Bare PAKE: Universally Composable Key Exchange from Just Passwords

Manuel Barbosa[1,5(✉)], Kai Gellert[2], Julia Hesse[3], and Stanislaw Jarecki[4]

[1] University of Porto (FCUP) and INESC TEC, Porto, Portugal
mbb@fc.up.pt
[2] University of Wuppertal, Wuppertal, Germany
kai.gellert@uni-wuppertal.de
[3] IBM Research Europe - Zurich, Rüschlikon, Switzerland
jhs@zurich.ibm.com
[4] University of California Irvine, Irvine, USA
sjarecki@uci.edu
[5] Max Planck Institute for Security and Privacy, Bochum, Germany

Abstract. In the past three decades, an impressive body of knowledge has been built around secure and private password authentication. In particular, secure password-authenticated key exchange (PAKE) protocols require only minimal overhead over a classical Diffie-Hellman key exchange. PAKEs are also known to fulfill strong composable security guarantees that capture many password-specific concerns such as password correlations or password mistyping, to name only a few. However, to enjoy both round-optimality and strong security, applications of PAKE protocols must provide *unique* session and participant identifiers. If such identifiers are not readily available, they must be agreed upon at the cost of additional communication flows, a fact which has been met with incomprehension among practitioners, and which hindered the adoption of provably secure password authentication in practice.

In this work, we resolve this issue by proposing a new paradigm for truly *password-only* yet securely composable PAKE, called *bare* PAKE. We formally prove that two prominent PAKE protocols, namely CPace and EKE, can be cast as bare PAKEs and hence do not require pre-agreement of anything else than a password. Our bare PAKE modeling further allows to investigate a novel "reusability" property of PAKEs, i.e., whether n^2 pairwise keys can be exchanged from only n messages, just as the Diffie-Hellman non-interactive key exchange can do in a public-key setting. As a side contribution, this add-on property of bare PAKEs leads us to observe that some previous PAKE constructions relied on unnecessarily strong, "reusable" building blocks. By showing that "non-reusable" tools suffice for standard PAKE, we open a new path towards round-optimal post-quantum secure password-authenticated key exchange.

J. Hesse—The author was supported by the Swiss National Science Foundation (SNSF) under the AMBIZIONE grant "Cryptographic Protocols for Human Authentication and the IoT.

© International Association for Cryptologic Research 2024
L. Reyzin and D. Stebila (Eds.): CRYPTO 2024, LNCS 14921, pp. 183–217, 2024.
https://doi.org/10.1007/978-3-031-68379-4_6

1 Introduction

Passwords continue to be a dominant form of user authentication on the internet. While some alternatives to password-based authentication emerge, such as WebAuthn [44], passwords remain prevalent, primarily due to usability issues (see e.g. [12]). More broadly, there is no alternative to low-entropy secrets in applications where authentication is based on *something you know*, such as a Personal Identification Number (PIN). Finally, passwords also play a central role at the infrastructure level, where they are widely used for device authentication, e.g., when setting up local wireless networks and IoT devices [30].

Due to its practical significance, cryptographically secure password authentication is an active area of research that dates back to the seminal paper of Bellovin and Merritt [9], more than thirty years ago. These protocols aim at ensuring *minimal leakage* of passwords while still allowing for checking their correctness. However, despite the impressive body of knowledge that was built during this time, there has not been much adoption of provably secure password authentication in the real world. Almost all password-protected login sites still deploy the privacy elusive "password-over-TLS", where cleartext passwords are handed over to service providers, who hash the password and compare it with their database. For PIN-based authentication, ad-hoc solutions are also often deployed and are then subject to a-posteriori analyses (e.g., Password Authenticated Connection Establishment (PACE) [10], and the Client-to-Authenticator Protocol in FIDO2 [5,11]).

One reason for the adoption of provably secure password authentication to remain scarce is that there are almost no *specifications* that could serve as a basis for implementors to adopt this type of cryptography. Writing good specifications is challenging, and in the case of password authentication, it is further hindered by a gap between the security models used to formally analyze such protocols and their use in practice. In particular, existing models [1,17,22,26,31] impose on the implementation a notion of a *unique party identifier* and/or a *unique session identifier*, and it is often unclear [16] how these parameters should be instantiated in practice. Known ways to correctly instantiate them impose significant protocol costs. The consequences are either specifications to which proofs in the literature do not apply or ones that are potentially unnecessarily inefficient.

In this paper, we revise models for secure password authentication to optimize for specification simplicity and implementation efficiency. In particular, we dispense with the need for the above-mentioned identifiers, introduce a security notion that captures true *password-only* authentication, and show that password-only implementations of known protocols meet this notion. Overall, we simplify and clarify the requirements that an implementer must meet to deploy password-authenticated protocols, leading to standards that are easier to adopt.

Security Models for Password Authentication. Cryptographic literature models password authentication as a Password-Authenticated Key Exchange (PAKE) [9], which establishes a secure shared key between two participants if they run the protocol on the same password. PAKE security was first formalized

in a game-based model by Bellare-Pointcheval-Rogaway (BPR) [8]. The BPR model has been adopted in the analysis of many PAKE proposals, e.g., EKE [8], SPEKE [29,34,39], SPAKE2 [4], TBPEKE [40], and CPace [3]. However, the BPR model provides only limited assurance of real-world security, most importantly because it analyzes each user in separation, and assumes that every user chooses their password independently, while in the real world, password choices are often correlated, e.g. with prior passwords of the same user, or with passwords used by other family members, etc. To deal with these (and other) shortcomings of the BPR model, Canetti et al. [17] proposed a Universally Composable (UC) PAKE model which addresses password correlations, password mistyping, password information leakage, and arbitrary interactions between protocol instances. The UC model further ensures security under arbitrary protocol composition, enabling security arguments for protocols that use generic PAKE as a subroutine, e.g., generic compilers from symmetric PAKE to asymmetric PAKE (where one party, the server, knows only a password hash instead of the password itself) [26,33], or from asymmetric PAKE to strong asymmetric PAKE (where the server's password hash is privately salted) [35]. For these reasons the UC PAKE model has become a gold standard of PAKE security [1,3,13,26,27,31,35,41,42].

Limitations of the UC PAKE Model. Despite their widely recognized benefits, all previous UC PAKE models[1] come with hidden costs and fail to implement password-only authentication. Indeed, according to these models, each protocol participant $\mathcal{P}$ must supply three further inputs: i. a *session identifier sid* ii. a *party identifier*, and iii. a corresponding identifier $\mathcal{CP}$ of the intended counterparty in this protocol instance.[2]

The UC PAKE requirements on the session identifier are due to general conventions of the UC framework [14], namely that each protocol instance must be identified by a globally unique session identifier. The standard UC PAKE model [17] implies that PAKE participants can successfully establish a joint session key only if they use the same session identifier *sid*. Moreover, the requirement for global uniqueness implies no security guarantees to an honest party that re-uses the same *sid* string that was used by some pair of honest parties before. The requirements on party identifiers are similar: The UC framework assumes that each party that participates in the protocol has a unique identifier, i.e., that no two honest parties carry the same identifier $\mathcal{P}$.[3] The UC PAKE

[1] We use the term *standard UC PAKE* to denote the original notion of Canetti et al. [17], and *UC PAKE* for including variants thereof, e.g., [1,31].

[2] The same problem appears in the BPR model for party identifiers, but not for session identifiers, which are protocol outputs rather than inputs.

[3] For a protocol realizing an ideal functionality that runs an independent session, with a globally unique session identifier, for a small set of parties, this means only that each of these participants must have a different party identifier. However, when analyzing protocols that realize multi-instance versions of ideal functionalities that may interact with an arbitrary number of honest participants, the uniqueness of party identifiers becomes a global requirement.

model enforces the use of these identifiers in the protocol because it allows the parties to establish a key only if they use matching identifiers, i.e., if one party uses its own and counterparty identifiers $\mathcal{P}$ and $\mathcal{CP}$, then the other must use respectively $\mathcal{P}', \mathcal{CP}'$ s.t. $\mathcal{P}' = \mathcal{CP}$ and $\mathcal{CP}' = \mathcal{P}$.

Problems Caused by Requirements on $sid, \mathcal{P}, \mathcal{CP}$ Identifiers. All of these inputs are problematic for an implementer, and they cannot be set to "don't care" symbols because of the above requirements of the UC PAKE model. Regarding the globally unique *sid* requirement, the standard way of satisfying it, as suggested in [14], is to precede a PAKE instance by a *sid*-picking protocol, where *sid* is set as a concatenation of two fresh nonces which the parties send to each other. However, this adds a network flow to the resulting implementation. Anyone who attempts to implement a PAKE in any application would surely be annoyed by this requirement, in particular, because the only justification that researchers could give is that unique identifiers are required by the UC PAKE model. While there is no known attack on implementations that ignore the uniqueness of identifiers, or skip them completely, current proofs in the literature do not carry over to such more efficient and straightforward PAKEs.

The UC PAKE requirement that participants hold unique identifiers for themselves and their counterparties is also problematic. How should such identifiers be implemented? Should they be IP addresses? First, it is not clear what security guarantees the model implies if two honest parties happen to use the same self-identifier, and IP addresses can be dynamic. Second, if one party is behind a firewall then their counterparty will not know their IP address. Should the identifiers be DNS names instead? That is good for servers but not for clients. In summary, whatever an implementation chooses to use for these identifiers, it is possible that they turn out not to be unique, or it will tie the implementation too strongly to a particular networking setting, and it will break if the setting changes. In this aspect too, an implementer can ask why they need these unique identifiers at all as an input to the protocol, and a cryptographer's answer is currently the same as above, i.e. that current proofs do not imply that a PAKE is UC secure without them.

Given the troubles of pre-exchanging and agreeing on all these identifiers, the following research question repeatedly comes up while standardizing or implementing UC PAKE protocols:

Can PAKE protocols enjoy composable security without relying on pre-agreed unique session- and party identifiers?

Our Proposal: The UC bare PAKE Model. In this work, we answer the above question in the affirmative. We propose to shift from standard UC PAKE to *bare* PAKE (bPAKE), which models composable key exchange from *just* passwords. Our bPAKE model eliminates all the above implementation dilemmas regarding identifier choices, and consequently yields more practical protocols that do not require the application to pre-agree on anything other than a password: In our bare PAKE model the only required input *is* the password. Session

identifiers still appear in our model, albeit only as an *output* of a protocol. They can be thought of (and used as such by applications) as a protocol transcript that uniquely identifies one particular key exchange session.

If two parties want to use PAKE to establish an agreement not only on their passwords but also on identifiers that might not be pre-agreed before the protocol starts, bare PAKE allows each party to enter its name as an optional input, and such name becomes part of their counterparty output. This way an application can reject a session if it was established with the counterparty whose name does not match the application's criteria, but these names do not have to be pre-agreed, and instead they can be communicated as part of the PAKE protocol. Furthermore, the name each party supplies is an arbitrary value, does not have to be unique, and can be set to $\perp$.

In summary, our UC bare PAKE functionality is just like the standard UC PAKE functionality in that it password-authenticates not only a session key but also a session and counterparty identifiers, but the latter are *protocol outputs instead of inputs*. The UC bare PAKE model therefore offers a simpler and application-friendly PAKE syntax—we argue that it is even friendlier than that of the BPR model [8]—but it is a variant of the UC PAKE model of Canetti et al. [17], hence it assures all the good properties of the UC framework including security under arbitrary password correlations, concurrent executions, and arbitrary protocol composition.

Further Benefits and New Insights: Reusability. The UC bare PAKE model draws further benefits from the above simplifications. If a PAKE protocol considers a password as the only necessary input, and the session identifier and counterparty's optional name are protocol outputs, then it becomes possible for a bare PAKE protocol participant to reuse their state and create keys with several counterparties from one password, where each key is accompanied by a separate session identifier and counterparty's name. Most PAKE schemes from the literature are password-encoded variants of the Diffie-Hellman (DH) non-interactive key exchange (NIKE) [21], which is known to enable the exchange of n^2 pairwise keys between n users. Can these PAKEs, such as EKE [9,22] or CPace [3,28], allow for a similar flavor of reusability, i.e., allow parties to reuse their state to produce *password-authenticated* keys from many incoming messages instead of just one? The standard UC PAKE versions of these protocols could not do that, because they require each party to specify a unique session and counterparty identifier as an input for each exchanged key. By contrast, bare PAKE allows us to investigate if PAKEs can be reusable.

As an application example, consider a client C who does not remember which password she should use with server S, but holds n password candidates, while server S has a single password registered for this user. With a reusable bare PAKE, C can run n PAKE instances, one for each password candidate, while S runs a single instance on its single password. If S's PAKE message is processed by each of C's instances, the client computes n session keys, and when S's instance processes each of the n messages sent by C's instances, the server also computes n session keys, and then they can learn if any of C's passwords is correct using a

key confirmation protocol. One could use n independent instances of UC PAKE in the same application, but a reusable bare PAKE reduces the costs because S initializes only one PAKE instance and sends only one PAKE message to C.[4]

In another example, consider an Internet of Things (IoT) setting [45], where each IoT device at home or office is initialized with the same password. With a reusable bare PAKE, each device can broadcast a single message to the network and then establish a password-authenticated pairwise key with any other device after processing that device's PAKE message. Concretely, if EKE or CPace are secure UC bare PAKEs, then all devices will be able to establish password-authenticated pairwise keys at the cost of a single broadcast per party!

Our Contributions. Besides the formal modeling of the new bare PAKE paradigm, our paper contains several contributions that we summarize below.

(1) The UC bare PAKE Model. We define a UC bare PAKE model which reformulates the standard UC PAKE notion of Canetti et al. [17] to make PAKEs truly *password-only*. A PAKE scheme secure in the UC bare PAKE model does not need pre-established unique session or party identifiers, which removes unnecessary hurdles for implementers. We stress that we achieve this without sacrificing *any* composability guarantees: our definitional work guarantees that the standard UC composition theorem implies that any usage of bare PAKE in an application can be replaced with the bare PAKE ideal functionality in the security analysis. Furthermore, the bare PAKE model allows UC PAKEs to be *reusable*, i.e., it allows for a single PAKE instance to establish password-authenticated session keys with an arbitrary number of parties. In our presentation we rely on the notion of structured UC protocols [15, eprint 2020 version] to clarify the subtle syntactical issues of what it means for a UC protocol not to depend on the local party and session identifiers, even though these technical artifacts are inherent to a security analysis in the UC framework.

(2) Compilers. Notably, the input/output interfaces of our bare PAKE model and previous models differ, e.g., standard UC PAKE requires a new password input for each key exchange while bare PAKE can exchange many keys from one input password. It is hence not possible to say that the UC model of bare PAKE is stronger than the one for standard PAKE, or vice versa. To demonstrate the usefulness of our bare PAKE notions, we thus give generic and simple compilers between them. First, we show that any UC bare PAKE can be generically converted to a standard UC PAKE, by simply running the bare PAKE on $\bar{pw} = pw||sid||\mathcal{P}||\mathcal{CP}$. This compiler justifies our bare PAKE approach in the sense that our model is strong enough to realize standard PAKE, essentially without a performance penalty. The compiler further offers a smooth migration path towards the bare PAKE paradigm: applications that for some reason *do* have unique session- and party identifiers at hand get the expected standard PAKE interface from a bare PAKE as well! Second, we show that any secure

[4] This application appeared in [36], which give a BPR-model analysis of a solution assuming pseudorandomness of PAKE protocol messages. Note that S can limit guessing attempts by upper-bounding the number n of C's instances it processes.

standard UC PAKE implies a *one-time* UC bare PAKE, where one-time means that a party terminates a password instance after producing one key from it. The compiler lets parties exchange unique session- and party identifiers before executing the standard UC PAKE with them. In settings where no external uniqueness guarantee on party identifiers is available, our compiler reflects the overhead that implementations need to account for if only standard UC PAKE protocols are available.

(3) CPace is Secure Without Identifiers, and Reusable. We then investigate which PAKEs from the literature can be cast as bare PAKEs, i.e., can be deployed without unique session or party identifiers and establish keys from pre-agreed passwords alone. Of course, with our second compiler above, every standard UC PAKE can be converted into a bare PAKE, albeit at the cost of one additional round of nonce exchanges. We are instead looking for round-preserving bare versions of Diffie-Hellman-based PAKEs, i.e., that only take one round. We start with CPace [28], a PAKE that won the recent symmetric PAKE selection process [20] of the IRTF and which is currently undergoing standardization [2]. CPace lets parties perform a Diffie-Hellman key exchange with a group generator $g \leftarrow H(pw||sid||\mathcal{P}||\mathcal{CP})$ that encodes the password and all identifiers. We prove that "bare CPace" securely realizes our new notion with a generator $g \leftarrow H(pw)$ that just encodes the password. Our proof relies on the same assumptions taken in the analysis of its $(sid, \mathcal{P}, \mathcal{CP})$-dependent version [3]. Our result implies reusability and composable security guarantees of the Internet-Draft version of CPace [2] with $sid, \mathcal{P}, \mathcal{CP}$ all set to $\perp$. We recommend that the specification switches to bare CPace, as this would enable the authenticated transmission of party names without any uniqueness requirements.

(4) EKE is Secure Without Identifiers, and Reusable. We continue with the seminal Encrypted Key Exchange (EKE) protocol of Bellovin and Merrit [8,9], which was the first in line of the many PAKE protocols built upon the non-interactive Diffie-Hellman key exchange. The idea of EKE is to password-encrypt the DH public keys, i.e., using a symmetric cipher and the password as the encryption key. In our work, we consider a general "bare EKE compiler" which starts from any non-interactive key exchange (NIKE) and any symmetric cipher, to obtain a PAKE that requires only pre-agreed passwords and doesn't need any additional $sid, \mathcal{P}, \mathcal{CP}$ inputs. We prove that this compiler can transform *any* NIKE that satisfies the standard notion of security for these protocols [19,23] into a reusable bare PAKE in the ideal cipher (IC) model. Our result shows that EKE can enjoy composability without unique identifiers, and it can be used to exchange an arbitrary number of keys from just one password. As a corollary of this result we get a modular proof that *Hashed* Diffie-Hellman (Hashed DH) NIKE—i.e., a Diffie-Hellman NIKE where the pairwise key is computed as $K_{i,j} = H(g^{x_i \cdot x_j})$ where H is a hash function—can be compiled into a *reusable* bare PAKE using EKE. The resulting protocol is tightly secure under gap-CDH in the Ideal Cipher and Random Oracle Models.

(5) Towards Post-quantum Secure PAKE from NIKE. An interesting side observation of our work on the bare version of EKE is that the reusability aspect crucially relies on the fact that secret keys of the underlying NIKE must be secure to reuse. Because standard UC PAKE does *not* allow for such reuse, an immediate question is whether we can relax the assumptions on the NIKE when aiming for the non-reusable standard UC PAKE notion. Indeed, we can show that a "one-time" version of NIKE is enough to realize standard UC PAKE and, furthermore, that "one-time" NIKE follows from passively secure NIKE in the Random Oracle Model. As a corollary, we derive that Hashed DH NIKE is one-time secure under CDH in the ROM, a result that explains previous CDH-based EKE security proofs [22] with a nice flavor of modularity. These results also open a path for round-optimal post-quantum secure EKE instantiations based on, for example, the passively secure version of Swoosh [25].

Other Related Work. Shoup [43] carries out an analysis of a concrete PAKE protocol in a variant of the UC model that addresses some of the concerns we also address in this paper. In particular, globally unique session identifiers are already replaced by locally unique instance identifiers, and session identifiers are seen as protocol outputs. However, the protocols in this model are still not password-only due to the use of party identifiers.

Küsters and Tuengerthal [38] identify conditions on protocols under which single-session security implies security in the *local-sid* multi-session setting. For the case of PAKE the latter is like our bPAKE functionality, except it still requires globally unique party identifiers and does not support re-use. However, their results assume multi-round protocols, since the simulator for the single-session protocol must compute each party's first message by following their protocol. Moreover, the joint-state cryptographic tools they consider do not include Ideal Ciphers or Random Oracles, which are essential in resp. EKE and CPace.

The NIST post-quantum competition and the upcoming standardization of Kyber KEM have driven several proposals of black-box constructions of PAKE protocols from KEM [7,41]. The modularity of these constructions is aligned, and indeed closely related to, our analysis of EKE-NIKE. One substantial difference, however, is that we focus on reusable single-simultaneous-flow PAKEs, which we show to be easy to construct from simplified[5] NIKE. KEM-based constructions seem to allow only one-sided reusability by the party who generates the KEM public key; indeed, the other party is carrying out KEM encryption and stores no reusable state. We do not explore one-sided reusability in this paper, but our definitions already cater to this possibility for future work.

Dos Santos et al. [41] propose an alternative to the use of the Ideal Cipher in EKE variants that they call a *randomized Half-Ideal-Cipher*. This proposal solves an important problem in EKE, which is how to instantiate an Ideal Cipher over a structured data type such as a KEM public key, or a group element in DH-based systems. The Half-Ideal Cipher is essentially a two-round Feistel network, where a small part of the input (of size twice the security parameter) is

[5] In contrast to the standard notion of NIKE, where party identities are an input to the protocol, simplified NIKE has only public keys [24].

generated uniformly at random and encrypted using an Ideal Cipher that works over this small domain. The remaining (potentially large) part of the input, e.g., the KEM public key, is simply masked using a group operation according to the structure of the Feistel construction. We believe that the proof of the EKE-NIKE construction we present in this paper can be easily adapted to rely on the same half-ideal cipher construction, thereby simplifying its instantiation with post-quantum secure NIKEs such as Swoosh [25].

2 Preliminaries

Notation. We denote the security parameter as κ. We denote $[\ell] = \{1, \ldots, \ell\}$. We write $x \leftarrow_{\mathrm{R}} \mathcal{S}$ for sampling x from uniform distribution over set $\mathcal{S}$. For a probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$ we write $y \leftarrow_{\mathrm{R}} \mathcal{A}(x_1, \ldots, x_\ell)$ for assigning to y an output of a randomized execution of $\mathcal{A}$ on input $x_1, \ldots, x_\ell$. For any variable a, by *record* $\langle a, [b] \rangle$ we denote a 2-tuple of values, where the first value is equal to a, and we assign variable name b to the second entry. We use $\{a, b\}_{\mathsf{ord}}$ to denote string $a||b$ if $a \leq_{\mathsf{lex}} b$ and string $b||a$ otherwise.

2.1 Computational Assumptions

We recall several standard computational assumptions we use in our security proofs. The last assumption, sim-gapCDH is used in the security analysis of CPace, both by us and in prior works on CPace [3,28].

Definition 1 (Pseudorandomness). *Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ be an efficiently computable function. We call F* pseudorandom *if for all PPT $\mathcal{A}$ there exists a negligible function μ, such that for all $\kappa \in \mathbb{N}$ we have*

$$\mathbf{Adv}^{\mathsf{random}}_{\mathcal{A},F}(\kappa) := \left| \Pr_{k \leftarrow_R \{0,1\}^\kappa}[\mathcal{A}^{F(k,\cdot)}(1^\kappa) = 1] - \Pr_{f \leftarrow_R \mathsf{Func}_\kappa}[\mathcal{A}^{f(\cdot)}(1^\kappa) = 1] \right| \leq \mu(\kappa),$$

where Func_κ is the set of all functions of input and output length κ.

Definition 2 (Computational Diffie-Hellman (CDH)). *Let $\mathbb{G}$ be a family of cyclic groups indexed by the security parameter, each of some prime order q and with generator g. Then we say that the* Computational Diffie-Hellman assumption *holds in $\mathbb{G}$ if for every PPT adversary $\mathcal{A}$ there exists a negligible function μ, such that for all $\kappa \in \mathbb{N}$ it holds*

$$\mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{A},\mathbb{G}}(\kappa) := \Pr\left[\mathcal{A}(\mathbb{G}_\kappa, g, g^a, g^b) = g^{ab} \mid a, b \leftarrow_R \mathbb{Z}_q\right] \leq \mu(\kappa).$$

Definition 3 (Gap Computational Diffie-Hellman (gapCDH)). *Let $\mathbb{G}$ be a family of cyclic groups, each with some prime order q. Then we say that the* Gap Computational Diffie-Hellman assumption *holds in $\mathbb{G}$ if for every PPT adversary $\mathcal{A}$ there exists a negligible function μ, such that for all $\kappa \in \mathbb{N}$ it holds that*

$$\mathbf{Adv}^{\mathsf{gapCDH}}_{\mathcal{A},\mathbb{G}}(\kappa) := \Pr\left[\mathcal{A}^{\mathsf{DDH}_x(\cdot,\cdot)}(\mathbb{G}, g, g^a, g^b) = g^{ab} \;\middle|\; \begin{matrix} g \leftarrow_R \mathbb{G}_\kappa \\ a, b \leftarrow_R \mathbb{Z}_q \end{matrix}\right] \leq \mu(\kappa)\,,$$

where oracle $\mathsf{DDH}_x(Y, Z)$ returns 1 if $Z = Y^x$ and 0 otherwise, for $x = a, b$.

Definition 4 (Simultaneous Gap Computational Diffie-Hellman (sim-gapCDH)). *Let $\mathbb{G}$ be a family of cyclic groups, each with some order q. Then we say that the* Simultaneous Gap Computational Diffie-Hellman assumption *holds in $\mathbb{G}$ if for every PPT adversary $\mathcal{A}$ there exists a negligible function μ, such that for all $\kappa \in \mathbb{N}$ it holds that* $\mathbf{Adv}^{\mathsf{sim-gapCDH}}_{\mathcal{A},\mathbb{G}}(\kappa) :=$

$$\Pr\left[\mathcal{A}^{\mathsf{DDH}_x(\cdot,\cdot)}(\mathbb{G}_\kappa, g, g^r, g^{r'}, g^a) = (B, B^{a/r}, B^{a/r'}) \;\middle|\; \begin{matrix} g \leftarrow_R \mathbb{G}_\kappa \\ a, r, r' \leftarrow_R \mathbb{Z}_q \end{matrix}\right] \leq \mu(\kappa),$$

where oracle $\mathsf{DDH}_x(Y, Z)$ returns 1 if $Z = Y^{a/x}$ and 0 otherwise, for $x = r, r'$. In other words, $\mathcal{A}$ wins if it provides B, K, K' such that $\mathsf{DDH}(g^r, g^a, B, K) = \mathsf{DDH}(g^{r'}, g^a, B, K') = 1$.

3 Bare Password-Authenticated Key Exchage

3.1 Previous UC PAKE Models

We briefly recall the original UC PAKE model of [17] and its multi-session [18] and lazy-extraction [1] extensions, all captured in Fig. 1, and then we introduce our UC Bare PAKE model, shown in Fig. 2, and we explain its mechanics.

UC PAKE Notion Canetti et al. The *single-session* PAKE functionality $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ss}}}$ of [17], see Fig. 1, defines how an ideal PAKE scheme should behave if each pair of participants uses a pre-agreed globally unique session identifier *sid* (and matching party and counterparty identifiers). The NewSession interface models the first party $\mathcal{P}$ using a given session identifier *sid* to initialize PAKE on password *pw*, with counterparty identifier $\mathcal{CP}$, and role specifying whether $\mathcal{P}$ plays an initiator or a responder role.[6] All inputs except for the password can be leaked, hence the ideal-world adversary $\mathcal{A}$ learns $(sid, \mathcal{P}, \mathcal{CP}, \mathsf{role})$. Functionality $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ss}}}$ assumes that only two parties ever run the protocol using a given *sid*, and that these two parties respectively use identifiers $(\mathcal{P}, \mathcal{CP})$ and $(\mathcal{P}', \mathcal{CP}')$ s.t. $(\mathcal{P}', \mathcal{CP}') = (\mathcal{CP}, \mathcal{P})$, and any other NewSession requests are dropped.

When a session is initialized it is marked fresh, and the adversary has two options: (1) First, it can passively connect the two sessions using the matching $sid, \mathcal{P}, \mathcal{CP}$ inputs by passing the message between them, which is modeled by a session-termination command $(\mathsf{NewKey}, sid, \mathcal{P}, \cdot)$ issued when $\mathcal{P}$ is fresh: If this is the first party to terminate then $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ss}}}$ makes it output a random key K (step 3 in NewKey). If it is the second party to terminate then $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ss}}}$ makes it output the same key if the two parties used equal passwords (step 2), or an independent key if the two passwords were unequal (step 3). Alternatively, (2) the adversary can mount an active attack against either session, modeled by the TestPwd interface, which restricts each such attack to a unique password guess

[6] Field role can be set to $\perp$ if the protocol is symmetric.

Session initialization, single session [17]

On (NewSession, $sid, \mathcal{P}, \mathcal{CP}, pw,$ role) from $\mathcal{P}$, send (NewSession, $sid, \mathcal{P}, \mathcal{CP},$ role) to $\mathcal{A}$, and if this is the first NewSession query, or it is the second one and record $\langle \mathcal{CP}, \mathcal{P}, [pw'] \rangle$ exists, then record $\langle \mathcal{P}, \mathcal{CP}, pw \rangle$ marked fresh.

Session initialization, multi session [18]

On (NewSession, $sid, \mathcal{P}, ssid, \mathcal{CP}, pw,$ role) from $\mathcal{P}$ send (NewSession, $sid, \mathcal{P}, ssid,$ $\mathcal{CP},$ role) to $\mathcal{A}$. If this is the first NewSession query for this $ssid$, or it is the second one and record $\langle \mathcal{CP}, ssid, \mathcal{P}, [pw'] \rangle$ exists, record $\langle \mathcal{P}, ssid, \mathcal{CP}, pw \rangle$ marked fresh.

Active attack

On (TestPwd, $sid, \mathcal{P}$, $ssid$, pw^*) from $\mathcal{A}$, if $\exists$ record $\langle \mathcal{P}$, $ssid$, $[\mathcal{CP}, pw] \rangle$ marked fresh then:

- If $pw^* = pw$ then change the mark to compromised and reply "correct guess".
- Otherwise change the mark to interrupted and reply "wrong guess".

Lazy password extraction [1]

- On (RegisterTest, $sid, ssid, \mathcal{P}$) from $\mathcal{A}$, if $\exists$ record $\langle \mathcal{P}, ssid, \dots \rangle$ marked fresh then change the mark to interrupted and flag it Tested;
- On (LateTestPwd, $sid, ssid, \mathcal{P}, pw^*$) from $\mathcal{A}$, if $\exists$ record $\langle \mathcal{P}, ssid, [\mathcal{CP}, pw, K] \rangle$ with flag Tested then remove this flag and return K' to $\mathcal{A}$ s.t.:
  - If $pw^* = pw$ then set $K' := K$ else pick $K' \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.

Key generation

On (NewKey, $sid, \mathcal{P}$, $ssid$, K^*) from $\mathcal{A}$, if $\exists$ record $rec = \langle \mathcal{P}$, $ssid$, $[\mathcal{CP}, pw] \rangle$ not marked completed then do:

1. If the record is compromised, ~~or either $\mathcal{P}$ or $\mathcal{CP}$ is corrupted,~~ then set $K := K^*$.
2. If rec is marked fresh and $\exists$ record $\langle \mathcal{CP}$, $ssid$, $\mathcal{P}, pw, [K'] \rangle$ marked completed which was marked fresh when $\mathcal{CP}$ output $(sid,$ $ssid$, $K')$ then set $K := K'$.
3. In all other cases pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$.

Finally, append K to rec, change its mark to completed, output $(sid, ssid, K)$ to $\mathcal{P}$.

Fig. 1. The original single-session PAKE functionality $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{SS}}}$ of Canetti et al. [17], with single-use session identifiers sid, includes dashed boxes and excludes light gray parts. Crossed-out code corresponds to subsequent patches [3]. Excluding dashed boxes and including light gray parts creates a *multi-session* variant of the same functionality [18], denoted $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{MS}}}$ (or $\mathcal{F}_{\mathsf{PAKE}}$ for short), with global session identifier sid and single-use subsession identifiers $ssid$. Adding dark gray parts to $\mathcal{F}_{\mathsf{PAKE}}$ defines $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$, which allows *lazy extraction* from active attacks [1]. Adding the solid boxes defines a *leaky* variant of either functionality, which the default version excludes.

pw^*. If this guess is correct, the session's mark is changed to compromised,[7] otherwise it becomes interrupted.[8] Moreover, if $\mathcal{P}$'s session is compromised then $\mathcal{A}$'s session-termination query $(\mathsf{NewKey}, \mathcal{P}, sid, K^*)$ causes $\mathcal{P}$ to output an adversarially chosen key K^* (step 1), while if it is interrupted then $\mathcal{P}$ outputs a random key (step 3), but unlike in option (1) this key cannot be transferred to party $\mathcal{P}'$ because $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ss}}}$ ensures that two parties can output the same key only if they use the same passwords *and* both terminate as fresh (see step 2).

Multi-session and Lazy-Extraction Extensions of UC PAKE. Canetti and Rabin [18] showed that each single-session functionality can be cast as a multi-session one, where individual protocol instances are differentiated by unique *sub-session* identifiers *ssid*, and the session identifier *sid* designates global parameters e.g. a CRS, or an instance of a "globally available" functionality which all parties can rely on, e.g. a Random Oracle hash or an Ideal Cipher encryption. In Fig. 1 we show this multi-session form of the UC PAKE functionality, denoted $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ms}}}$. As shown in [18], if a protocol realizes $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ss}}}$ then it also realizes $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ms}}}$, as long as the environment ensures that tuples $(ssid, \mathcal{P}, \mathcal{CP})$ satisfy the same requirements posed above on tuples $(sid, \mathcal{P}, \mathcal{CP})$. In the remainder of this work we will treat the multi-session version of the PAKE functionality, i.e. $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{ms}}}$, as the definition of UC PAKE, denoted $\mathcal{F}_{\mathsf{PAKE}}$ for short.

Abdalla et al. [1] introduced a *lazy-extraction* extension of the UC PAKE functionality, which we show as functionality $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{le}}}$ in Fig. 1. In this extension the adversary can use interface RegisterTest to actively attack a session on a committed but hidden password guess. Such session becomes interrupted and outputs a random key K when it is terminated via the NewKey query, but the adversary can then reveal a unique password guess pw^* used in this attack via interface LateTestPwd, and $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{le}}}$ responds with the correct key K if the guess is correct, or an independent value if the guess is wrong. We include this extension because [1] showed it is necessary and sufficient to capture several round and computation-efficient PAKE protocols, including CPace [3,28], which in this work we generalize to the UC bare PAKE model.

3.2 The UC Bare PAKE Model

In Fig. 2 we describe an ideal functionality $\mathcal{F}_{\mathsf{bPAKE}}$ which defines our proposed UC *bare* PAKE (bPAKE) model. It is a multi-session functionality, where identifier *sid* identifies global common parameters, e.g. an instance of an Ideal Cipher or a Random Oracle. However, our bPAKE functionality $\mathcal{F}_{\mathsf{bPAKE}}$ makes several major departures from the multi-session PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$. First and foremost, functionality $\mathcal{F}_{\mathsf{bPAKE}}$ models PAKE as a *password-only* protocol, where the functionality does not enforce that the participants have pre-agreed unique subsession identifiers *ssid* or party identifiers $\mathcal{P}, \mathcal{CP}$. Indeed, interface NewSession in $\mathcal{F}_{\mathsf{bPAKE}}$ does not take inputs $(\mathcal{P}, ssid, \mathcal{CP})$ which were present in

[7] In [17] the PAKE functionality leaks if this case occures to the adversary, but our default notion omits that leakage since it is not present in the protocols we analyze.

[8] Either mark prevents $\mathcal{A}$ from issuing another TestPwd query for the same session.

$\mathcal{F}_{\mathsf{PAKE}}$, and when sessions are passively connected, via interface PassiveNewKey, functionality $\mathcal{F}_{\mathsf{bPAKE}}$ allows two sessions to output the same session key K only if their passwords pw are the same (see step 2 in PassiveNewKey in Fig. 2).

In addition to the password input pw, the other inputs of bPAKE (see the NewSession interface in Fig. 2) include an instance identifier i, and fields id and role. Input role plays the same function of distinguishing protocol initiators and responders as in UC PAKE. The *instance identifier* i plays no security role, and is used only to distinguish between many bPAKE instances which one party can execute, hence i's must only be locally unique. Finally, id is a *party identifier* which party $\mathcal{P}$ uses in that instance. It is an arbitrary string, might not be unique, can be set to $\perp$, and $\mathcal{P}$'s counterparty does not need prior knowledge of it. Its security property is that if the PAKE instances of $\mathcal{P}$ and $\mathcal{P}'$ are passively connected then $\mathcal{P}$ outputs $cpid$ used by $\mathcal{P}'$ as its counterparty name $cpid$, and vice versa. In particular, in bPAKE party $\mathcal{P}$ outputs an identifier of its counterparty instead of specifying that identifier in its inputs.

The bPAKE functionality supports *(sub)session identifiers ssid*, but it treats $ssid$ also as a protocol output rather than input. The session identifiers $ssid$ are public and their only semantic implication is that each $ssid$ output is globally unique except for a pair of instances which are passively connected. Functionality $\mathcal{F}_{\mathsf{bPAKE}}$ enforces this by storing previously used $ssid$'s in set S, and ensuring that any new $ssid$ satisfies $ssid \notin \mathsf{S}$.[9] Note that this implies that each session key corresponds to a globally unique public $ssid$, and that different $ssid$'s are associated with independent random session keys. In more details, let $\mathcal{P}_i$ and $\mathcal{P}'_j$ be two bPAKE instances which are initialized by $\mathcal{P}$ and $\mathcal{P}'$ on respective inputs $(i, pw, id, \mathsf{role})$ and $(j, pw', id', \mathsf{role}')$, and which are passively connected by the adversary, and let $(K, ssid, cpid)$ and $(K', ssid', cpid')$ be their outputs. Rules of $\mathcal{F}_{\mathsf{bPAKE}}$ imply that if $pw \neq pw'$ or $ssid \neq ssid'$ then K and K' are independent. Indeed, unless two random keys picked by $\mathcal{F}_{\mathsf{bPAKE}}$ collide, two passively connected instances output the same keys if and only if they use the same passwords and they are passively connected. Furthermore, if $ssid = ssid'$ then $(cpid, cpid') = (id', id)$, i.e. shared $ssid$ implies correct counterparty names. Taken together, $\mathcal{F}_{\mathsf{bPAKE}}$ rules imply that event $K = K'$ can occur if and only (except for negligible probability of collision among random keys) if either of the following two conditions hold:

1. $(pw, ssid, id, cpid) = (pw', ssid', cpid', id')$, moreover this case can occur only if $\mathcal{P}_i$ and $\mathcal{P}'_j$ are passively connected.
2. Both $\mathcal{P}_i$ and $\mathcal{P}'_j$ are actively attacked, via the ActiveNewKey interface, and both attacks are successful, i.e. the attack against $\mathcal{P}_i$ used $pw^* = pw$ and the attack against $\mathcal{P}'_j$ used $pw^* = pw'$. In this case $cpid, cpid'$ are arbitrary but $ssid, ssid'$ are globally unique, i.e. $ssid \neq ssid'$ and $ssid$ and $ssid'$ are not output as session identifiers on any other session (by any party).

[9] $\mathcal{F}_{\mathsf{bPAKE}}$ rules allow the ideal-world adversary to set $ssid$'s at will when a session terminates, but each of our protocols implements $ssid$ as a protocol transcript, and the global $ssid$ uniqueness is assured by the entropy of protocol messages.

Functionality initiation: set $\mathsf{S} := \{\}$.

Session initiation

On $(\mathsf{NewSession}, sid, i, pw, id, \mathsf{role})$ from $\mathcal{P}$, send $(\mathsf{NewSession}, sid, \mathcal{P}, i, id, \mathsf{role})$ to $\mathcal{A}$. If record $\langle \mathcal{P}, i, ... \rangle$ does not exists, record $\langle \mathcal{P}, i, pw, id \rangle$.

