Security analysis of CPace and (strong) AuCPace

Björn Haase

Endress+Hauser Liquid Analysis GmbH&Co. KG, Germany
bjoern.haase@endress.com

Abstract.
As a response to standardization requests regarding PAKE protocols, the IRTF working group CFRG has setup a PAKE selection process in 2019. In 2019 after the IETF 106 meeting in Singapore, a second round of the selection process has been launched with the four remaining protocol candidates SPEKE, CPace, OPAQUE and AuCPace. This paper provides an update regarding the security analysis for the candidates CPace and AuCPace.

The previous security analysis for AuCPace has been carried out for a protocol requiring explicit key confirmation. This paper presents a security analysis also of variants of the AuCPace and CPace protocols providing implicit authentication. Moreover, now the case of the pre-computation attack resistant protocol has been considered explictly.

This paper also includes the results of a repeated review of the intellectual property right situation regarding CPace, AuCPace and SPEKE2. We personally come to the conclusion that in contrast to SPEKE2, CPace and AuCPace could be used without conflicting intellectual property rights.

Finally, this paper also considers the suggestions given by Ran Canetti on the CFRG regarding alternative strategies for establishing the session ids.

Keywords: Password Authenticated Key Exchange, V-PAKE , PAKE, elliptic curves, Cryptographic Protocols, Universal Composability, IEC-62443, Industrial Control, Curve25519, X25519, OPRF

1 Introduction

Since recently, wireless and networking technology becomes integrated in products and devices that previously operated in a stand-alone setting, both in consumer applications in the so-called "Internet of Things" (IoT) and the corresponding industrial setting (IIoT). Conventional web security mechanisms are based on a PKI infrastructure that is not always available.

One use-cases where PKI could not be relied on is the so-called "IoT-onboarding" process where an initial authentication is required for establishment of PKI infrastructure data. Another use case is given by end-user applications where username and password are the only available means of authentication, e.g. in case of lacking knowledge of users for properly setting up a PKI.

In such settings, Password Authenticated Key Exchange protocols (PAKE) are known to provide a significant security advantage. Correspondingly, a PAKE selection process has been setup at the CFRG working group for which the protocols AuCPace and CPace have been nominated.

For the second round further documentation was requested. For AuCPace and CPace this information is compiled in this paper and the internet drafts for AuCPace and[Haa20c]

and CPace [Haa20b]. The draft for AuCPace also considers the concept of password-authenticated transactions which are based on a variant of AuCPace with implicit authentication.

1.1 Contributions of this paper

This paper aims at

- covering the use case of implicitly authenticated AuCPace with a security analysis.
- explicitly covering the security analysis of *strong* AuCPace.
- considering findings from the CFRG review that the assumption set for CPace needs to be modified because security is not based on the CDH problem alone but on the related SDH problem as introduced an analyzed in [PW17].
- re-writing some portions of the proof of AuCpace according to feedback from the CFRG reviews for better readability and clarity.

1.2 Organization of this paper

This paper is organized as follows.

First we give a short review of PAKE protocols and their security analysis. Here we also consider the results of our personal opinion regarding intellectual property rights.

Subsequently, we describe the design rationales and the protocol flow for both, AuCPace and strong AuCPace. We then provide the security analysis of CPace, AuCPace, partially augmented AuCPace and strong AucPace.

2 Review of PAKE protocols and their security analysis

2.1 PAKE protocols and their patent situation

Despite the clear-cut security advantages, there are a couple of reasons that hampered use of PAKE protocols in a number of applications [EKSS09]. Here patent pitfalls did play a major role. Notably, EKE [BM92], SPEKE [Jab96] and those from the PAK protocol family [BMP00] were patented. Among the patents that we found in our analysis, only the patents on PAK seem still to be relevant. The remarkable property is that these patents seem to have recently been granted an extended expiration period of 23 years in total until march 2023 in some countries. Unfortunately, we believe that the SPAKE2 [AP05] with its close similarity with PAK might be affected by the patents on PAK. Since we are not patent lawyers, we recommend more detailed analysis regarding the IPR situation for SPAKE2.

Among the algorithmic substeps for which intellectual property rights have to be considered, hashing to curve plays an important role. Specifically this holds for some variants of the so-called simplified SWU method in our opinion. Correspondingly circumvention approaches have been used, such as the so-called generic-mapping method in PACE [BFK09]. Recently research has been carried out regarding circumvention approaches for the mappings. One such approach which circumvents all patents that we are aware of has been suggested in version 05 of the [AFH19].

We have carefully re-reviewed the patent landscape regarding CPace and AuCPace. While denying any legal responsibility, we declare that we are not aware of any own or foreign intellectual property right coverage regarding the CPace and and AuCPace protocols as discussed in this paper when used in conjunction with Elligator2 or the SSWU maps from version 05 of [AFH19].

2.2 Security guarantees of PAKE and V-PAKE protocols

All types of PAKE and V-PAKE protocols base security on a low-entropy secret, the password. The direct consequence is that they could not unconditionally be proven secure because in any case exhaustive or dictionary searches by so-called "online attacks" could not be prevented. For each "online" session with a server an attacker always could test at least one password. Due to the low entropy this attack should always be assumed to succeed.

Online attacks could however be mitigated by rate-limiting countermeasures, e.g., by wait times after observing failed login attempts. This countermeasure could not be applied if the authentication protocol allows for "offline" password tests, as, e.g., in case of weak challenge-response protocols.

The security models for PAKE and V-PAKE, thus aim at proving that the "online attack" is the best available attack strategy and that at most one password per online session could be tested by the adversary.

V-PAKE goes one step further than balanced PAKE by additionally considering the case that an attacker might get access to the password database of a server by "compromising" it. This type of attack should be considered highly relevant for (I)IoT devices because often the password database is stored in unprotected memory. Unlike in a "server-room setting" an attacker might more easily gain physical access.

In the event of a server compromise it is technically infeasible to prevent offline attacks. Here V-PAKE protocols aim at forcing the adversary to additionally mount an offline search, i.e., the attacker is required to successfully mount both, server compromise and offline dictionary attacks for a successful impersonation.

As a consequence a real security improvement could actually be realized only if the V-PAKE protocol is used in conjunction with dedicated (specifically with memory-hard) password hashes that slow down an offline search.

For most augmented PAKE protocols, it is possible to interleave the substep of offline dictionary attack with the substep of server compromise, e.g., by pre-computing so-called rainbow tables. In [JKX18] Jarecki, Krawczyk and Xu have introduced the notion of "pre-computation attack resistance" of a so-called *strong* V-PAKE protocol that allows an attacker to mount the offline search only *after* having successfully compromised the server.

2.3 Security models

One important approach for analyzing a PAKE is the game-based "real-or-random" (ROR) model by Bellare, Pointcheval and Rogaway (BPR) [BPR00]. This was later extended to the so-called "find-then-guess" (FTG) model [AFP05]. Simulation-based proof techniques, were established, e.g., by Boyko, MacKenzie and Patel (BMP) in [BPR00].

In 2005 Canetti, Halevi, Katz, Lindell and MacKenzie [CHK+05] have suggested an alternative approach, based on the framework of Universal Composability (UC) [Can01], specifically in its joint-state version [CR03]. It has been shown that UC-secure PAKE constructions could not be realized without either, idealized assumptions, such as random oracles or a common reference string [CHK+05]. One of the advantages of analyzing PAKE protocols in the UC framework is that no assumptions regarding the password distribution apply. Related passwords or mistyped related passwords and forward secrecy are inherently also considered. For this reason, the UC-based approach is considered to be providing particularly strong security guarantees [PW17, JKX18].

2.4 Review of the UC framework

In this paper we assume some familiarity with the framework of Universal Composability (UC). As a short introduction, we will give a summary of the essence here. For more

details, we refer the reader to [Can00].

The general idea of UC is to define security in terms of idealized functionalities $\mathcal{F}$ which provide services to a set of players P_i. Moreover the framework considers an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ and a real-world protocol π whose security is to be analyzed. In the context of UC all of the algorithmic strategy of $\mathcal{A}$, $\mathcal{Z}$ and π are provided in form of code for an interactive Turing machine (ITM). In an actual real-world execution, a plurality of interactive Turing machine instances (ITI) is generated upon request of the environment $\mathcal{Z}$. For instance several ITI may execute the ITM algorithm π for the parties P_i. Also the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ are given their respective ITI instance.

In the UC framework this "real-world" case is compared with an "ideal-world" case where the protocol π is replaced with the ideal functionality $\mathcal{F}$ and the real-world adversary $\mathcal{A}$ with an ideal-world adversary $\mathcal{S}$. The security model is based on the observation that if any (polynomially bounded) environment algorithm $\mathcal{Z}$ cannot distinguish between the "real" and "ideal" world executions with any significant advantage, then using instances of protocol π is just as secure as using the ideal functionality $\mathcal{F}$.

From the perspective of the players P_i, $\mathcal{F}$ provides a set of subroutine calls that calculate a given function. For instance the subroutine call of the ideal functionality of a PAKE protocol $\mathcal{F}_{\text{pwKE}}$ returns a session key.

The definition of the algorithm of the ITM $\mathcal{F}$ makes sure that sensitive information is hidden from the adversary as long as no "corruption" of parties occurs. Thus, the security targets are inherently guaranteed. With corruptions, we model the case that the adversary gets control over some of the protocol partners and, e.g., is able to retrieve data from that partner's internal memory. In the literature different types of corruption could be distinguished. In so-called "static corruptions" the adversary may gain control only just before starting with an actual protocol execution. In the so-called adaptive or Byzantine corruption models we give the attacker more power by allowing him to corrupt parties at any time during protocol execution. In this paper we consider the stronger Byzantine corruption.

The original UC theorem from [Can01] allows to analyze the security of a system viewed as a single unit, but it does not guarantee anything if different protocols share some amount of state and randomness, such as, e.g., a hash function functionality. For this reason for our application, the UC theorem cannot be used as-is. Our analysis, just as the strongly related work in [ACCP08] is thus implemented in the framework of universal composition with joint state [CR03].

2.5 Overall proof strategy used in this paper

The previous security analysis for AuCPace in [HL18b] did aim at re-using existing ideal functionality formulations, specifically the functionality $\mathcal{F}_{\text{apwKE}}$ from [GMR06]. However in the course of the review at the CFRG PAKE selection process several problems in the original formulation of $\mathcal{F}_{\text{apwKE}}$ have been identified. Correspondingly the proofs for OPAQUE and AuCPace have had to be considerably re-worked.

In this paper we consider these findings and introduce new ideal functionalities $\mathcal{F}_{\text{ipwKE}}$ and $\mathcal{F}_{\text{aipwKE}}$ that capture the security notion as provided by an implicitly authenticated protocol.

For the proof, we first consider the balanced CPace subprotocol. We then show by using the UC composition theorem that the combination of the sub-protocols securely implements and thus provides conventional "full" augmentation.

3 The AuCPace protocol

3.1 Design rationales for the AuCPace protocol

Our dominating guideline for the protocol development was server-side efficiency. In a typical use case of a remote human-machine interface (HMI) for a resource-constrained target, the HMI client-side has much more powerful computing capabilities.

We assessed power consumption to be one major issue in line with the results of [HL17]. Note that this goes beyond just minimizing the number of exponentiations. Also the choice of a suitable curve, and the need for point compression and point verification sub-steps should be considered. We observed, e.g., that x-coordinate-only Diffie-Hellman implementations could provide significant advantages, regarding all of speed, ease-of-implementation and memory-consumption. We therefore searched for constructions not requiring full group operations, because this could inherently avoid the need for point compression and verification. Specifically on curves with twist security this could allow for more efficient methods against small-subgroup[LL97] attacks.

We also tried to minimize the number of required primitives in the protocols. E.g. we aimed at avoiding the digital signature scheme needed for some constructions, such as the Ω-Method from Gentry, MacKenzie and Ramzan [GMR06].

3.2 Parameters of the AuCPace protocol

The AuCPace protocol is depicted in figures 1 and 2. The protocol is parametrized by

- A password based key derivation function PBKDF_σ that is itself parametrized by algorithm settings σ, specifying e.g. the memory consumption for the password hash or an iteration count. PBKDF_σ calculates a string from the password pw, a username and a so-called "salt" value. For our reference implementation of AuCPace, we use the memory-hard scrypt [PJ12] password hash.

- A (hyper-)elliptic curve $\mathcal{C}$ with a group $\mathcal{J}$ with co-factor $c_\mathcal{J}$ and a Diffie-Hellman (DH) protocol operating on both, $\mathcal{C}$ and its quadratic twist $\mathcal{C}'$. We denote the DH base point in $\mathcal{J}$ with B. We don't require full group structure in $\mathcal{J}$ but could also instantiate AuCPace if only group operations modulo negation are available. This can result in more efficient implementations. (For more details regarding this efficiency aspect see the corresponding discussion for the qDSA/EdDSA signature schemes in [RS17].). For our reference implementation, we use Curve25519 [Ber06] and the x-coordinate-only Diffie-Hellman protocol X25519. In this paper we follow the recommendation in [Ber14] and reserve the name Curve25519 for the curve and X25519 for the protocol. For the DH protocol we mostly use a simple exponentiation notation, even if additional co-factor handling and clamping might apply for the scalars for guaranteeing that the result of any exponentiation is always in $\mathcal{J}$. (The exception from this rule is that we decided explicitly detail the co-factor $c_\mathcal{J}$ complexity in figures 1 and 2 because this handling is actually crucial for security.)

- An encoding that represents either a point Y on $\mathcal{J}$ or on the quadratic twist in a fixed-size bit stream. In our reference implementation we make use of an encoding of the x-coordinate of the point on Curve25519.

- A verification algorithm that checks whether the order of an encoded element Y within $\mathcal{C}$ or $\mathcal{C}'$ is large enough for the security target specified by the complexity of the computational Diffie-Hellman problem (CDH) for security parameter k. In our reference implementation we make use of Curve25519's twist security and the integrated co-factor of 8 for X25519 scalars, i.e., we just verify that $\mathrm{X25519}(x, Y) \neq 0$.

- A Map2Point operation and its inverse map $\mathsf{Map2Point}^{-1}$. $\mathsf{Map2Point}(s)$ is required to map a string s to a point from a cryptographically large subgroup $\mathcal{J}_m$ of $\mathcal{C}$, such that the discrete logarithm of the point is unknown. The inverse map $s = \mathsf{Map2Point}^{-1}(X, l)$ is required to map a point $X \in \mathcal{J}_m$ to a bit string s of length l bits such that for any randomly sampled $X \in \mathcal{J}_m$ the string s is indistinguishable from a random bit string of length l. For our reference implementation we use Elligator2 introduced by Bernstein, Hamburg, Krasnova and Lange in [BHKL13] on Curve25519, where the sign of the inverse map result is chosen at random.

- Hash functions $\mathsf{H}_0 \ldots \mathsf{H}_4$. For our reference implementation we use SHA512 where the hash function index is prepended as domain separation identifier. (We chose SHA512 for having a high security margin. Imperfections of the hash function for proofs using the random-oracle model, should be considered to be particularly critical.)

We will refer to our reference implementation of AuCPace using the actual choices above as AuCPace25519. While denying any legal responsibility, the authors declare that they are not aware of any intellectual property right or patent limiting the use of AuCPace25519.

3.3 Configuring the password verifier on the server

Two basic sub-protocols could be distinguished. In a first sub-protocol the server is given a password-verifier $W = B^w$ for storage in its database. This protocol is depicted in figure 1. The second sub-protocol, shown in figure 2, uses the available password verifier for establishing a session key.

The configuration of password verifiers requires one message. We assume, that the specific group $\mathcal{J}$ and the permissible set of PBKDF parametrizations σ of the server are known to the client. The client chooses a fresh "salt" value and hashes the password pw to yield a secret scalar w. Then a password verifier $W = B^w$ is calculated and sent to the server for storage in the database. Optionally user authorization data (uad) is also transmitted. The server then checks whether the parametrization σ and the authorization setting to attribute to the user are acceptable and stores the verifier W in the database together with the salt and the authorization settings.

Note that when using ephemeral "salts" it could not completely be hidden whether or not a user entry is available in the server's database, specifically we could not hide the meta information that a user has changed his password.

In line with other V-PAKE papers ([PW17, CHK+05, JKX18]) for the scope of this paper we concentrate on the security proof for the session establishment only, i.e., for this first sub-protocol we assume that communication is using a confidential channel. We also assume that the client is properly authenticated and has sufficient privileges. More formally, we do not consider adversaries $\mathcal{A}$ that read or modify messages of this sub-protocol.

It is important to note that the password is *never* passed in clear-text to the server. This also implies that the computationally complex PBKDF function is calculated on the client. Note also that by letting the client choose the "salt" value, we provide a path for distributing password verifiers from a centralized user credential server by use of an offline ticket mechanism, i.e., using a single unidirectional message.

With respect to the generation of the password verifier W our protocol shares some similarities with AugPake [SKI10] and VTBPEKE [PW17] which also use a group element $W = B^\xi$ as verifier where a password-derived key ξ is used as a secret exponent.

Store password operation for AuCPace

Server		**Client**
		salt $\leftarrow_{\$} \{0,1\}^l$
		$w = \mathsf{PBKDF}_\sigma(pw,$ username, salt)
		$W = B^{w\ c_{\mathcal{J}}}$
	username, salt, uad, W, σ $\longleftarrow$	
Upon successful verification, record (username, salt, uad, W, σ) in the password file.		

Figure 1: Protocol for password configuration for AuCPace. The optional data element uad represents application data associated with this specific user account, e.g. specifying the granted authorization level on the server.

3.4 Establishing session keys based on the password pw and the password verifier W

Establishment of the session key ISK is realized by a sequence of several steps shown in figure 2. First a subsession id *ssid* is established as required by the UC framework prior to entering the protocol.

Here we follow the suggestion of Ran Canetti et on the CFRG list [Can19] of letting the initiating protocol party choose the session id value. This way the transmission of the session id as generated by the client could be merged with the inital client message, reducing the total message count.

An alternative option would have been to obtain *ssid* just by concatenating the two random nonces s and t generated by both parties [BLR04] or the method provided by [HL18b].

Secondly, a password related string PRS is calculated. We refer to this sub-protocol as the AuCPace augmentation layer. Establishing PRS involves one message round. After learning the user name, the server fetches the "salt" value from the database. In case that no entry is available for the user name, we suggest to hash a server-specific secret constant together with the username in order to yield a dummy salt value.

The server then transmits its "salt" string together with the value σ required for parameterizing the password hash PBKDF_σ and an ephemeral public key $P = B^{p\ c_{\mathcal{J}}}$ and the group representation $\mathcal{J}$ to use for the Diffie-Hellman protocol. Both, server and client entity then calculate a password-derived string PRS. While the server uses the password verifier W from its database, the client has to calculate PBKDF_σ. We neither expect to have permanent storage available on the client, nor do we recommend any permanent caching of "w". If no entry is available for W in the server's password file, or if the point verification on the client fails, the protocol is continued with a randomly sampled PRS string instead of aborting. Our approach somewhat mitigates the fact that the openly communicated "salt" value leaks some information on the server's password file contents. (At the same time we have to accept more workload when facing some types of denial-of-service attacks.)

Then client and server enter the balanced sub-protocol CPace with the password-derived string PRS as password. There, first an ephemeral generator G is calculated by use of the Map2Point algorithm.

Calculation of G involves a "channel identifier" CI which is hashed together with the PRS. In the context of TCP/IP, the CI might be constructed by concatenating unique

AuCPace

Server		**Client**
	Agree on $ssid$	
		$ssid \leftarrow_{\$} \{0,1\}^{k_1}$
	$ssid \longleftarrow$	
	AuCPace Augmentation layer	
$x \leftarrow_{\$} \{1 \dots m_{\mathcal{J}}\}$		
$X = B^{x\ c_{\mathcal{J}}}$		
	username $\longleftarrow$	
W,σ,salt = lookup$W$(user)		
	X, salt, $\sigma \longrightarrow$	
		$w = \mathsf{PBKDF}_{\sigma}(pw,$ user, salt)
if lookup failed $PRS \leftarrow_{\$} \{0,1\}^{k_2}$,		abort if X invalid
else $PRS = W^{x\ c_{\mathcal{J}}}$		$PRS = X^{w\ c_{\mathcal{J}}}$
	CPace substep	
$g' = \mathsf{H}_1(PRS\|\|ssid\|\|CI)$		$g' = \mathsf{H}_1(PRS\|\|ssid\|\|CI)$
$G = \mathsf{Map2Point}(g')$		$G = \mathsf{Map2Point}(g')$
$y_a \leftarrow_{\$} \{1 \dots m_{\mathcal{J}}\}$		$y_b \leftarrow_{\$} \{1 \dots m_{\mathcal{J}}\}$
$Y_a = G^{y_a\ c_{\mathcal{J}}}$		$Y_b = G^{y_b\ c_{\mathcal{J}}}$
	$Y_a \longrightarrow$	
	$Y_b \longleftarrow$	
$K = Y_b^{y_a\ c_{\mathcal{J}}}$		$K = Y_a^{y_b\ c_{\mathcal{J}}}$
abort if Y_b invalid		abort if Y_a invalid
$ISK = \mathsf{H}_2(ssid\|\|K\|\|Y_a\|\|Y_b)$		$ISK = \mathsf{H}_2(ssid\|\|K\|\|Y_a\|\|Y_b)$

Figure 2: Protocol AuCPace without explicit key confirmation.

representations of the server's and client's IP address and TCP port numbers. Hashing CI into G allows us to fend off certain types of relay attacks.

Incorporating the *ssid* into the calculation guarantees the generator G to be ephemeral also in the partially augmented setting from section 7.1. Note that prefixing the session id is also suggested in section 10.3 of [KTR13] for a security analysis in the IITM model [KTR13] in case that one global random oracle is to be used for an unlimited numbers of subsessions.

After determining G the two parties implement a Diffie-Hellman protocol by exchanging group elements Y_a and Y_b and deriving a shared secret point K. Note that it is mandatory for the receiving party to verify the points Y_a and Y_b. Then an intermediate session key ISK is derived from K. Note that unlike the suggestion from [HL18b] we incorporate the full transcript of the CPace substep into the calculation of the intermediate session key ISK.

As last sub-protocol, optionally explicit authentication is added by exchange of two authenticator messages (T_a and T_b in [HL18b]). In this paper we present a security analysis for a protocol variant without this explicit authentication.

With respect to the mandatory point verification, we do not impose the conventional requirement that the implementation has to verify that the points X, Y_a and Y_b are $\in \mathcal{J}$. Instead we impose a less strict requirement that could be implemented more efficiently on some curves, notably if they have secure twists: We require the verification that the order of the respective points is large with respect the required complexity assumption for the SDH and CDH problems, such as to prevent collisions in the resulting points K and X^w due to small subgroup attacks.

3.5 Key difference between AuCPace and previously known SPEKE-based constructions

The main difference between SPEKE and the balanced CPace subprotocol shown here lies in the fact that CPace works with an ephemeral generator that depends on the session id and the channel identifier CI. This allows for both, a natural way for proving security in the UC framework and a memory-optimized approach for implicitly authenticating data such as the CI.

The feature of the ephemeral generator is shared between CPace and PACE. Unlike PACE, CPace does not need an additional block cipher.

Most importantly, AuCPace allows for two different levels of augmentation. If an ephemeral key pair (x, X) is used, we obtain the conventional security guarantees of an augmented PAKE protocol. Additionally, AuCPace allows for a "partially augmented" mode where the key pair (x, X) is of long-term type and generated once during password registration. The security guarantee of partial PAKE augmentation is discussed in [HL18b].

4 The *strong* AuCPace protocol (with pre-computation attack resistance)

In [JKX18] the notion of pre-computation attack resistant or so-called "strong" verifier-based PAKE protocols has been introduced and formally defined by giving an ideal functionality $\mathcal{F}_{\text{SaPAKE}}$ for the UC framework. In the same paper, Jarecki, Krawczyk and Xu also presented a generic compiler from any conventional UC-secure augmented PAKE protocol to a protocol providing pre-computation resistance.

Since AuCPace is proven secure in the UC framework, this "compiler" from [JKX18] could be employed for turning AuCPace into a pre-computation attack resistant "strong" PAKE protocol. For this approach, the salt value to use for the password hash is no longer openly communicated but calculated by the OPRF functionality $\mathcal{F}_{\text{OPRF}}$.

Store password operation for strong AuCPace

Server		**Client**
		$\mathrm{q} \leftarrow_{\$} \{0,1\}^l$
		$Z = \mathsf{Map2Point}(\mathsf{H}_1(\text{username} \mid\mid pw))$
		$\text{salt} = Z^{(\mathrm{q}\ c_{\mathcal{J}})}$
		$w = \mathsf{PBKDF}_\sigma(pw, \text{username}, \text{salt})$
		$W = B^{w\ c_{\mathcal{J}}}$
	username, q, uad, W, σ $\longleftarrow$	

Upon successful verification, record
(username, q, uad, W, σ)
in the password file.

Figure 3: Modified protocol for password registering for strong AuCPace.

Unfortunately, and similarly to the considerations for the security proof of OPAQUE [JKX18] the generic "compiler" approach of [JKX18] would formally require a strict sequencing of the OPRF and the V-PAKE substeps. This would result in additional communication rounds.

When inserting the data exchange of the OPRF into the AuCPace messages, the composition theorems don't guarantee security as-is. Doing so requires further detail analysis.

Turning AuCPace into the pre-computation attack resistant "strong AuCPace" will essentially come at the cost of one additional exponentiation for the server and two for the client devices for each connection establishment.

4.1 Differences between AuCPace and "strong AuCPace" protocols

The essential component for providing pre-computation resistance consists in keeping the salt value secret.

In order to do so, only minor adaptions are required, firstly for the password registration process and secondly for the AuCPace augmentation layer. The required modifications for password verification are shown in figure 3. The main difference from figure 1 is that the server no longer stores the salt value but only a secret exponent q.

Figure 4 displays the corresponding modifications for the session establishment for the augmentation layer sub-protocol from figure 2. The password file lookup on the server for a given user name returns the secret exponent q and the password verifier W. In case that the user database lookup fails (user unknown to the server), we suggest to continue with a random point W and hash the user name together with a server-specific secret SSS in order to obtain a username-specific secret exponent $q = \mathsf{H}(SSS||\ \text{username})$. On the client side the point Z is blinded with a random value r. (The inversion operation $1/(r(c_{\mathcal{J}})^2)$ for calculating the exponent used for un-blinding the point UQ is calculated modulo the order of the prime-order subgroup of the curve.)

We would like to explicitly point the reader to the fact that for the proof, it was mandatory to explicitly consider the co-factors $c_{\mathcal{J}}$. It is also mandatory for the client and server to verify the points. (However the simplified verification strategies from section 3.4 also apply for the salt blinding substep in strong AuCPace.)

The additional computations correspond exactly to the required operations for the Diffie-Hellman OPRF function from [JKX18] when using the "compiler" approach.

StrongAuCPace augmentation layer

Server		**Client**
$x \leftarrow_\$ \{1 \dots m_{\mathcal{J}}\}$		$r \leftarrow_\$ \{1 \dots m_{\mathcal{J}}\}$
$X = B^{x\ c_{\mathcal{J}}}$		$Z = \mathsf{Map2Point}(\mathsf{H}_1(\text{username} \mid\mid pw)); U = Z^{r\ c_{\mathcal{J}}}$
	username, U $\longleftarrow$	
W, σ,q = lookup$W$(user)		
$UQ = U^{q\ c_{\mathcal{J}}}$		
abort if UQ invalid		
	X, UQ, σ $\longrightarrow$	
		salt $= UQ^{\frac{1}{r\ (c_{\mathcal{J}})^2}\ c_{\mathcal{J}}}$
		abort if salt invalid
		$w = \mathsf{PBKDF}_\sigma(pw, \text{user}, \text{salt})$
if lookup failed $PRS \leftarrow_\$ \{0,1\}^{k_2}$,		abort if X invalid
else $PRS = W^{x\ c_{\mathcal{J}}}$		$PRS = X^{w\ c_{\mathcal{J}}}$

Figure 4: Strong AuCPace augmentation layer. In comparison to figure 2 in addition to the user name, the blinded point U is transmitted to the server. The server does reply with the point UQ instead of the clear-text salt value.

5 Proof of indistinguishability for the balanced sub-protocol CPace

In this section we will deal with the balanced PAKE protocol CPace corresponding to the middle part of figure 2. We formalize this sub-protocol in figure 5.

In this proof we show that the protocol in figure 5 emulates the relaxed $\mathcal{F}_{\text{ipwKE}}$ functionality as introduced in figure 6) $\mathcal{F}_{\text{ipwKE}}$ receives the passwords pw from the environment-controlled parties P_i and P_j and returns upon a NewKey request the same random session key sk if and only if (iff) the passwords match.

For the security analysis in this section we need to map the inputs and outputs of the protocol from figure 2 to the notation used for defining the ideal functionality $\mathcal{F}_{\text{pwKE}}$ in figure 6. The password related components (W^x and X^w respectively) correspond to the passwords pw and the resulting session key sk from figure 6 corresponds to $sk1$ in figure 2. We define the channel identifier CI to be the concatenation of P_i and P_j.

5.1 Proof strategy

Our proof closely follows the strategy from [ACCP08]. We also use a sequence of games G_0 to G_4 in which the simulator algorithms $\mathcal{S}_0$ to $\mathcal{S}_4$ are executed. We organize these algorithms $\mathcal{S}_n$ as a combination of independent ITI that only interact through their well-defined APIs and have internal state (tape) that is not accessible from the outside. Initially we have one such ITI for each simulated honest party P_i, the hash functionality $\mathcal{F}_{\text{RO}}$ and one ITI executing the algorithm of the real-world adversary $\mathcal{A}$. We let $\mathcal{S}_n$ invoke the other ITI during the course of the execution, i.e., we treat the real-world adversary algorithm $\mathcal{A}$ as a black box subroutine library for $\mathcal{S}_n$.

The protocol CPace is parametrized by a security parameter k. CPace implements the hashes with output size $2k$ H_1 and H_2 by the functionality $\mathcal{F}_{\mathrm{RO}}$. The protocol also uses a Diffie-Hellman key exchange protocol (written in exponentiation notation X^y) operating on points X on a group modulo negation $\mathcal{J}$ of order $m_{\mathcal{J}}$ and its twist. It also uses a $\mathsf{Map2Point}$ primitive mapping a string of length $2k$ on an element Y where Y^r generates $\mathcal{J}$. It finally uses a $\mathsf{Verify}(Y)$ algorithm checking for the subgroup order of a given Y on either $\mathcal{J}$ or the twist.

Protocol:

1. When P_j receives input $(\mathsf{NewSession}, sid, P_i, P_j, pw, \text{responder role})$, he calculates $g' = \mathsf{H}_1(sid||P_i||P_j||pw)$ and $G = \mathsf{Map2Point}(g')$. He then samples a fresh nonzero scalar y_b with $0 < y_b < m_{\mathcal{J}}$ and calculates $Y_b = G^{y_b}$. He sets up a session record $(sid, P_i, y_b, Y_b, \text{responder role})$ and marks it as fresh. He then waits for a $(\mathsf{CPace\ initiator}, sid, P_i, Y_a)$ message.

2. When P_i receives input $(\mathsf{NewSession}, sid, P_i, P_j, pw, \text{initiator role})$, he calculates $g' = \mathsf{H}_1(sid||P_i||P_j||pw)$ and $G = \mathsf{Map2Point}(g')$. He then samples a fresh nonzero scalar y_a with $0 < y_a < m_{\mathcal{J}}$ and calculates $Y_a = G^{y_a}$. He sets up a session record $(sid, P_j, y_a, \text{initiator role})$ and marks it as fresh. He then sends $(\mathsf{CPace\ initiator}, sid, P_i, Y_a)$ to P_j.

3. When P_j receives input $(\mathsf{CPace\ initiator}, sid, P_i, Y_a)$ and finds a fresh session record $(sid, P_i, y_b, Y_b, \text{responder role})$ it sends message $(\mathsf{CPace\ responder}, sid, P_j, Y_b)$ message to P_i. He then calculates $\mathsf{Verify}(Y_a)$. If the point order is sufficient for security parameter $2k$ it calculates $K = Y_a^{y_b}$ and $ISK = \mathsf{H}_2(sid||K||Y_a||Y_b)$ and outputs (sid, ISK). It aborts otherwise. In either case it marks the session record as $\mathsf{completed}$.

4. When P_i receives input $(\mathsf{CPace\ responder}, sid, P_j, Y_b)$ and finds a fresh session record $(sid, P_j, y_a, \text{initiator role})$ he calculates $\mathsf{Verify}(Y_b)$. If the point order is sufficient for security parameter $2k$ it calculates $K = Y_b^{y_a}$ and $ISK = \mathsf{H}_2(sid||K||Y_a||Y_b)$ and outputs (sid, ISK). It aborts otherwise. In either case it marks the session record as $\mathsf{completed}$.

Figure 5: CPace Protocol definition for the proof of indistinguishability.

When $\mathcal{A}$ decides to corrupt a party P_i, we need to provide it all of the corrupted ITI's internal accessible state. The subsequent behavior of this party is then controlled by $\mathcal{A}$. The adversary is also given the secret scalars used in the real-world protocol. Finally, we give $\mathcal{S}_n$ also access to an ITM $\mathcal{F}_n$ in each game, where $\mathcal{F}_0$ is initially not providing any service. In each game we extend the functionality of $\mathcal{F}_n$ until it implements $\mathcal{F}_{\text{ipwKE}}$.

In the games in our proof we re-factor the algorithms $\mathcal{S}_n$ such that each change is indistinguishable for the environment $\mathcal{Z}$.

At the end of the game sequence, we end up with an algorithm $\mathcal{S}_4$ that makes only calls to $\mathcal{F}_4$ which itself implements exactly the ideal functionality.

5.2 Discussion of the implicitly authenticated $\mathcal{F}_{\text{ipwKE}}$ functionality

One of the main findings in the course of the CFRG PAKE selection process for protocols that came with a UC proof was, that significant adaptions to the previously published ideal functionalities were necessary in order to properly capture the security properties of the protocols.

This paper is the second major revision in comparison to the journal paper. Already previous first major revision did re-work the detailed re-formulation of the functionalities by introducing the concept of a relaxed ideal functionality $\mathcal{F}_{\text{pwKE}}$.

This paper additionally incorporates recent findings from Julia Hesse [Hes19] and the revised version of the OPAQUE paper from Stanislav Jarecki, Hugo Krawczyk and Jiayu Xu [JKX19].

$\mathcal{F}_{\text{ipwKE}}$ as used here (figure 6) is derived from $\mathcal{F}_{\text{pwKE}}$ as defined in [CHK+05] and adds two main features. Firstly, it turned out that for implicitly authenticated protocol, variants it is infeasible to prevent "late" password tests.

For the same reasons, the journal version of the OPAQUE paper [JKX18] did introduce the concept of "relaxed" functionalities which we did adopt in the previous revision of this paper. In the latest revision of the OPAQUE paper those relaxed functionalities were replaced by an approach that uses a specific "interrupt" query in order to model that the adversary is actively entering an interaction with a honest party, e.g. by modifying or supplying chosen messages.

We believe that the current approach from the eprint version of the OPAQUE paper [JKX19] is significantly superior to the previous "relaxed" functionality from the eurocrypt 2018 paper [JKX18] revision. The advantage is, that it explicitly models that in order to mount an online password test, the adversary has to actively interact with the honest party. We correspondingly have modified the definition of our $\mathcal{F}_{\text{ipwKE}}$ functionality.

The resulting extensions to the [CHK+05] are highlighted with a grey shaded background.

5.3 The simultaneous Diffie-Hellmann (SDH) problem

The task for a security proof of a PAKE protocol essentially consists of showing, that in active attacks, at most one *single* password test is feasible, i.e. not two or more. An adversary, thus has the capability to calculate a valid session key for one first password, but shall not have the capability to calculate a second session key that corresponds to a second password.

The specific feature that extends the adversary's capability in comparison to the conventional Diffie-Hellmann problems is that an adversary is given the power to introduce one chosen group element, e.g. Y_b, into the protocol run.

In order to formalize this capability for SPEKE-like protocols such as CPace and TBPEKE [PW17], it turned out to be necessary to define a new problem class that was coined simultaneous Diffie-Hellmann problem (SDH) in [PW17]. We repeat the definition of the SDH problem from [PW17] in our notation as follows.

The functionality $\mathcal{F}_{\text{ipwKE}}$ is parametrized by a security parameter k. It interacts with an adversary $\mathcal{S}$ and a set of parties via the following queries:

Upon receiving a query (NewSession,sid, P_i, P_j, pw,role) **from party** P_i **:**
Send (NewSession,sid, P_i, P_j) to $\mathcal{S}$. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record (P_j, P_i, pw'), then record (P_i, P_j, pw) and mark this record fresh.

Upon receiving a query (Interrupt,sid, P_i) **from the adversary** S **:**
If there is a record of the form (P_i, P_j, pw) which is fresh then do:
Mark the record interrupted and set $\mathbf{dPT}(sid, P_i) := 1$

Upon receiving a query (TestPwd,sid, P_i, pw') **from the adversary** S **:**
If there is a record of the form (P_i, P_j, pw) which is fresh , then do: Set $\mathbf{dPT}(sid, P_i) := 0$. If $pw = pw'$, mark the record compromised and reply to $\mathcal{S}$ with "correct guess". If $pw \neq pw'$, mark the record interrupted and reply with "wrong guess".

Upon receiving a query (DelayedTestPwd,sid, P_i, pw') **from the adversary** S **:**
If there is a record of the form (P_i, P_j, pw) which is completed where sk was passed to P_i upon the NewKey query and $\mathbf{dPT}(sid, P_i) = 1$ do:
If $pw = pw'$, reply to $\mathcal{S}$ with sk. If $pw \neq pw'$, reply with "wrong guess".
In any case set $\mathbf{dPT}(sid, P_i) := 0$.

Upon receiving a query (NewKey,sid, P_i, sk) **from** $\mathcal{S}$ **where** $|sk| = k$**:**
If there is a record of the form (P_i, P_j, pw), and this is the first NewKey query for P_i, then:

- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, sk) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, pw') with $pw' = pw$, and a key sk' was sent to P_j and (P_j, P_i, pw) was fresh at the time, then output (sid, sk') to P_i.
- In any other case, pick a new random key sk' of length k and send (sid, sk') to P_i.

Either way, mark the record (P_i, P_j, pw) as completed.

Figure 6: Ideal functionality $\mathcal{F}_{\text{ipwKE}}$ re-presenting balanced PAKE with implicit authentication, i.e. without explicit key confirmation messages. The text without the grey and yellow sections corresponds to the definition from [CHK+05] for $\mathcal{F}_{\text{pwKE}}$. Note that in our proof we need to extend this functionality by allowing for one single DelayedTestPwd (dPT) query if the honest party has been interrupted prior to session key generation, as detailed in the highlighted grey extension text.

Definition 1. SDH problem: Given a set of three random generators of a cyclic group of prime order p (Y_a, G_1, G_2) with $G_1 = {Y_a}^{1/y_a}$ and $G_2 = {Y_a}^{1/y'_a}$; provide $Y_b \neq 1$ and (K_1, K_2) such that K_1 is a solution of a first CDH problem for generator G_1 and points (Y_a, Y_b) and K_2 is a solution of the second CDH problem for generator G_2 and points (Y_a, Y_b) with $K_1 = Y_b^{y_a}$ and $K_2 = Y_b^{y'_a}$.

This definition captures the capability of an active adversary in a SPEKE-like protocol. The active adversary is given the power of providing the honest party a point Y_b of his choice, e.g. by choosing a scalar y_b and calculating $Y_b = G_1^{y_b}$ and calculating $K_1 = Y_a^{y_b}$. The SDH requires then to additionally solve the CDH problem for the tuple (G_2, Y_a, Y_b) with a second generator G_2. This problem could alternatively also be defined as follows.

Definition 2. SDH problem, equivalent alternative representation: Given a set of generators of a group with prime order p (G_1, G_2, Y_a) with $G_2 = G_1^{\alpha}$; provide $Y_b \neq 1$ and a tuple (K_1, K_2) such that there exists two scalars y_b and y'_b such that the four equations $Y_b = G_1^{y_b}$, (1); $y_b = \alpha \times y'_b$, (2) and $Y_a^{y_b} = K_1$, (3); $Y_a^{y'_b} = K_2$ (4) hold.

As seen in this alternative formulation of the SDH problem, it is required to somehow relate the discrete logarithm of the generators G_1 and G_2 for deriving any information from equation (2). Without the capability of exploiting the information from equation (2) the problem is under-specified.

Note that an adversary that is able to solve the CDH problem for any generator G and two points (Y_a,Y_b) is also able to solve the SDH problem. Moreover any adversary capable of solving the DLP problem is also able to solve the SDH problem. I.e. hardness of SDH implies the hardness of CDH.

In [PW17] the SDH problem was shown to be intracable in the generic group model of [Sho97].

5.4 Game-based proof

Theorem 1. *The protocol CPace from figure 5 securely realizes $\mathcal{F}_{ipwKE}$ in the $\mathcal{F}_{RO}$ hybrid model in the presence of adaptive adversaries under the assumptions*

- *of the hardness of the SDH problem in $\mathcal{J}$ (A0)*
- *that the number of elements $X \in \mathcal{J}_m$ is not significantly smaller than the order of the group and that for any given randomly sampled group element $X \in \mathcal{J}_m$ the inverse map $\mathsf{Map2Point}^{-1}(X, l)$ is indistinguishable from a random string of length l (A1)*
- *that for any base point B and uniformly distributed random string s, $\mathsf{Map2Point}(s)$ is close to uniformly distributed within the image of $\mathsf{Map2Point}$ and the probability for finding an exponent y such that $B^y = \mathsf{Map2Point}(s)$ is negligible. (A2).*

Note that the assumption A1 is covered by the security guarantees of maps such as Elligator2 [BHKL13] and will allow us for covering the case of fully adaptive adversaries. