\mathbf{Exp}^{\mathsf{ror}}_{\mathrm{PBMSG}}$ for the real-or-random game for a password-authenticated secure channel protocol. The bit β'' is not used in the experiment, but is used to simplify the calculation of advantage.

We require that if the network transcripts of two instances using the same password have opposite roles and are otherwise identical, then their message transcripts have opposite roles and are otherwise identical.

Remark 27. Note that the initiator and the responder send messages, but these are network messages, not the messages that are actually sent between the users of the secure channel protocol.

The Experiment The experiment is shown in Figure 4. It is mostly straightforward. The only curious thing is that a chosen ciphertext query does not immediately reveal the result of the decryption to the adversary. Instead, the adversary must explicitly reveal the result. This gives the adversary greater access to the decryption algorithm, without making the game trivial.

Partnering Two instances are partners if they agree on the network messages sent as part of setting up the conversation, except that the last messages sent may not have arrived yet. Note that we require partners to have opposite roles.

Two instances with network transcripts $(0, ad, \mathbf{s}^{(0)}, \mathbf{c}_0^{(0)}, \mathbf{c}_1^{(0)})$ and $(1, ad, \mathbf{s}^{(1)}, \mathbf{c}_0^{(1)}, \mathbf{c}_1^{(1)})$ using the same password are *partners* if both instances are accepting and $\mathbf{s}^{(0)} = \mathbf{s}^{(1)}$, or if for some ρ, that instance is accepting and $\mathbf{s}^{(1-\rho)}$ is a prefix of $\mathbf{s}^{(\rho)}$, missing only the last message.

Authentication As usual, we would like to require that every accepting instance has a partner, but this cannot work, since the adversary may know or learn passwords. Instead, we account for this fact by counting the number of instances without a partner and hoping that this corresponds to the number of passwords the adversary knows.

Authentication holds if every accepting instance has at most one partner. The adversary's *success rate* is the number of accepting instances without a partner.

Integrity Integrity is only relevant for instances with partners, and the idea is that anything that one party receives was sent by the other party. Morally, what we want is plaintext integrity, so that the message sequence received by one party is a prefix of the message sequence sent by the other party. We instead define ciphertext integrity, which anyway implies plaintext integrity via correctness.

Integrity holds for two partner instances with network transcripts $(0, ad, \mathbf{s}^{(0)}, \mathbf{c}_0^{(0)}, \mathbf{c}_1^{(0)})$ and $(1, ad, \mathbf{s}^{(1)}, \mathbf{c}_0^{(1)}, \mathbf{c}_1^{(1)})$, if $\mathbf{c}_0^{(1)}$ is a prefix of $\mathbf{c}_0^{(0)}$ and $\mathbf{c}_1^{(0)}$ is a prefix of $\mathbf{c}_1^{(1)}$.

Freshness We model confidentiality using a real-or-random challenge query. As usual, we must require that challenge queries are only made for instances with partners. We want to allow the adversary the ability to decrypt chosen ciphertexts. Because our encryption is stateful, this means that the adversary must ask honest instances to decrypt the ciphertexts produced by challenge

queries. But the adversary should not see the result of these decryptions. To allow for this, we do not immediately return the result of any chosen ciphertext query. Instead, the adversary will have to reveal the result of each query.

A challenge query for some instance is *fresh* if the instance has a partner and any corresponding chosen ciphertext query for the partner instance has not been revealed.

An execution is *fresh* if every challenge query in the execution is fresh.

The Definition The notion of security is straight-forward. We want confidentiality of messages, up to message length, captured through the challenge queries. And we want integrity, which implies agreement on ciphertexts and their order, up to the most recent ciphertexts not being received. As usual, the adversary consists of a pair of interactive algorithms, one of which decides the passwords while the other does the actual attack on the cryptography.