Key generation

On $(\mathsf{ActiveNewKey}, sid, \mathcal{P}, i, pw^*, K^*, ssid, cpid)$ from $\mathcal{A}$, if $\exists$ record $\langle \mathcal{P}, i, [pw, id] \rangle$:

- // Repeat *ssid* means repeat output.
  If $\exists$ record $\langle \mathsf{ses}_{\mathsf{act}}, \mathcal{P}, i, ssid, cpid, [K] \rangle$ output $(sid, i, K, ssid, cpid)$ to $\mathcal{P}$.
- // Actively attacked parties still get *ssid* uniqueness guarantee.
  Otherwise, if $ssid \notin \mathsf{S}$:
  - Add *ssid* to S
  - If $pw^* = pw$ then set $K := K^*$ else pick $K \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
  - If $pw^* = \perp$ then save $\langle \mathsf{latetest}, \mathcal{P}, ssid, pw, K \rangle$
  - Save $\langle \mathsf{ses}_{\mathsf{act}}, \mathcal{P}, i, ssid, cpid, K \rangle$
  - Output $(sid, i, K, ssid, cpid)$ to $\mathcal{P}$.

On $(\mathsf{PassiveNewKey}, sid, \mathcal{P}, i, \mathcal{P}', i', ssid)$ from $\mathcal{A}$, if $\exists$ record $\langle \mathcal{P}, i, [pw, id] \rangle$:

- // Repeat *ssid* means repeat output.
  If $\exists$ rec. $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}, i, [\mathcal{P}', i'], ssid, [cpid, K] \rangle$ output $(sid, i, K, ssid, cpid)$ to $\mathcal{P}$.
- // If peer was not created, request is ignored.
  Otherwise, if $\exists$ record $\langle \mathcal{P}', i', [pw', id'] \rangle$ then set $cpid := id'$ and do:
  1. // Complete protocol for the first-to-terminate participant.
     If $ssid \notin \mathsf{S}$:
     Add *ssid* to S
     Pick $K \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.
     Save $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}, i, \mathcal{P}', i', ssid, cpid, K \rangle$
     Output $(sid, i, K, ssid, cpid)$ to $\mathcal{P}$.
  2. // Complete protocol for the second-to-terminate participant.
     If $\exists$ record $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}', i', \mathcal{P}, i, ssid, id, [K'] \rangle$:
     If $pw' = pw$ then set $K := K'$ else pick $K \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.
     Save $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}, i, \perp, \perp, ssid, cpid, K \rangle$
     Output $(sid, i, K, ssid, cpid)$ to $\mathcal{P}$.
  3. // Re-use of *ssid* but with mismatched remaining data
     Otherwise, ignore $\mathsf{PassiveNewKey}$ request

Late Password Test Attack

On $(\mathsf{LateTestPwd}, sid, \mathcal{P}, i, ssid, pw^*)$ from $\mathcal{A}$, if $\exists$ record $\langle \mathsf{latetest}, \mathcal{P}, ssid, [pw, K] \rangle$ then delete this record and return K' to $\mathcal{A}$ s.t.:

- If $pw^* = pw$ then set $K' := K$ else pick $K' \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.

Fig. 2. The *bare* PAKE functionality $\mathcal{F}_{\mathsf{bPAKE}}$. Including dark grey text defines a *lazy extraction* version of this functionality, denoted $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$.

Functionality $\mathcal{F}_{\mathsf{PAKE}}$ guarantees item (1) with regards to passively connected parties, but it requires *ssid*, *id*, *cpid* to be pre-agreed, it requires *id*, *cpid* (denoted $\mathcal{P}, \mathcal{P}'$ therein) to be globally unique, and it requires the environment to create a globally unique *ssid*, while $\mathcal{F}_{\mathsf{bPAKE}}$ does not impose such requirements on inputs, and instead enforces property (1) on its outputs. Moreover, functionality $\mathcal{F}_{\mathsf{PAKE}}$ does *not* enforce guarantee (2), in particular $ssid = ssid'$ can hold in all attack cases because these are $\mathcal{F}_{\mathsf{PAKE}}$ inputs instead of outputs.

The global uniqueness of *ssid*'s implies that *ssid* output by $\mathcal{F}_{\mathsf{bPAKE}}$ is a *channel binder*, see e.g. [32], which uniquely identifies a channel defined by key K, and which can be used to authenticate it with secondary means, e.g. with a public key, using the SIGMAC method [32,37]. By contrast, the fact that $\mathcal{F}_{\mathsf{PAKE}}$ does not enforce guarantee (2) implies that $\mathcal{F}_{\mathsf{PAKE}}$'s *ssid*'s cannot play that role.

Non-leaky and "free dating" Aspects of Bare PAKE Functionality. Functionality $\mathcal{F}_{\mathsf{bPAKE}}$ handles active attacks and session outputs differently from $\mathcal{F}_{\mathsf{PAKE}}$. First, note that if functionality $\mathcal{F}_{\mathsf{PAKE}}$ is *non-leaky*, i.e. Fig. 1 omitting the fragments in solid boxes, the adversary $\mathcal{A}$ doesn't learn if $pw^* = pw$ in an active attack, therefore w.l.o.g. $\mathcal{A}$ can postpone an attack until the creation of the session output. This is why in $\mathcal{F}_{\mathsf{bPAKE}}$ we combine an active attack and session-output fixing into query ActiveNewKey that takes both a password pw^* and a key K^*. This has the same effect as sending TestPwd with pw^* followed by NewKey with K^* to $\mathcal{F}_{\mathsf{PAKE}}$. On the other hand, if a session concludes without an active attack, $\mathcal{A}$ can emulate that using query PassiveNewKey.

Another $\mathcal{F}_{\mathsf{bPAKE}}$ feature is that the pair of protocol instances which can be *matched*, i.e. which share a key if they are passively connected and use the same passwords, is not fixed before the protocol starts, and instead can be adaptively chosen in protocol execution. In $\mathcal{F}_{\mathsf{PAKE}}$ instances can match only if they use the same *ssid*, and the environment can give the same *ssid* to at most two instances. By contrast, matching in $\mathcal{F}_{\mathsf{bPAKE}}$ is based only on passwords, and a given instance can be potentially matched with any other which uses the same password. Hence query PassiveNewKey identifies the target instance $\mathcal{P}_i$ together with the counterparty instance $\mathcal{P}'_{i'}$ it connects with. If $id[\mathcal{P}_i]$ is the name used by instance $\mathcal{P}_i$ in NewSession, functionality $\mathcal{F}_{\mathsf{bPAKE}}$ reacts to $(\mathsf{PassiveNewKey}, \mathcal{P}, i, \mathcal{P}', i', ssid)$ by picking random key K, setting $cpid := id[\mathcal{P}'_{i'}]$, storing $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}, i, \mathcal{P}', i', ssid, cpid, K\rangle$ and sending $(K, ssid, cpid)$ to P_i. The adversary then has an option to also connect $\mathcal{P}'_{i'}$ to $\mathcal{P}_i$ by querying $(\mathsf{PassiveNewKey}, \mathcal{P}', i', \mathcal{P}, i, ssid)$ with the same session identifier *ssid*. (If *ssid* is a protocol transcript this happens if the last message of $\mathcal{P}_i$ is sent to $\mathcal{P}'_{i'}$.) Since *ssid* is not new, i.e. $ssid \in \mathsf{S}$, and $\exists$ record $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}, i, \mathcal{P}', i', ssid, id[\mathcal{P}'_{i'}], K\rangle$, instance $\mathcal{P}'_{i'}$ will output $(K', ssid, cpid' = id[\mathcal{P}_i])$ for $K' = K$ if $pw[\mathcal{P}_i] = pw[\mathcal{P}'_{i'}]$ and random K' otherwise.[10]

[10] See step (2) in PassiveNewKey in Fig. 2, although it can be difficult to pattern-match the above with Fig. 2 because in this second PassiveNewKey instances $\mathcal{P}'_{i'}$ and $\mathcal{P}_i$ play the opposite roles compared to the notation in that step in the figure.

Re-use of Bare PAKE Instances. If instance-matching in bPAKE is adaptive, then can it happen more than once? Indeed, if a PAKE protocol each party sends only one message,[11] party $\mathcal{P}_i$ sends its message m_i and holds a local state st_i, and when it receives message m_j from counterparty $\mathcal{P}_j$, it uses (st_i, m_j) to compute session output $(K, ssid, cpid)$. But could $\mathcal{P}_i$ not terminate at that point, continue holding state st_i, and when $\mathcal{P}_i$ receives a second message, let's say message m_t from party $\mathcal{P}_t$, then could $\mathcal{P}_i$ compute another session output $(K', ssid', cpid')$, this time on input (st_i, m_t)?

Functionality $\mathcal{F}_{\mathsf{bPAKE}}$ allows for such re-use of bPAKE session state: After bPAKE instance $\mathcal{P}_i$ is created via NewSession, and e.g. if $\mathcal{P}_i$ is an initiator a real-world $\mathcal{P}_i$ would send its first message, every time a real-world adversary forwards to $\mathcal{P}_i$ a message from some other session $\mathcal{P}'_{i'}$, the ideal-world adversary $\mathcal{A}$ sends $(\mathsf{PassiveNewKey}, \mathcal{P}, i, \mathcal{P}'_{i'}, ssid)$ with some $ssid$, and this query is processed as explained above, but this can happen unlimitted number of times for the same $\mathcal{P}_i$. Likewise, if the real-world adversary sends his own message instead of one produced by some honest instance, its ideal-world counterpart $\mathcal{A}$ will send $(\mathsf{ActiveNewKey}, \mathcal{P}, i, pw^*, K^*, ssid, cpid)$, and since some messages the real-world $\mathcal{P}_i$ can be forwarded from honest parties and some created by the adversary, the ideal-world adversary $\mathcal{A}$ can send messages of the form $(\mathsf{PassiveNewKey}, \mathcal{P}, i, ...)$ and messages of the form $(\mathsf{ActiveNewKey}, \mathcal{P}, i, ...)$ interspered in an arbitrary sequence. Each command creates a new session information on $\mathcal{P}_i$, each one identified by a unique $ssid$: Some of them can represent passively connected sessions, some actively attacked ones, but $\mathcal{F}_{\mathsf{bPAKE}}$ rules imply that there is no difference between re-using the state of a single bPAKE instance $\mathcal{P}_i$ and running multiple independent PAKE instances on the same password.[12]

Finally, if $\mathcal{P}_i$ reuses its bPAKE state to process responses from multiple counterparties, then $\mathcal{P}_i$ might process the same response twice. Since this would make $\mathcal{P}_i$'s transcript the same in these two interactions, this corresponds to $\mathcal{A}$ re-using the same $ssid$ in two key-generation queries, either PassiveNewKey or ActiveNewKey. In that case real-world $\mathcal{P}_i$ has identical session outputs in response to such queries, hence $\mathcal{F}_{\mathsf{bPAKE}}$ assures the same happens in the ideal world, by saving respectively $\langle \mathsf{ses}_{\mathsf{hbc}}, \mathcal{P}, i, ..., ssid, cpid, K \rangle$ or $\langle \mathsf{ses}_{\mathsf{act}}, \mathcal{P}, i, ssid, cpid, K \rangle$, and whenever PassiveNewKey and ActiveNewKey query is made for some $\mathcal{P}, i, ssid$, $\mathcal{F}_{\mathsf{bPAKE}}$ checks if the corresponding record exists, and if so then $\mathcal{F}_{\mathsf{bPAKE}}$ resends to $\mathcal{P}_i$ tuple $(K, ssid, cpid)$ stored in that record.

3.3 Syntax of Bare PAKE and Structured Protocols

The way we defined $\mathcal{F}_{\mathsf{bPAKE}}$ allows us to think of a unique global instance of the bare PAKE ideal functionality/protocol to which a party can resort at any time to establish a key with a counterparty that uses the same password. Indeed, a

[11] As e.g. in EKE [8], SPEKE [29,34,39], SPAKE2 [4], TBPEKE [40], and CPace [3].

[12] If party $\mathcal{P}_i$ wants to use state st_i to process only n sessions then it can process only the first n session triples output by $\mathcal{P}_i$. This is equivalent to terminating a bPAKE instance, and one can extend $\mathcal{F}_{\mathsf{bPAKE}}$ to explicitly support such feature.

party can create an arbitrary number of sessions using the input parameter i, and these will behave independently of each other. In what follows we discuss the syntax of such protocols in the real-world and clearly separate what is the code of a bare PAKE protocol from the boiler-plate infrastructure that manages communications, network addresses and other pieces of information that have no bearing on correctness and security.

A *structured protocol* [15, 2020 version on eprint, Sect. 5] consists of a protocol *shell* and a protocol *body*. We leverage this terminology to resolve the conundrum of formalizing interactive protocols where message recipients are unknown: as proposed by Canetti, we use this formalism to syntactically restrict real-world protocol access to session identifiers and party identifiers that are a UC framework artifact and should not be required by the practical protocol. The idea is to let a shell, which is a modelling component in the security analysis that abstracts the operation of a maliciously controlled network, to manage the sending of outgoing messages and the assignment of incoming ones. The body runs the "cryptographic core", i.e., it executes the bare PAKE code oblivious of any addressing information and session identification that could be provided from the outside. In this section we give only the minimum terminology that should allow the reader to follow the rest of the paper, without being overwhelmed with the details. For the full terminology and reasoning underlying structured protocols, we refer to the full version of this work [6].

The Body: Syntax of a Bare PAKE. A bare PAKE protocol is a five-tuple of ppt algorithms $(\mathsf{Setup}, \mathsf{StartIni}, \mathsf{StartRsp}, \mathsf{EndIni}, \mathsf{EndRsp})$.[13]

- $\mathsf{Setup}(1^\kappa)$ takes as input the security parameter and produces some parameters prm that are (locally) shared between multiple protocol instances.
- $\mathsf{StartIni}(prm, pw, id)$ takes as input the parameters prm, a password pw and a party name id and it outputs a first message m_1 and a state st_I.
- $\mathsf{StartRsp}(prm, pw, id)$ takes as input the parameters prm, a password pw and a party name $name$ and it outputs a state st_R.
- $\mathsf{EndRsp}(st_R, m_1)$ takes as input a state st_R and a message m_1 and, if it completes successfully, it outputs a key K, a session identifier $ssid$, a counterparty name $cpid$ and a message m_2. Otherwise it outputs $\perp$.
- $\mathsf{EndIni}(st_I, m_2)$ takes as input a state st_I and a message m_2 and, if it completes successfully, it outputs a key K, a subsession identifier $ssid$ and a counterparty name $cpname$. Otherwise it outputs $\perp$. This algorithm is deterministic.

In the case of single-simultaneous-flow (SSF) protocols, only StartIni and EndIni need to be specified, and we assume that m_1 produced by StartIni can be used as an input to EndIni.

Correctness of the protocol requires that an honest execution results in both parties agreeing on the same key K, session identifier $ssid$ and obtain the counter-

[13] This limits our analysis to two-pass protocols, which we do in this paper for the sake of simplicity. Our approach can be extended to protocols with additional rounds.

party name that was initially provided by their peer. Formally, the following should hold for all id_I, id_R, pw:

$$\Pr\left[\begin{array}{c|l} (id_I, cpid_I) = (cpid_R, id_R) & prm \leftarrow_{\mathrm{R}} \mathrm{Setup}(1^\kappa) \\ \wedge & (m_1, st_I) \leftarrow_{\mathrm{R}} \mathrm{StartIni}(prm, pw, id_I) \\ (ssid_I, K_I) = (ssid_R, K_R) & st_R \leftarrow_{\mathrm{R}} \mathrm{StartRsp}(prm, pw, id_R) \\ & (K_I, ssid_I, cpid_I, m_2) \leftarrow_{\mathrm{R}} \mathrm{EndRsp}(m_1, st_R) \\ & (ssid_R, cpid_R, K_R) \leftarrow \mathrm{EndIni}(m_2, st_I) \end{array}\right] = 1\,.$$

The Shell: Handling Instances and Communications. The above body of a bare PAKE is completed with a "wrapper" that models communication and session handling, called a *shell*. The shell resolves questions such as *"Which of Alice's passwords is used to compute a key from an incoming message?"*. Various such wrappers could be defined: one that always compute the maximum number of keys from incoming messages, or one that allows pointing at a particular password instance to compute a key with. In this work, we mostly work with the latter option, as we believe most applications will be working in this scenario. We call this shell π_{Sh}. For the sake of this overview, we illustrate its workings in Fig. 3. π_{Sh} propagates messages to the "malicious network", which in UC terminology is represented by the adversary. This captures the non-determinism of where messages are sent to, i.e., all possible scenarios of who is using a message that the shell propagates.

4 Transformations Between PAKE and Bare PAKE

From a theoretical point of view, one might ask how the two primitives relate, e.g., "Is PAKE stronger than bPAKE?" or vice versa, but one cannot say that either functionality directly implies the other in the UC-emulation sense, since their input/output behaviors are trivially distinguishable. Instead, we consider more interesting practical questions of whether we can build one protocol from the other:

- Suppose we have an implementation of a UC PAKE, but we do not know how to guarantee global session identifier uniqueness, nor are we sure what kind of party identifiers to use. Can we use it to agree on a key based only on a password? In other words, can we build a bare PAKE from a PAKE?
- Conversely, suppose we have an implementation of a bare PAKE, and we want to integrate it into a higher level protocol that expects the PAKE interface. Can we build a PAKE from a bare PAKE?

We see the first question as a formal clarification of the often raised question of how to fix session identifiers and party identifiers in practice. We see the second question as a way to formalize several choices for secure deployment that PAKE standards can offer to end-users: starting from a bare protocol taking only the password, but explaining how to cater to applications that need to fix session

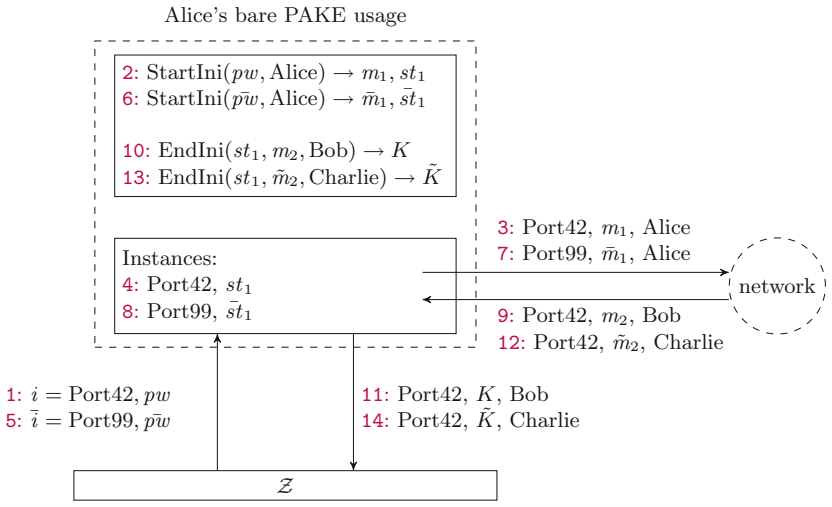

Fig. 3. Illustration of the shell π_{Sh} handling multiple passwords of Alice (omitting interface names and session- and subsession identifiers from inputs, outputs and messages for brevity, and showing a single-simultaneous flow protocol). Upon NewSession input (1:) with instance identifier Port42 and password pw, the shell (2:) calls the body to produce a message, (3:) propagates it to the network and stores the instance state (4:). The same happens for another input password $\bar{pw}$ with instance identifier Port99 (5:,6:,7:,8:). An incoming message (9:) is inspected by the shell for its instance identifier. The shell finds a corresponding instance, (10:) invokes the body with the incoming message and the instance state and (11:) outputs the key. Another incoming message (12:) for the same instance is treated the same way (13:,14:), resulting in Alice sharing keys with Bob and Charlie on instance Port42.

identifiers and/or party identifiers externally and bind them to the agreed key. We provide two compilers that transform a PAKE into a bare PAKE, and vice versa. The intuition of both compilers is given in Fig. 4. The detailed protocol description in the UC terminology as well as their formal security proofs can be found in the full version [6].

Corollary 1. *There exists a non-interactive protocol that tightly realizes* $\mathcal{F}_{\mathsf{PAKE}}$ *in the* $\mathcal{F}_{\mathsf{bPAKE}}$*-hybrid model, and there exists a 1-round protocol that tightly realizes* $\mathcal{F}_{\mathsf{bPAKE}}$ *in the* $\mathcal{F}_{\mathsf{PAKE}}$*-hybrid model.*

5 Password-Only Encrypted Key Exchange

In this section we present a black-box construction of a bare PAKE from a non-interactive key exchange (NIKE) protocol. This modular construction shows that the standard notion of security for NIKE implies, with small computational and bandwidth overhead, a reusable bare PAKE. The construction can be instantiated with a post-quantum NIKE, such as SWOOSH [25], which opens the

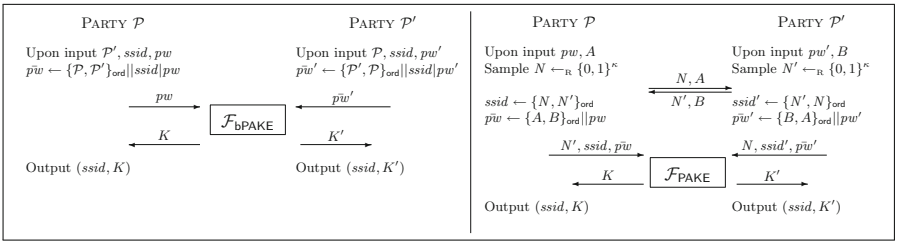

Fig. 4. Left: BarePAKE-to-PAKE compiler. Parties use the bare PAKE with their password, appended by all identifiers, without the optional name input. **Right:** PAKE-to-barePAKE compiler. Parties exchange nonces which serve both as their unique party identifier as well as subsession identifier for the PAKE. Their clear-text transmitted names are appended to the password, to ensure authenticity.

way for new PAKE constructions based on lattice-based components and possibly post-quantum-secure PAKEs.[14] Furthermore, plugging in the resulting bare PAKE protocol into the bPAKE-to-PAKE transformation allows us to recover some well known results for some Diffie-Hellman-based PAKE protocols that have appeared in the literature.

5.1 Simplified NIKE

We start from the notion of a simplified NIKE (sNIKE), as introduced in [24]. In contrast to the standard notion of NIKE, where party identities are an input to the protocol, simplified NIKE has only public keys. This notion is well suited to our goal of constructing PAKE protocols that are agnostic of party identities.

Definition 5. *An* sNIKE *scheme is a collection of three efficient algorithms* sNIKE.Setup, sNIKE.Keygen, *and* sNIKE.ShKey, *together with a shared key space* $\mathcal{SK}$.[15]

– sNIKE.Setup*: On input the security parameter, this randomized algorithm outputs system parameters prm.*

[14] We do not carry out a security analysis against quantum adversaries because it is not well understood, to the best of our knowledge, on how to deal with quantum adversaries in the ideal cipher model. Nevertheless, we note that the ideal cipher is only relevant when protecting against active attacks, which means that passive security (even with a posteriori password compromise) follows directly from the security of the underlying NIKE. This means that our protocol is suitable for applications that are concerned with preserving the confidentiality of data exchanged in the presence of passive attackers today, which may log the data and have access to a quantum computer in the future.

[15] In practice, $\mathcal{SK}$ will be the set of bit strings of fixed length ℓ for some $\ell \geq \kappa$. We consider the more general case to cover core protocols such as Diffie-Hellman, where keys are elements of a group with order at least 2^κ.

– sNIKE.Keygen*: On input prm, this randomized algorithm outputs a key pair* (sk, pk).
– sNIKE.ShKey*: On input a secret key sk and a public key pk, this algorithm outputs either a shared key in* $\mathcal{SK}$ *for the two keys, or a failure symbol* $\perp$. *This algorithm is assumed to always output* $\perp$ *if sk is the secret key corresponding to pk.*

We say an sNIKE *is* δ-correct *if*

$$\Pr\left[\begin{array}{l|l} \mathsf{sNIKE.ShKey}(sk_1, pk_2) = & prm \leftarrow\$ \mathsf{sNIKE.Setup}(1^\kappa) \\ \quad \mathsf{sNIKE.ShKey}(sk_2, pk_1) & (sk_1, pk_1) \leftarrow\$ \mathsf{sNIKE.Keygen}(prm) \\ & (sk_2, pk_2) \leftarrow\$ \mathsf{sNIKE.Keygen}(prm) \end{array}\right] \geq 1 - \delta(\kappa)\,.$$

An sNIKE *scheme is* perfectly correct *if this holds for* $\delta(\kappa) = 0$.

Security. Later in this section we will give a new generic construction of bare PAKE from sNIKE and show that its security follows from the vanilla security notion for sNIKE, which we recall here. Figure 5 shows the security game for sNIKE. The notion was originally introduced by Cash, Kiltz and Shoup [19] and hence we call it (CKS). The same figure shows, in blue, additional restrictions that define *one-time CKS* security (OT-CKS). This leads to a weaker version of sNIKE security, which we will show is sufficient to construct one-time (non-reusable) bare PAKE, and hence also PAKE. We will further show that this weaker version of one-time sNIKE security can be achieved in the random oracle model starting from a passive one-way secure sNIKE, of which a prominent case is the textbook Diffie-Hellman protocol under the CDH assumption. A corollary of these results is a modular restating of previous results showing that UC PAKE can be constructed directly from CDH in the joint ideal-cipher and random oracle model.

Definition 6. *We say that* sNIKE *is* (m, n)-CKS*, or* CKS-secure *for* m *honest keys and* n *corrupt keys if, for every ppt adversary* $\mathcal{A}$ *placing at most* m *queries to* RegHonest *and* n *queries to* RegCorrupt*, the following advantage function is negligible in the security parameter* κ.

$$\mathbf{Adv}^{(m,n)\text{-CKS}}_{\mathcal{A},\mathsf{sNIKE}}(\kappa) := |\Pr[\mathsf{CKS}^{\mathsf{sNIKE}}_{\mathcal{A}}(1^\kappa, 1) \Rightarrow 1] - \Pr[\mathsf{CKS}^{\mathsf{sNIKE}}_{\mathcal{A}}(1^\kappa, 0) \Rightarrow 1]|\,.$$

We say that sNIKE *is* (m, n)-OT-CKS*, or* one-time CKS-secure*, for* m, n *as above, if this holds when the one-time restrictions are in place.*

We also define a minimal security notion for sNIKE, namely a *one-way* (OW) counterpart of $(2, 0)$-CKS, where there are only two honest keys, there are no corrupt keys, and the adversary must explicitly compute the session key agreed between the two honest keys. We specify this security experiment in Fig. 6. Note that in this experiment a secret key is used at most once, and so the one-time restriction is redundant.

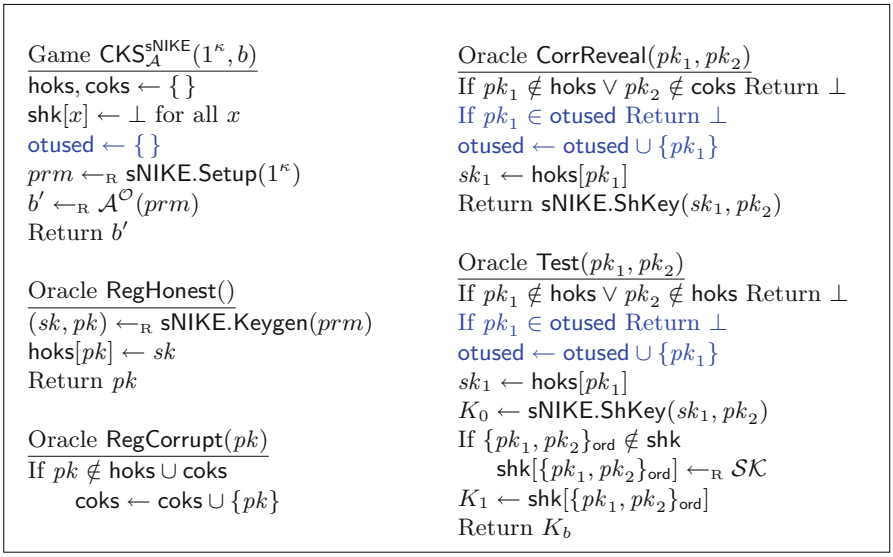

Fig. 5. CKS-style security game for an sNIKE scheme [19,24]. Oracle $\mathcal{O}$ provides adversary $\mathcal{A}$ with access to oracles $\mathsf{RegHonest}, \mathsf{RegCorrupt}(\cdot), \mathsf{CorrReveal}(\cdot,\cdot), \mathsf{Test}(\cdot,\cdot)$. Variables hoks and coks represent the sets of respectively honest and corrupt keys, and shk is a table of shared keys assigned to a pair of honest keys. Blue code enforces one-time-use restrictions on secret keys, and it corresponds to a weaker notion of CKS security which we call OT-CKS.

Definition 7. *We say that* sNIKE *is* OW*, or* one-way secure*, if for every ppt adversary $\mathcal{A}$, the following advantage function is negligible in the security parameter κ.*

$$\mathbf{Adv}^{\mathsf{OW}}_{\mathcal{A},\mathsf{sNIKE}}(\kappa) := \Pr[\mathsf{OW}^{\mathsf{sNIKE}}_{\mathcal{A}}(1^\kappa) \Rightarrow 1]\,.$$

We now show that the above one-wayness notion implies general one-time secure sNIKE in the Random Oracle Model. For any sNIKE scheme let H[sNIKE] be the same scheme except the shared key output is "post-processed" using a hash function H, i.e., $\mathsf{H}[\mathsf{sNIKE}].\mathsf{ShKey}(sk, pk) = \mathsf{H}(\mathsf{sNIKE.ShKey}(sk, pk))$. In what follows, we will call $\mathsf{sNIKE.ShKey}(sk, pk)$ a *pre-key*. The proof of the following theorem is given in the full version [6]. It is a direct reduction, where we leverage the fact that one can correctly program the Random Oracle to justify a single corrupt reveal query by guessing which point in the Random Oracle table could allow the adversary to detect an inconsistent simulation.

Theorem 1 (OW $\Rightarrow$ OT-CKS). *Let* sNIKE *be a δ-correct and* OW*-secure simplified NIKE with shared key space $\mathcal{SK}$. Then, if we model hash function $\mathsf{H} : \mathcal{SK} \rightarrow \{0,1\}^\kappa$ as a random oracle, for every attacker $\mathcal{A}$ against the (m,n)-*OT-CKS *security of* H[sNIKE]*, making at most q_H queries to* H *and placing at most q_T queries to* Test*, there exists an attacker $\mathcal{B}$ in* OW *security game*

Game $\mathsf{OW}^{\mathsf{sNIKE}}_{\mathcal{A}}(1^\kappa)$
$prm \leftarrow_{\mathrm{R}} \mathsf{sNIKE.Setup}(1^\kappa)$
$(sk_1, pk_1) \leftarrow_{\mathrm{R}} \mathsf{sNIKE.Keygen}(prm)$
$(sk_2, pk_2) \leftarrow_{\mathrm{R}} \mathsf{sNIKE.Keygen}(prm)$
$(i, j, K') \leftarrow_{\mathrm{R}} \mathcal{A}(prm, pk_1, pk_2)$
$K \leftarrow \mathsf{sNIKE.ShKey}(sk_i, pk_j)$
Return 1 if $K' = K \neq \perp$ and 0 otherwise

Fig. 6. One-wayness game for an sNIKE scheme.

of sNIKE *such that*

$$\mathbf{Adv}^{(m,n)\text{-OT-CKS}}_{\mathcal{A},\mathsf{H}[\mathsf{sNIKE}]}(\kappa) \leq q_T \cdot \delta(\kappa) + m^2 \cdot (q_H + 1)^2 \cdot \mathbf{Adv}^{\mathsf{OW}}_{\mathcal{B},\mathsf{sNIKE}}(\kappa)\,,$$

One-Time NIKE from CDH. We now consider the classical construction of a NIKE known as hashed Diffie-Hellman (HDH):

- sNIKE.Setup: On input the security parameter, outputs $prm = (\mathbb{G}, g, q, H)$, the description of a cyclic group $\mathbb{G}$ of prime order q and generator g, along with a hash function mapping $\mathbb{G}$ to $\{0,1\}^\kappa$.
- sNIKE.Keygen: On input prm, this randomized algorithm outputs a key pair (g^a, a), where a is generated as $a \leftarrow_{\mathrm{R}} \mathbb{Z}_q$.
- sNIKE.ShKey: On input a public key pk_1 and a secret key $sk_2 = a$, this algorithm outputs $H(pk_1^a)$ if $pk_1 \neq g^a$ and $\perp$ otherwise.

Corollary 2. *Hashed Diffie-Hellman is a* (m,n)-OT-CKS *secure sNIKE under the CDH assumption, when we model* H *as a random oracle. More precisely, for every one-time* sNIKE *attacker* $\mathcal{A}$ *against HDH making at most* q_H *queries to* H *and* q_T *queries to the Test oracle, there exists an algorithm* $\mathcal{B}$ *running in essentially the same time as* $\mathcal{A}$ *such that*

$$\mathbf{Adv}^{\mathsf{OT\text{-}CKS}}_{\mathcal{A},\mathsf{HDH}}(\kappa) \leq m^2 \cdot (q_H + 1)^2 \cdot \mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{B},\mathbb{G}}(\kappa).$$

Proof. The corollary follows by observing that the classic unauthenticated Diffie-Hellman protocol a perfectly correct sNIKE and its one-way security is exactly the CDH problem. □

Remark 1. We present this corollary here because, combined with the results that will follow in the rest of this section, it provides a clear and modular explanation of why one can prove that EKE-HDH is a UC-secure PAKE assuming only CDH (in the combined Ideal Cipher and Random Oracle Model). It also opens the way for constructing PAKE from one-way secure lattice-based NIKE, which may eliminate the need for costly NIZK computations and bandwidth.

Remark 2. In the particular case of HDH, it is well known that much tighter reductions can be obtained using so-called strong DH assumptions. In particular, HDH is a tightly CKS-secure sNIKE assuming Gap DH: a direct reduction

can be constructed that generates all honest key pairs by randomizing the Gap DH challenge, and answers corrupt-reveal queries using the gap oracle. Note, however, that the reduction in this case makes significantly more work than running the adversary and $O(q_H^2)$ queries to the gap oracle. This overhead in the reduction can be reduced by including the public keys in the input to the key derivation hash, in which case the number of gap oracle queries is linear in q_H.

5.2 The EKE Construction

Figure 7 shows the canonical "non-interactive" (SSF) bare PAKE using the EKE blueprint, generalized from Diffie-Hellman to any sNIKE. The protocol is defined in the $\mathcal{F}_{crs}$-hybrid model, where the common-reference-string distribution is defined by the sNIKE global parameter generation algorithm $\mathsf{sNIKE.Setup}$.[16] Throughout the discussion, for conciseness, we will keep $\mathcal{F}_{crs}$ implicit and assume that all parties have access to prm.

algorithm StartIni(prm, pw, id)**:**	**algorithm** EndIni(m', st)**:**
$(sk, pk) \leftarrow_{\mathrm{R}} \mathsf{sNIKE.Keygen}(prm)$	Parse $(sk, pw, m) \leftarrow st$
$c \leftarrow \mathsf{IC}_{pw}(pk)$	Parse $(c', id') \leftarrow m'$
$m \leftarrow (c, id)$	$pk' \leftarrow \mathsf{IC}_{pw}^{-1}(c')$
$st \leftarrow (sk, pw, m)$	$\bar{K} \leftarrow \mathsf{sNIKE.ShKey}(pk', sk)$
Return (m, st)	$ssid \leftarrow \{m, m'\}_{\mathsf{ord}}$
	$K \leftarrow \mathsf{KDF}(\bar{K}, ssid)$
	Return $(ssid, id', K)$

Fig. 7. Protocol EKE-NIKE: A bare password-encrypted key exchange (EKE) based on sNIKE and an ideal cipher IC over the domain of NIKE public keys. The protocol is single simultaneous flow (SSF). The setup algorithm run by $\mathcal{F}_{crs}$ is Setup $= \mathsf{sNIKE.Setup}$.

Theorem 2 (Security of EKE-NIKE). *Let π be the bPAKE protocol that results from combining EKE-NIKE in Fig. 7 with wrapper code π_{SH} (cf. Section 3.3). Then π UC-emulates $\mathcal{F}_{\mathsf{bPAKE}}$ under static corruptions, assuming IC is an ideal cipher on domain $\mathcal{C} = \mathcal{PK}$, sNIKE is CKS-secure, and the distribution of public keys produced by $\mathsf{sNIKE.Keygen}$ is computationally close to uniform over $\mathcal{PK}$ and has min-entropy at least κ bits.*

More precisely, the UC emulation bound is given by

$$\mathcal{D}_{\mathcal{Z}}^{\pi,\{\mathcal{F}_{\mathsf{bPAKE}},\mathsf{Sim}\}}(\kappa) \leq$$
$$\mathbf{Adv}_{\mathcal{B},\mathsf{sNIKE}}^{(q_{\mathsf{IC}},q_{\mathsf{IC}})\text{-}\mathsf{CKS}}(\kappa) + q_K \cdot \epsilon_{\mathsf{KDF}} + q_{\mathsf{IC}} \cdot \epsilon_{\mathsf{sNIKE.Keygen}} + 2 \cdot q_{\mathsf{IC}}^2 \cdot (1/|\mathcal{PK}| + 1/2^{\kappa})$$

[16] The $\mathcal{F}_{crs}$-hybrid model assumes that all parties have access to a functionality that publishes a common reference string (CRS) sampled from some distribution. In our work this CRS does not need to be programmed by the simulator, and hence this functionality can be global.

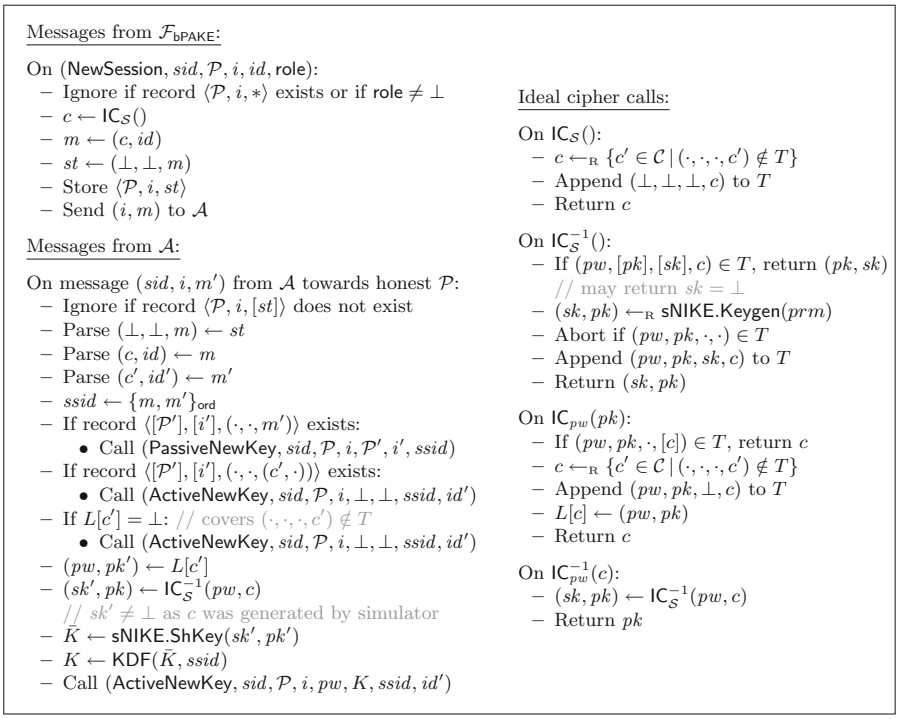

Fig. 8. Simulator for the proof of Theorem 2. The simulator runs sNIKE.Setup on start-up and provides the resulting prm to the adversary as the output of $\mathcal{F}_{crs}$. These are also used throughout the simulation.

where $\mathcal{D}_{\mathcal{Z}}^{\pi,\{\mathcal{F}_{\mathsf{bPAKE}},\mathsf{Sim}\}}(\kappa)$ *is the distinguishing advantage of environment* $\mathcal{Z}$ *between the real world execution of* π *and the simulation presented by* Sim *interacting with* $\mathcal{F}_{\mathsf{bPAKE}}$, q_{IC} *is an upper bound on the total number of IC computations,* q_K *is an upper bound on the number of session keys derived by honest parties,* ϵ_{KDF} *is the maximum distinguishing advantage of ppt adversaries against the PRF property of KDF, and* $\epsilon_{\mathsf{sNIKE.Keygen}}$ *is the maximum distinguishing advantage of ppt adversaries in distinguishing public-keys produced by* sNIKE.Keygen *from uniform values in* $\mathcal{PK}$.

Proof. (Sketch.) The full proof is given in the full version [6]. Here we give only a sketch. We present the simulator that justifies our protocol in Fig. 8. The simulator generates random IC ciphertexts as the outgoing messages of honest parties, so as not to commit to an input key-pair (it does not know under which password to encrypt it). This is undetectable to the attacker unless it guesses one of these public keys, which we exclude using the assumption that they are high-entropy. The inverse operation of the ideal cipher is simulated by generating key pairs, whenever the adversary tries to decrypt a fresh ciphertext—in particular

the ones that honest parties produced—hence the assumption that public keys look uniform. This will allow the simulator to a-posteriori recover the secret key of an honest party, when it extracts a correct password guess from the adversary (see below).

When an adversary delivers an encrypted key to an honest party, we have two options: either 1) the ciphertext was honestly generated, or 2) it was maliciously generated. If 1) occurred, then the adversary is launching a passive attack (this may be only partially passive if the adversary alters the rest of the message) and will not know the associated session key down to the security of the underlying sNIKE scheme; in this case the simulator calls the functionality on PassiveNewKey or ActiveNewKey without a password guess, depending on whether the attack is passive, or partially passive. If 2) occurred, then the simulator still needs to deal with two subcases: i) it can extract the password associated with that ciphertext; or ii) it cannot extract the password. In the first subcase the simulator constructs a plausible protocol execution for a possible correct password guess (as described above) and calls ActiveNewKey. In the second subcase we show that the adversary simply has no control of the public key that the honest party would recover from that ciphertext. In all cases where the functionality chooses a random session key, we show that the attacker cannot distinguish this from the real-world down to sNIKE security. In the cases that the password is extracted and it is correct, the simulator perfectly mimics the behavior of the honest party. □

Remark 3. Suppose EKE-sNIKE is used as a PAKE, i.e., each instance of the protocol is used to process at most one incoming message. Then, the proof above still works, but the reduction to sNIKE security is now a valid reduction to *one-time* sNIKE security. Combined with the results on HDH sNIKE given in the beginning of this section, this gives us an alternative proof that EKE-HDH is a UC-secure PAKE down to CDH. Furthermore, it also allows us to plug-in a passively secure lattice-based sNIKE such as the core of SWOOSH [25], which may open the way for post-quantum secure PAKE from lattices without relying on costly NIZK components.

We note that our proof applies when there exists domain such that 1) the distribution (of some encoding) of NIKE public keys is indistinguishable from uniform over said domain, and 2) there is an ideal cipher over domain said domain. Therefore, if the NIKE public keys can be encoded as uniform elements of some domain, the question remains how to instantiate the IC over this domain in practice. The half-ideal cipher abstraction, which is an IC domain extender for some algebraic domains, is one possible solution for instantiation but requires indifferentiable hashing to the public-key space. It is an open question whether isogeny-based NIKEs satisfy these properties.

Remark 4. These results (see also the discussion on adaptive corruptions in Sect. 7) give strong evidence that reusable bPAKE is harder to construct than standard PAKE. This does not stand in contention with the results we gave in Sect. 4 with respect to being able to construct PAKE from bare PAKE and

vice-versa. Indeed, although the construction of PAKE from bare PAKE is really showing an implication, the construction of bare PAKE from PAKE is running many independent instances of the PAKE protocol for the same password, rather than reusing components previously computed by the party for the same password.

Tightness of Theorem 2. Note that the reduction of NIKE security to bPAKE is tight. Nevertheless, the NIKE advantage may depend on q_{IC}, which may be of the order of 2^{κ}, and this maps to the number of NIKE public keys. This means that the tightness of EKE-NIKE depends crucially on the security bound for the underlying NIKE. In particular, if the NIKE bound does not depend on this number, then neither does the EKE-NIKE bound. This is the case, for example, for HDH NIKE that has a tight reduction to gap-CDH. Moreover, if EKE-NIKE is used as a non-reusable bPAKE, security requires only a *one-time* NIKE, which allows for NIKE instantiations with tighter reductions and/or weaker assumptions.

6 Password-Only CPace

In Fig. 9 we cast the CPace PAKE protocol [3,28] as a bare PAKE. The differences to "basic CPace" (referred to as "CPace" in the below) of Abdalla et al. [3] are as follows.

- Bare CPace derives the **group generator only from** *pw*, while CPace derives the generator from *pw* and both party identifiers. If multiple keys need to be exchanged, or more than two parties use the protocol, CPace also needs to additionally make the generator computation dependent on the session identifier *sid* [3].
- Bare CPace lets parties **send their own name** alongside their DH message, while in CPace parties retrieved the counterparty name from the application (i.e., as input).
- CPace computes the session key as a hash of the Diffie Hellman value and the DH public keys. Bare CPace requires to additionally **include the party names into the final key derivation hash**, to achieve the desired authentication properties of the party names (matching output keys imply that parties reliably received the name of their respective counterparty).
- Bare CPace lets a party **output the counterparty's name**, while CPace required it as input to derive the password as described above.
- Bare CPace lets parties **output a subsession (key exchange) identifier** *ssid* which is composed of the two party names and messages, and which is unique with overwhelming probability if output by an honest party. Basic CPace required such a unique session identifier as input.

We now specify the algorithms for the body of bare CPace: $\mathrm{Setup}_{\mathsf{bCPace}}$, $\mathrm{StartIni}_{\mathsf{bCPace}}$, $\mathrm{EndIni}_{\mathsf{bCPace}}$. Because CPace is a single-simultaneous flow protocol, we can omit the receiver algorithms StartRsp and EndRsp. The protocol is

defined in the $\mathcal{F}_{crs}$-hybrid model, where the common-reference-string distribution is defined by the parameter generation algorithm $\text{Setup}_{\mathsf{bCPace}}$.

- $\text{Setup}_{\mathsf{bCPace}}(1^\kappa)$ takes as input the security parameter and produces public parameters prm, containing a group $\mathbb{G}$ of order q and hash functions $H : \{0,1\}^* \to \{0,1\}^\kappa$, $\mathcal{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$.
- $\text{StartIni}_{\mathsf{bCPace}}(prm, pw, id)$ takes as input prm, a password pw and a party name id. It then computes $g \leftarrow \mathcal{H}_{\mathbb{G}}(pw)$, samples $a \leftarrow_{\text{R}} \mathbb{Z}_q$, sets $A \leftarrow g^a$ and outputs $m_1 \leftarrow (A, id)$ and $st_I \leftarrow (pw, id, g, a, A)$.
- $\text{EndIni}_{\mathsf{bCPace}}(st_I, m_2)$ takes as input a state $st_I := (pw, id, g, a, A)$ and a message $m_2 := (B, cpid)$. It then sets $ssid \leftarrow \{(A||id), (B||cpid)\}_{\mathsf{ord}}$, computes $K \leftarrow H(B^a, \{A, B\}_{\mathsf{ord}})$ and outputs $K, ssid$, and $cpid$.

We denote with Π_{bCPace} the structured protocol with the shell executing the wrapper code π_{SH} (cf. Section 3.3), calling the body's algorithms $\text{Setup}_{\mathsf{bCPace}}$, $\text{StartIni}_{\mathsf{bCPace}}$, $\text{EndIni}_{\mathsf{bCPace}}$ specified above. For clarity, we state Π_{bCPace} below, marking in gray the protocol parts that run in the body. Because CPace is single-simultaneous flow and hence does not have initiator and responder roles, we assume wlog that parties retrieve inputs with $\mathsf{role} = \perp$.

Structured protocol Π_{bCPace}

On creation, do:
- generate group $\mathbb{G}$ of order q //$\text{Setup}_{\mathsf{bCPace}}$
- pick functions $H : \{0,1\}^* \to \{0,1\}^\kappa$, $\mathcal{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ //$\text{Setup}_{\mathsf{bCPace}}$
- store $prm \leftarrow (\mathbb{G}, q, H, \mathcal{H}_{\mathbb{G}})$

On $(\mathsf{NewSession}, sid, i, pw, id, \perp)$, do:
- ignore this query if record $\langle [i], ... \rangle$ exists
- $g \leftarrow \mathcal{H}_{\mathbb{G}}(pw)$, sample $a \leftarrow_{\text{R}} \mathbb{Z}_q$, set $A \leftarrow g^a$ //$\text{StartIni}_{\mathsf{bCPace}}(prm, pw, id)$
- set $st \leftarrow (pw, id, g, a, A)$
- store $\langle i, st \rangle$
- send message $(sid, i, (A, id))$ to $\mathcal{A}$

When $\mathcal{A}$ delivers message $(sid, i, (B, cpid))$, do:
- retrieve record $\langle [i], (pw, id, g, a, A) \rangle$, ignore message otherwise
- $ssid \leftarrow \{(A||id), (B||cpid)\}_{\mathsf{ord}}$, $K \leftarrow H(B^a, ssid)$ //$\text{EndIni}_{\mathsf{bCPace}}(st, (B, cpid))$
- output $(sid, i, K, ssid, cpid)$

Fig. 9. Bare CPace as structured protocol Π_{bCPace}. Gray parts are run in body, the rest is the wrapper code $\pi_{\text{SH}}(Colorfigureonline)$ executed in the shell.

Theorem 3 (Security of bare CPace). *Protocol* Π_{bCPace} *UC-emulates the* lazy-extraction *version of the bare PAKE functionality,* $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$*, shown in Fig. 2, in the* $\mathcal{F}_{crs}$*-hybrid model, with* $\mathcal{H}_{\mathbb{G}}, H$ *modeled as random oracles, if the*

gapCDH and sim-gapCDH assumptions hold in $\mathbb{G}$*, and with respect to static party corruption.*

That is, for any efficient adversary $\mathcal{A}$ *against* Π_{bCPace} *there exists an efficient simulator* Sim *that interacts with* $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ *and produces a view such that for all efficient environments it holds that*

$$\mathcal{D}_{\mathcal{Z}}^{\Pi_{\mathsf{bCPace}},\{\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}},\mathsf{Sim}\}}(\kappa) \leq \frac{(q_{\mathsf{H2G}}+q_{\mathsf{var}})^2+q_{\mathsf{ns}}^2}{2\cdot 2^{\kappa}} + q_{\mathsf{var}}q_{\mathsf{pw}}\mathbf{Adv}_{\mathbb{G}}^{\mathsf{gapCDH}} + q_{\mathsf{H2G}}^2 q_{\mathsf{pw}}\mathbf{Adv}_{\mathbb{G}}^{\mathsf{sim-gapCDH}}$$

where $\mathcal{D}_{\mathcal{Z}}^{\Pi_{\mathsf{bCPace}},\{\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}},\mathsf{Sim}\}}(\kappa)$ *is the distinguishing advantage of environment* $\mathcal{Z}$ *between the real world execution of* Π_{bCPace} *and the simulation presented by* Sim *interacting with* $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$*, and where* q_{var} *is the overall number of passwords in the system,* q_{H2G} *is the number of* $\mathcal{H}_{\mathbb{G}}$ *queries issued by* $\mathcal{A}$*,* q_{ns} *is the number of* $\mathsf{NewSession}$ *queries issued by* $\mathcal{A}$*, and* q_{pw} *is the maximum number of parties receiving the same password through a* $\mathsf{NewSession}$ *input.*

Proof Sketch. The high level idea of the proof is as follows. Starting from the real execution where environment $\mathcal{Z}$ interacts with parties running Π_{bCPace} and a dummy adversary $\mathcal{A}$, we subsequently change the execution until we end up with $\mathcal{Z}$ interacting with functionality $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and a simulator Sim. The two main challenges are as follows: the simulator needs to produce message indistinguishable from real ones without knowing the passwords of honest parties, and we have to randomize the output keys of parties. The simulation approach is to use a common "simulation" generator g_{Sim} to generate all group elements. The simulator maintains trapdoors $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ from queries $\mathcal{H}_{\mathbb{G}}(pw) = g_{\mathsf{Sim}}^r$ for adversarial password hashes, and uses these trapdoors to identify password guesses. In a bit more detail, output keys in CPace are computed from hashing Diffie-Hellman keys, e.g., $H(K, ssid)$, where $K := \mathcal{H}_{\mathbb{G}}(pw)^{ab}$ and a and b are the secret keys of Alice and Bob sharing the key. Since Sim does not know $\mathcal{H}_{\mathbb{G}}(pw)$ when simulating Alice's message $A := g_{\mathsf{Sim}}^s$, Sim does not explicitly know Alice's secret key a. Hence Sim cannot compute $K = B^a$ from Bob's message B. However, *after* learning Alice's password pw, Sim's knowledge of trapdoor r with $\mathcal{H}_{\mathbb{G}}(pw) = g_{\mathsf{Sim}}^r$ lets Sim compute Alices "correct" secret key as s/r, since $A = g_{\mathsf{Sim}}^s = (g_{\mathsf{Sim}}^r)^{s/r} = \mathcal{H}_{\mathbb{G}}(pw)^{s/r}$.

Our proof proceeds as follows. First, we switch the simulation to the common simulation base g_{Sim} and let Sim remember trapdoors for all group elements, i.e., $\mathcal{H}_{\mathbb{G}}$ queries, and simulated messages of honest parties. Then, we randomize the output keys of parties. For honest sessions, keys are pseudorandom under the gapCDH assumption, since keys output by parties are hashed Diffie-Hellman keys (requiring the environment to solve for these DH keys if it aims at detecting the randomization). However, the reduction requires a DDH oracle (hence the security of CPace relies on gap-type assumptions) to consistently simulate other keys output by the parties. For attacked parties, the randomization of

keys is more complex, because the simulator has to extract password guesses from adversarial messages reaching the attacked party. Here, we can show that if the sim-gapCDH assumption holds in $\mathbb{G}$, the adversary cannot manufacture adversarial messages that constitute a guess for two different passwords. Sim can then extract the unique guess from the adversarial queries to $\mathcal{H}_{\mathbb{G}}$ and H. After randomizing all output keys, randomizing the protocol transcript can simply leverage the entropy of secret keys, since no other simulated values depend on the password anymore. Because $\mathbb{G}$ is cyclic, $\mathcal{H}_{\mathbb{G}}(pw)$ is a generator and the (whp) uniqueness of secret keys is enough to argue a uniform distribution of honest parties' messages.

With outputs being determined by $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and the transcript being sampled at random from the group, the simulation does not depend on the passwords anymore and the ideal execution is reached, concluding the proof. The detailed proof is in the full version [6].

Tightness of Theorem 3. The proof of Theorem 3 established the security of CPace in a *multi-session* setting, where sessions are allowed to be re-used and can output multiple keys. Compared to the standard and *non-reusable* CPace [3], we have an additional cost of a factor q_{pw} for sim-gapCDH and a factor approximately linear in the number of new sessions for gap-CDH. The latter is attributed to the 1-to-n nature of reusable PAKE (i.e., every input password can result in the output of n keys), while standard PAKE is 1-to-1. This also means that a 1-to-1 bare CPace, where a party only outputs one key per password, is almost as tight as CPace with session identifiers [3]: If everybody uses a different password (as in the PAKE built from bare CPace via the bPAKE-to-PAKE transform in Fig. 4) then $q_{\mathsf{pw}} = 1$ and the bounds of our proof and the proof of [3] would be exactly equal.

7 Security Under Adaptive Corruptions

Adaptive corruptions refer to party corruptions that occur at a point in time where that a party already executed some parts of the protocol honestly. Upon corruption, the whole internal state produced up to this point is revealed to the adversary, and from that point on, the adversary controls all actions of the corrupted party.

Adaptive corruptions are a challenging but realistic attack scenario, and in this section we argue that some of the protocols we consider in this work maintain their guarantees even when adaptive corruptions are allowed. In a nutshell, the challenge of simulating adaptive corruptions lies in manufacturing secret values of the corrupted party that "explain" both its inputs and its sent messages up to the point of corruption. Here, note that the messages were simulated *without* knowledge of the secret inputs, i.e., the passwords (Fig. 10).

7.1 Adaptive Security of EKE-NIKE

We considered the possibility of proving security of EKE-NIKE under adaptive corruptions assuming that the underlying sNIKE offers the standard notion of

On (AdaptiveCorruption, $sid, \mathcal{P}$) from $\mathcal{A}$:
- Initialize an empty array state
- For each record $\langle \mathcal{P}, [i, pw, id] \rangle$:
  - For each record $\langle \mathsf{sesinf}, \mathcal{P}, i, [ssid, id', k] \rangle$:
    - set $\mathsf{state}[ssid] \leftarrow (i, pw, id, id', k)$
- Send state to $\mathcal{A}$

Fig. 10. Interface for adaptive corruptions in $\mathcal{F}_{\mathsf{bPAKE}}$ (or variants thereof).

key corruption where the game reveals the long term secret key of an honest party, provided that no Test query has been made related to this key. However, the proof fails because in the UC setting adaptive corruption means that the internal state of a party may be revealed even if it has computed a key in the past.

For the particular case of standard PAKE discussed above, the proof can be extended for the adaptive corruption case, assuming that parties erase the secret keys after use. (This is good practice in general to achieve forward security in practice.) To see this, note that the reduction to one-time sNIKE would either be 1) simulating an honest party before corruption, in which case it can use Test and CorrReveal and it will never need to reveal the underlying secret key; 2) or simulating the honest party after corruption, in which case it already obtained the secret key from the sNIKE game.

7.2 Adaptive Security of CPace

We demonstrate that bare CPace is adaptively secure, i.e., Theorem 3 holds with respect to adaptive party corruptions. In the real world, upon adaptively corrupting a party, the shell hands the internal state consisting of all of the shell's records $\langle i, (pw, id, g, a, A) \rangle$ to the adversary, where $\mathcal{H}_{\mathbb{G}}(pw) = g$ and $A = g^a$ is the message produced upon input (NewSession, $sid, i, pw, id, \perp$) to the party while it was not yet corrupted.

The simulation of adaptive corruptions is given in Fig. 11. In a nutshell, the strategy of the simulator is to (1) adjust the secret keys of adaptively corrupted party $\mathcal{P}$ to the passwords of all previous NewSession inputs, and to (2) adjust the random oracle $H()$ to the previous output keys of $\mathcal{P}$. For (1), the simulator leverages the $\mathcal{H}_{\mathbb{G}}$ trapdoors as follows: let $\mathcal{H}_{\mathbb{G}}(pw) = R$ denote the generator that $\mathcal{P}$ should have computed upon input (NewSession, ..., pw, ...), and let $S := g^s_{\mathsf{Sim}}$ denote the simulated message of $\mathcal{P}$ upon that input. Note that, upon adaptive corruption, it is possible that Sim already handed R to $\mathcal{A}$, since $\mathcal{A}$ is allowed to query $\mathcal{H}_{\mathbb{G}}$ on arbitrary values. Sim now needs to figure out which secret key y_i "explains" the message S, i.e., for which y_i it holds that $\mathcal{H}_{\mathbb{G}}(pw)^y_i = S$. We have $\mathcal{H}_{\mathbb{G}}(pw)^{y_i} = (g^r_{\mathsf{Sim}})^{y_i} = g^s_{\mathsf{Sim}}$ for $y_i = sr^{-1}$, and hence Sim can claim sr^{-1} to be the secret key of $\mathcal{P}$ computed upon receiving pw as input. For (2), we then use secret key y_i to compute the Diffie-Hellman value $k = B^{y_i}$ computed by $\mathcal{P}$ upon receiving message B.

On $(\mathsf{AdaptiveCorruption}, sid, \mathcal{P})$ from $\mathcal{A}$:
- Send $(\mathsf{AdaptiveCorruption}, sid, \mathcal{P})$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and receive back state
- For each $ssid$ in state:
  - Parse $\mathsf{state}[ssid]$ as (i, pw, id, id', k)
  - Parse $ssid := \{(A||id), (B||id')\}_{\mathsf{ord}}$, i.e., $(sid, i, (B, id'))$ is the message that led $\mathcal{P}$ to output k
  - Retrieve record $(\mathcal{P}, i, [s, S], id, [\mathsf{keys}_i])$
  - Do once for every i:
    - If there is a record $(\mathcal{H}_{\mathbb{G}}, pw, [r, R])$ then set $y_i \leftarrow sr^{-1}$ // y_i is the secret key!
    - If there is no such record, sample $r \leftarrow_{\mathrm{R}} \mathbb{Z}_q$, set $y_i \leftarrow sr^{-1}$, set $R \leftarrow g^r_{\mathsf{Sim}}$ and store $(\mathcal{H}_{\mathbb{G}}, pw, r, R)$
  - Record $(H, B^{y_i}, ssid, k)$ // Programming H to the output key
  - Send $(\langle i, (pw, id, R, y_i, B)\rangle)_i \in \mathsf{state}$ to $\mathcal{A}$

Fig. 11. Simulation of adaptive corruptions for Π_{bCPace}.

References

1. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Jiayu, X.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 278–307. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-56784-2_10
2. Abdalla, M., Haase, B., Hesse, J.: CPace, a balanced composable PAKE. IRTF CFRG draft (2020)
3. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 711–741. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-92068-5_24
4. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30574-3_14
5. Barbosa, M., Boldyreva, A., Chen, S., Warinschi, B.: Provable security analysis of FIDO2. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 125–156. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-84252-9_5
6. Barbosa, M., Gellert, K., Hesse, J., Jarecki, S.: Bare Pake: universally composable key exchange from just passwords. Cryptology ePrint Archive, Paper 2024/234 (2024). https://eprint.iacr.org/2024/234
7. Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: Get a CAKE: generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In: Tibouchi, M., Wang, X. (eds.) ACNS 2023, Part II. LNCS, vol. 13906, pp. 516–538. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33491-7_19
8. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_11

9. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press (1992)
10. Bender, J., Fischlin, M., Kügler, D.: Security analysis of the PACE key-agreement protocol. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 33–48. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04474-8_3
11. Bindel, N., Cremers, C., Zhao, M.: FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. Cryptology ePrint Archive, Report 2022/1029 (2022). https://eprint.iacr.org/2022/1029
12. Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F.: The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In: 2012 IEEE Symposium on Security and Privacy, pp. 553–567. IEEE Computer Society Press, May 2012
13. Bradley, T., Jarecki, S., Xu, J.: Strong asymmetric PAKE based on trapdoor CKEM. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 798–825. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_26
14. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001)
15. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: IEEE Symposium on Foundations of Computer Science – FOCS 2001, pp. 136–145. IEEE (2001)
16. Canetti, R.: SIDS in UC-secure PAKE and KE. IRTF CFRG mail archive (2019)
17. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_24
18. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_16
19. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_8
20. CFRG. CFRG PAKE selection. IRTF website (2020)
21. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
22. Dupont, P.-A., Hesse, J., Pointcheval, D., Reyzin, L., Yakoubov, S.: Fuzzy password-authenticated key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 393–424. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_13
23. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 254–271. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_17
24. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. Cryptology ePrint Archive, Report 2012/732 (2012). https://eprint.iacr.org/2012/732
25. Gajland, P., de Kock, B., Quaresma, M., Malavolta, G., Schwabe, P.: Swoosh: practical lattice-based non-interactive key exchange. Cryptology ePrint Archive, Report 2023/271 (2023). https://eprint.iacr.org/2023/271

26. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_9
27. Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: asymmetric PAKE from key-hiding key exchange. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 701–730. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_24
28. Haase, B., Labrique, B.: Making password authenticated key exchange suitable for resource-constrained industrial control devices. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 346–364. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_17
29. Hao, F., Shahandashti, S.F.: The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585 (2014). https://eprint.iacr.org/2014/585
30. He, W., et al.: Rethinking access control and authentication for the home internet of things (IoT). In: 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, August 2018, pp. 255–272. USENIX Association (2018)
31. Hesse, J.: Separating symmetric and asymmetric password-authenticated key exchange. In: Galdi, C., Kolesnikov, V. (eds.) SCN 2020. LNCS, vol. 12238, pp. 579–599. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57990-6_29
32. Hesse, J., Jarecki, S., Krawczyk, H., Wood, C.: Password-authenticated TLS via OPAQUE and post-handshake authentication. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 98–127. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30589-4_4
33. Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Shin, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 485–504. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_26
34. Jablon, D.P.: Extended password key exchange protocols immune to dictionary attacks. In: 6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997), Cambridge, MA, USA, June 18–20, 1997, pp. 248–255. IEEE Computer Society (1997)
35. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 456–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_15
36. Kiefer, F., Manulis, M.: Oblivious PAKE: efficient handling of password trials. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 191–208. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23318-5_11
37. Krawczyk, H.: SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 400–425. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_24
38. Küsters, R., Tuengerthal, M.: Composition theorems without pre-established session identifiers. In: Chen, Y., Danezis, G., Shmatikov, V., (eds.) ACM CCS 2011, pp. 41–50. ACM Press (2011)
39. MacKenzie, P.: On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057 (2001). https://eprint.iacr.org/2001/057

40. Pointcheval, D., Wang, G.: VTBPEKE: verifier-based two-basis password exponential key exchange. In: Karri, R., Sinanoglu, O., Sadeghi, A.-R., Yi, X. (eds.), ASIACCS 17, pp. 301–312. ACM Press (2017)
41. Santos, B.F.D., Yanqi, G., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 128–156. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30589-4_5
42. Santos, B.F.D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 127–156. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07085-3_5
43. Shoup, V.: Security analysis of spake2+. Cryptology ePrint Archive, Paper 2020/313 (2020)
44. W3C. Web authentication working group (2017). https://www.w3.org/groups/wg/webauthn/
45. Wikpedia. Internet of things (2023). https://en.wikipedia.org/wiki/Internet_of_things/