Definition 28. A $(\tau, n_c, n_u, n_s, n_e)$-*adversary* against a password-authenticated secure channel protocol PBMSG is a pair of interactive algorithms $(\mathcal{G}, \mathcal{A})$ that interacts with the experiment in Figure 4, where $\mathcal{G}$ makes at most n_u password queries, $\mathcal{A}$ makes at most n_c challenge queries, n_s execute queries and n_e chosen plaintext and ciphertext queries, and where the runtime of the adversary and the experiment is at most τ.

The *advantage* of the adversary $(\mathcal{G}, \mathcal{A})$ against PBMSG is

$$\mathbf{Adv}^{\mathsf{ror}}_{\text{PBMSG}}(\mathcal{G}, \mathcal{A}) = \max\{2|\Pr[E_c] - 1/2|, \Pr[E_i], \Pr[E_a]\},$$

where E_c is the event that the adversary's guess equals β if the execution is fresh, or that the adversary's guess equals β'' if the execution is not fresh; E_i is the event that integrity does not hold for some pair of partnered instances; and E_a is the event that the authentication does not hold for some accepting instance.

The *success rate* $\mathbf{SuccR}^{\mathsf{pbmsg}}_{\text{PBMSG}}(\mathcal{G}, \mathcal{A})$ of the adversary $(\mathcal{G}, \mathcal{A})$ against PBMSG is the number of accepting instances using distinct, unrevealed password indexes which do not have partners.

Let $\mathcal{X}$ be a probability space on W. A *simple* $(\tau, n_c, n_u, n_s, n_e)$-*adversary for* $\mathcal{X}$ against a password-authenticated secure channel protocol PBMSG is an interactive algorithm $\mathcal{A}$ such that $(\mathcal{G}_{\mathcal{X}}, \mathcal{A})$ is a $(\tau, n_c, n_u, n_s, n_e)$-adversary against PBMSG.

We now prove the password-authenticated secure channel protocol from Example 22 secure. Combining this result with Propositions 10 and Propositions 12 gives us a desired theorem for the overall construction.

Proposition 23. *Consider the scheme* PBMSG *from Example 22 based on a password-authenticated key exchange scheme* PAKE *and a symmetric cryptosystem* SYM. *Let* $(\mathcal{G}, \mathcal{A})$ *be a context-respecting* $(\tau, n_c, n_u, n_s, n_e)$-*adversary against* PBMSG. *Then there exists a context-respecting* $(\tau'_1, 0, n_u, n_s)$-*adversary* $(\mathcal{G}, \mathcal{B}_1)$ *against* PAKE, *a context-respecting* $(\tau'_2, 0, n_u, n_s)$-*adversary* $(\mathcal{G}, \mathcal{B}_2)$ *against* PAKE,

context-respecting (τ_3', n_s, n_u, n_s)*-adversaries* $(\mathcal{G}, \mathcal{B}_3)$ *and* $(\mathcal{G}, \mathcal{B}_3')$ *against* PAKE*, a multi-key* $(\tau_4', n_u, n_e+n_c, n_e+n_c)$*-adversary* $\mathcal{B}_4$ *against ciphertext integrity for* SYM *and a multi-key* $(\tau_5', n_u, n_c, n_e+n_c, n_e)$*-adversary* $\mathcal{B}_5$ *against real-or-random for* SYM*, with* τ_1'*,* τ_2'*,* τ_3'*,* τ_4' *and* τ_5' *essentially equal to* τ*, such that*

$$\begin{aligned}\boldsymbol{Adv}^{\mathsf{ror}}_{PBMSG}(\mathcal{G}, \mathcal{A}) \leq\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_2) + \\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_3) + 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_3') + \\ & 2\boldsymbol{Adv}^{\mathsf{int\text{-}ctxt}}_{SYM}(\mathcal{B}_4) + \boldsymbol{Adv}^{\mathsf{ind\text{-}cca}}_{SYM}(\mathcal{B}_5) \text{ and,} \\ \boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}(\mathcal{G}, \mathcal{B}_1) \leq\ & \boldsymbol{SuccR}^{\mathsf{pbmsg}}_{PBMSG}(\mathcal{G}, \mathcal{A}).\end{aligned}$$

The idea is that the adversary will have to break the password-authenticated key exchange protocol in order to break authentication or learn anything about the session keys. Ciphertext integrity for the symmetric scheme then guarantees integrity for the conversation. Finally, confidentiality for the symmetric scheme guarantees confidentiality for the messages. We sketch the proof.

Sketch of proof of Proposition 23 The proof is organised as a sequence of games. Let $E_{c,i}$, $E_{i,i}$ and $E_{a,i}$ be the events in Game i corresponding to E_c, E_i and E_a, respectively, and let τ_i denote the runtime of Game i.

We start with the usual game where the adversary interacts with the experiment. The adversary's success rate can be bounded in terms of an adversary against the password-authenticated key exchange.

The first modification to the game is to stop if any instance ever has more than one partner. This ensures that authentication holds for every instance. If this change is noticeable, we get an adversary against authentication for the password-authenticated key exchange.

Next, we replace the session keys of instances with partners by random session keys. At this point, we no longer need to care about the password-authenticated key exchange. If this change is noticeable, we get an adversary against confidentiality for the password-authenticated key exchange.

We then stop if an instance with a partner ever accepts a ciphertext that the partner did not send, or accepts it out of order. This guarantees that integrity holds. If this change is noticeable, we get an adversary against ciphertext integrity for the symmetric scheme.

Finally, we observe that the resulting game is essentially the real-or-random game for a symmetric scheme, which we use to bound the adversary's confidentiality advantage.

Game 0 This game is the adversary $(\mathcal{G}, \mathcal{A})$ interacting with the experiment $\mathbf{Exp}^{\mathsf{ror}}_{\mathrm{PBMSG}}$.

The runtime τ_0 of this game is τ.

We immediately account for the adversary's success rate by construction an adversary against the password-authenticated key exchange with the same success rate.

Lemma 29. *There exists a context-respecting $(\tau_1', 0, n_u, n_s)$-adversary $(\mathcal{G}, \mathcal{B}_1)$ against* PAKE*, with τ_1' essentially equal to τ_0, such that*

$$\boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}_1) \leq \boldsymbol{SuccR}^{\mathsf{pbmsg}}_{\mathrm{PBMSG}}(\mathcal{G}, \mathcal{A}).$$

Game 1 In this game, we stop if any instance ever has more than one partner.

This requires some accounting, but τ_1 is essentially equal to τ_0.

In this and future games, whenever an instance has a partner, it is unique.

Let F_1 be the event that we stop in this game. It is immediate that $\Pr[E_{a,1}] = 0$, and we also get that

$$\Pr[E_{a,0}] \leq \Pr[F_1]. \tag{22}$$

and

$$|\Pr[E_{c,1}] - \Pr[E_{c,0}]| \leq \Pr[F_1], \qquad |\Pr[E_{i,1} - E_{i,0}]| \leq \Pr[F_1]. \tag{23}$$

Lemma 30. *There exists a context-respecting $(\tau_1', 0, n_u, n_s)$-adversary $(\mathcal{G}, \mathcal{B}_2)$ against* PAKE*, with τ_2' essentially equal to τ, such that*

$$\Pr[F_1] = \boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}_2).$$

Game 2 In this game, when an instance outputs a state and the instance has a partner, we discard the key in the state. If the partner has output a state, we instead use the key from that state. Otherwise, we sample a key from K and use that.

This requires some accounting, but τ_2 is essentially equal to τ_1.

In this game, the messages encrypted and decrypted with symmetric encryption are independent of the password-authenticated key exchange.

Lemma 31. *There exists context-respecting (τ_3', n_s, n_u, n_s)-adversaries $(\mathcal{G}, \mathcal{B}_3)$ and $(\mathcal{G}, \mathcal{B}_3')$ against* PAKE*, with τ_3' essentially equal to τ_2, such that*

$$|\Pr[E_{c,2}] - \Pr[E_{c,1}]| \leq \boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}_3),$$
$$|\Pr[E_{i,2}] - \Pr[E_{i,1}]| \leq \boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}_3').$$

Game 3 In this game, we stop if an instance with a partner ever successfully decrypts a ciphertext not sent by its partner, or successfully decrypts ciphertexts sent by its partner in the wrong order.

This requires some accounting, but τ_3 is essentially equal to τ_2.

In this game, integrity failures never happen and

$$\Pr[E_{i,3}] = 0. \tag{24}$$

Let F_3 be the event that we stop in this game. We get that

$$|\Pr[E_{c,3}] - \Pr[E_{c,2}]| \leq \Pr[F_3], \qquad E_{i,2} = \Pr[F_3]. \tag{25}$$

Lemma 32. *There exists a multi-key $(\tau_4', n_s, n_e + n_c, n_e + n_c)$-adversary $\mathcal{B}_4$ against ciphertext integrity for* SYM*, with τ_4' essentially equal to τ_3, such that*

$$\Pr[F_3] = \boldsymbol{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathrm{SYM}}(\mathcal{B}_4).$$

Conclusion We have now bounded E_a and E_i, so what remains is to bound the probability that the adversary correctly guesses the challenge bit.

Proposition 23 then follows from equations (22)–(25) and Lemmas 29 to 33.

Lemma 33. *There exists a multi-key* $(\tau_5', n_s, n_c, n_e{+}n_c, n_e)$*-adversary* $\mathcal{B}_5$ *against real-or-random for* SYM*, with* τ_5' *essentially equal to* τ_3*, such that*

$$2|E_{c,3} - 1/2| \leq \boldsymbol{Adv}^{\mathsf{ind\text{-}cca}}_{SYM}(\mathcal{B}_5).$$

A.2 Device Authorisation: Details and Sketches

Security for a device authorisation protocol, as defined, is fairly straight-forward. We want confidentiality, integrity and authentication for the connections. We model confidentiality as real-or-random security, in the sense that the adversary cannot tell if a ciphertext contains a message of their choice or a randomly chosen message of the same length. Integrity is ciphertext integrity, which essentially states that an instance of the connection protocol will only accept the ciphertexts sent by its parner, in the same order they were sent. Authentication says essentially that whenever we establish a token, the password is known to the partner, and whenever we establish a connection, the token is known to the partner.

Definition 34. Consider an instance of role ρ where the initiator or responder algorithm sent and received a sequence of messages $\mathbf{s} = (s_1, s_2, \ldots, s_n)$. It has the *transcript* $(\rho, ad, \mathbf{s})$.

An instance of the connection protocol first runs the handshake algorithm to get a state, and then runs the encryption and decryption algorithms multiple times, each time updating the state. The *role* of the instance is the same as the role of the token input to the handshake algorithm. Consider an instance of role ρ that encrypts a sequence of messages $\mathbf{m}_\rho = (m_{\rho,1}, \ldots, m_{\rho,n_\rho})$ to ciphertexts $\mathbf{c}_\rho = (c_{\rho,1}, \ldots, c_{\rho,n_\rho})$, and decrypts a sequence of ciphertexts $\mathbf{c}_{1-\rho} = (c_{1-\rho,1}, \ldots, c_{1-\rho,n_{1-\rho}})$ to a sequence of messages $\mathbf{m}_{1-\rho} = (m_{1-\rho,1}, \ldots, m_{1-\rho,n_{1-\rho}})$. The *message transcript* of an instance is $(\rho, ad, \mathbf{m}_0, \mathbf{m}_1)$. The *network transcript* of an instance is $(\rho, ad, \mathbf{c}_0, \mathbf{c}_1)$. (Note that the encryptions and decryptions may be interleaved arbitrarily, but the transcripts do not record this information.)

We require that if two authentication instances outputting two tokens have the same transcripts, except with opposite roles, and if two connection instances run with the output tokens have the same network transcripts, then their message transcripts are identical, except with roles corresponding to their tokens.

The Experiment The experiment is shown in Figure 5. It is very similar to the secure channel experiment in Figure 4, but adds queries to establish connections and reveal tokens.

Partnering Two authorisation instances are partners if they agree on the associated data and the messages sent, except that the last messages sent may not have arrived yet. Note that we require partners to have opposite roles.

The real-or-random experiment proceeds as follows:

1. Sample $\beta, \beta'' \xleftarrow{r} \{0,1\}$.
2. On the jth *password* query pw, do:
   (a) Record (j, pw) and send j.
3. On the ith *execute* query (ρ, j, ad), do:
   (a) If (j, pw) is not recorded, send $\perp$ and stop.
   (b) If $\rho = 0$, start the ith instance as $\mathcal{I}(ad, pw)$. When the instance sends s, send (i, s). Otherwise, start the ith instance as $\mathcal{R}(ad, pw)$ and send i.
4. On the *send* query (i, s), do:
   (a) If (i, tok) is recorded, send $\perp$.
   (b) Otherwise, send s to the ith instance. If the ith instance outputs $\perp$, record $(i, \perp)$ and send $(i, \perp)$. If the ith instance outputs tok, record (tok, i, tok). If the instance sent a message s' and did not output a state, send (i, s'). If the instance sent a message s' and output a state, send $(i, \top, s')$. If the instance output a state and did not send a message, send $(i, \top)$.
5. On the lth *handshake* query (hshk, i, ad), do:
   (a) If (tok, i, tok) is recorded, compute $st \leftarrow \mathcal{H}(tok, ad)$, record $(l, i, st, 0)$ and send $\top$ to the adversary.
   (b) Otherwise, send $\perp$ to the adversary.
6. On a *chosen plaintext* / *chosen ciphertext* / *challenge* query (cp, l, m) / (cc, l, c) / (ror, l, m_0), do:
   (a) If $(l, \cdot, st, \nu)$ is not recorded, send $\perp$ and stop.
   (b^{cp}) Compute $(st', c) \leftarrow \mathcal{E}(st, m)$ and send c.
   (b^{cc}) Compute $(st', m) \leftarrow \mathcal{D}(st, c)$, record $(\mathsf{msg}, l, \nu, m)$ and send $\top$. If decryption failed, update the record to $(l, \perp)$, send $\perp$ and stop.
   ($\mathrm{b}^{\mathrm{ror}}$) Sample $m_1 \xleftarrow{r} \{m \in P \mid |m| = |m_0|\}$, compute $(st', c) \leftarrow \mathcal{E}(st, m_\beta)$ and send c.
   (c) Update the record to $(l, i, st', \nu + 1)$.
7. On the *reveal* query (x, π), do:
   (a) If $\mathsf{x} = \mathsf{msg}$, $\pi = (l, \nu)$ (*message reveal*) and $(\mathsf{msg}, l, \nu, m)$ is recorded, send (i, s_0).
   (b) If $\mathsf{x} = \mathsf{tok}$, $\pi = i$ (*token reveal*) and (tok, i, tok) is recorded, send tok.
   (c) If $\mathsf{x} = \mathsf{ltk}$, $\pi = i$ (*long-term key reveal*) and (i, pw) is recorded, send pw.
   (d) Otherwise, send $\perp$.

Eventually, the adversary outputs $\beta' \in \{0,1\}$.

Figure 5: Experiment $\mathbf{Exp}^{\mathsf{ror}}_{\mathrm{PBAUTH}}$ for the real-or-random game for a password-authenticated device authorisation. The bit β'' is not used in the experiment, but is used to simplify the calculation of advantage.

Two authorisation instances using the same password with transcripts $(0, ad, \mathbf{s}^{(0)})$ and $(1, ad, \mathbf{s}^{(1)})$ are *partners* if $\mathbf{s}^{(0)} = \mathbf{s}^{(1)}$, or if for some ρ, that instance output a token and $\mathbf{s}^{(1-\rho)}$ is a prefix of $\mathbf{s}^{(\rho)}$, missing only the last message.

Two connection instances are *partners* if they derived their initial state from tokens output by authorisation instance partners.

Integrity Integrity is only relevant for connection instances with partners, and as for secure channels, we define only ciphertext integrity.

Integrity holds for two partner connection instances with network transcripts $(0, ad, \mathbf{c}_0^{(0)}, \mathbf{c}_1^{(0)})$ and $(0, ad, \mathbf{c}_0^{(1)}, \mathbf{c}_1^{(1)})$, if $\mathbf{c}_0^{(1)}$ is a prefix of $\mathbf{c}_0^{(0)}$ and $\mathbf{c}_1^{(0)}$ is a prefix of $\mathbf{c}_1^{(1)}$.

Freshness We model confidentiality using a real-or-random challenge query in much the same way as for secure channels.

A challenge query for some connection instance is *fresh* if the instance has a partner, any corresponding chosen ciphertext query for the partner instance has not been revealed and neither token has been revealed.

The Definition By this point, the definition is a straight-forward exercise.

Definition 35. A $(\tau, n_c, n_u, n_s, n'_s, n_e)$-*adversary* against a password-authenticated device authorisation protocol PBAUTH is a pair of interactive algorithms $(\mathcal{G}, \mathcal{A})$ that interacts with the experiment in Figure 5, where $\mathcal{G}$ makes at most n_u password queries, $\mathcal{A}$ makes at most n_c challenge queries, n_s execute queries, n'_s handshake queries and n_e chosen plaintext and ciphertext queries, and where the runtime of the adversary and the experiement is at most τ.

The *advantage* of the adversary $(\mathcal{G}, \mathcal{A})$ against PBAUTH is

$$\mathbf{Adv}^{\mathsf{ror}}_{\text{PBAUTH}}(\mathcal{G}, \mathcal{A}) = \max\{2|\Pr[E_c] - 1/2|, \Pr[E_i], \Pr[E_a]\},$$

where E_c is the event that the adversary's guess equals β if the execution is fresh, or that the adversary's guess equals β'' if the execution is not fresh; E_i is the event that integrity does not hold for some pair of partnered connection instances; and E_a is the event that the authentication does not hold for some instance that outputs a token.

The *success rate* $\mathbf{SuccR}^{\mathsf{pbauth}}_{\text{PBAUTH}}()$ of the adversary $(\mathcal{G}, \mathcal{A})$ against PBAUTH is the number of accepting authorisation instances using distinct, unrevealed password indexes which do not have partners.

Let $\mathcal{X}$ be a probability space on W. A *simple* $(\tau, n_c, n_u, n_s, n'_s, n_e)$-*adversary for* $\mathcal{X}$ against a password-authenticated authorisation protocol PBAUTH is an interactive algorithm $\mathcal{A}$ such that $(\mathcal{G}_\mathcal{X}, \mathcal{A})$ is a $(\tau, n_c, n_u, n_s, n'_s, n_e)$-adversary against PBAUTH.

We now prove the password-authenticated device authorisation protocol from Example 25 secure. Combining this result with Propositions 10 and Propositions 12 gives us a desired theorem for the overall construction.

Proposition 26. *Consider the scheme* PBAUTH *from Example 25 based on a password-authenticated key exchange scheme* PAKE*, a symmetric cryptosystem* SYM *and a key derivation function kdf. Let* $(\mathcal{G}, \mathcal{A})$ *be a context-respecting* $(\tau, n_c, n_u, n_s, n_e)$*-adversary against* PBAUTH*, with kdf modelled as a random oracle. Then there exists a context-respecting* $(\tau'_1, 0, n_u, n_s)$*-adversary* $(\mathcal{G}, \mathcal{B}_1)$ *against* PAKE*, a context-respecting* $(\tau'_2, 0, n_u, n_s)$*-adversary* $(\mathcal{G}, \mathcal{B}_2)$ *against* PAKE*, context-respecting* (τ'_3, n_s, n_u, n_s)*-adversaries* $(\mathcal{G}, \mathcal{B}_3)$ *and* $(\mathcal{G}, \mathcal{B}'_3)$ *against* PAKE*, a context-respecting multi-key* $(\tau'_4, n_u, n_e + n_c, n_e + n_c)$*-adversary* $\mathcal{B}_4$ *against ciphertext integrity for* SYM *and a context-respecting multi-key* $(\tau'_5, n_u, n_c, n_e + n_c, n_e)$*-adversary* $\mathcal{B}_5$ *against real-or-random for* SYM*, with* τ'_1*,* τ'_2*,* τ'_3*,* τ'_4 *and* τ'_5 *essentially equal to* τ*, such that*

$$\begin{aligned}\boldsymbol{Adv}^{\mathsf{ror}}_{\mathrm{PBAUTH}}(\mathcal{G}, \mathcal{A}) \leq\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}_2) + \\ & 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}_3) + 2\boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{\mathrm{PAKE}}(\mathcal{G}, \mathcal{B}'_3) + \\ & 2\boldsymbol{Adv}^{\mathsf{int\text{-}ctxt}}_{\mathrm{SYM}}(\mathcal{B}_4) + \boldsymbol{Adv}^{\mathsf{ind\text{-}cca}}_{\mathrm{SYM}}(\mathcal{B}_5) \text{ and,} \\ \boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}(\mathcal{G}, \mathcal{B}_1) \leq\ & \boldsymbol{SuccR}^{\mathsf{pbauth}}_{\mathrm{PBAUTH}}(\mathcal{G}, \mathcal{A}).\end{aligned}$$

The idea is that the adversary will have to break the password-authenticated key exchange protocol in order to break authentication or learn anything about the tokens. Since we have modelled *kdf* as a random oracle, then as long as a token has not been revealed and associated data is not reused (nonce-respecting), the adversary cannot learn anything about the session key used for a connection. Ciphertext integrity for the symmetric scheme then guarantees integrity for the conversation. Finally, confidentiality for the symmetric scheme guarantees confidentiality for the messages. We sketch the proof.

Sketch of proof of Proposition 26 The proof is organised as a sequence of games. Let $E_{c,i}$, $E_{i,i}$ and $E_{a,i}$ be the events in Game i corresponding to E_c, E_i and E_a, respectively, and let τ_i denote the runtime of Game i.

We start with the usual game where the adversary interacts with the experiment. The adversary's success rate can be bounded in terms of an adversary against the password-authenticated key exchange.

The first modification to the game is to stop if any instance ever has more than one partner. This ensures that authentication holds for every instance. If this change is noticeable, we get an adversary against authentication for the password-authenticated key exchange.

Next, we stop if the adversary queries the *kdf* random oracle with the session key from a partnered token where the token and its partner token are both unrevealed. Because we model *kdf* as a random oracle, we have now cryptographically separated the symmetric keys derived for connections from the password-authenticated key exchange. This allows us to use random symmetric keys as in the symmetric cryptography security experiments.

We then stop if a connection instance with a partner ever accepts a ciphertext that the partner did not send, or accepts it out of order. This guarantees

that integrity holds. If this change is noticeable, we get an adversary against ciphertext integrity for the symmetric scheme.

Finally, we observe that the resulting game is essentially the real-or-random game for a symmetric scheme, which we use to bound the adversary's confidentiality advantage.

Remark 28. Note that this proof requires key reveals in the multi-key symmetric encryption security games. The reason is that while the adversary cannot reveal tokens after making challenge queries, the adversary may have session keys derived from tokens and used for encryption and decryption before deciding whether to reveal or challenge. This means that the experiment has, essentially, to commit to the symmetric key.

The theorem relating the security of multi-key symmetric encryption with key reveals to ordinary single-key symmetric encryption is somewhat non-tight, but it is not very bad.

If we replaced the handshake algorithm with a proper key exchange algorithm, as dicussed above, we could avoid this issue.

Game 0 This game is the adversary $(\mathcal{G}, \mathcal{A})$ interacting with the experiment $\mathbf{Exp}^{\mathsf{ror}}_{\text{PBAUTH}}$.

The runtime τ_0 is τ.

Lemma 36. *There exists a context-respecting $(\tau_1', 0, n_u, n_s)$-adversary $(\mathcal{G}, \mathcal{B}_1)$ against PAKE, with τ_1' essentially equal to τ_0, such that*

$$\boldsymbol{SuccR}^{\mathsf{pake\text{-}e}}_{PAKE}(\mathcal{G}, \mathcal{B}_1) \leq \boldsymbol{SuccR}^{\mathsf{pbauth}}_{PBAUTH}(\mathcal{G}, \mathcal{A}).$$

Game 1 In this game, we stop if any authorisation instance ever has more than one partner.

This requires some accounting, but τ_1 is essentially equal to τ_0.

In this and future games, whenever an instance has a partner, it is unique.

Let F_1 be the event that we stop in this game. It is immediate that $\Pr[E_{a,1}] = 0$, and we also get that

$$\Pr[E_{a,0}] \leq \Pr[F_1]. \tag{26}$$

and

$$|\Pr[E_{c,1}] - \Pr[E_{c,0}]| \leq \Pr[F_1], \qquad |\Pr[E_{i,1} - E_{i,0}]| \leq \Pr[F_1]. \tag{27}$$

Lemma 37. *There exists a context-respecting $(\tau_1', 0, n_u, n_s)$-adversary $(\mathcal{G}, \mathcal{B}_2)$ against PAKE, with τ_2' essentially equal to τ_1, such that*

$$\Pr[F_1] = \boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_2).$$

Game 2 In this game, if the adversary queries the *kdf* random oracle with the secret k from a partnered token where both tokens are unrevealed, we stop the game.

This requires some accounting, but τ_2 is essentially equal to τ_1.

In this game, from the adversary's point of view, the messages encrypted and decrypted with symmetric encryption are independent of the password-authenticated key exchange.

Let F_2 be the event that we stop in this game. We get that

$$|\Pr[E_{c,2}] - \Pr[E_{c,1}]| \leq \Pr[F_2], \qquad |\Pr[E_{i,2} - E_{i,1}]| \leq \Pr[F_2]. \tag{28}$$

Lemma 38. *There exists a context-respecting (τ_3', n_s, n_u, n_s)-adversary $(\mathcal{G}, \mathcal{B}_3)$ against PAKE, with τ_3' essentially equal to τ_2, such that*

$$\Pr[F_2] \leq \boldsymbol{Adv}^{\mathsf{pake\text{-}ror}}_{PAKE}(\mathcal{G}, \mathcal{B}_3) + \frac{n_h n_s}{|K|}.$$

Game 3 In this game, we stop if an instance with a partner ever successfully decrypts a ciphertext not sent by its partner, or successfully decrypts ciphertexts sent by its partner in the wrong order.

This requires some accounting, but τ_3 is essentially equal to τ_2.

In this game, integrity failures never happen and

$$\Pr[E_{i,3}] = 0. \tag{29}$$

Let F_3 be the event that we stop in this game. We get that

$$|\Pr[E_{c,3}] - \Pr[E_{c,2}]| \leq \Pr[F_3], \qquad E_{i,2} = \Pr[F_3]. \tag{30}$$

Lemma 39. *There exists a multi-key $(\tau_4', n_s', n_e + n_c, n_e + n_c)$-adversary $\mathcal{B}_4$ against ciphertext integrity for SYM, with τ_4' essentially equal to τ_3, such that*

$$\Pr[F_3] = \boldsymbol{Adv}^{\mathsf{int\text{-}ctxt}}_{SYM}(\mathcal{B}_4).$$

Conclusion We have now bounded E_a and E_i, so what remains is to bound the probability that the adversary correctly guesses the challenge bit.

Proposition 26 then follows from equations (26)–(30) and Lemmas 36 to 40.

Lemma 40. *There exists a multi-key $(\tau_5', n_s', n_c, n_e+n_c, n_e)$-adversary $\mathcal{B}_5$ against real-or-random for SYM, with τ_5' essentially equal to τ_3, such that*

$$2|E_{c,3} - 1/2| \leq \boldsymbol{Adv}^{\mathsf{ind\text{-}cca}}_{SYM}(\mathcal{B}_5).$$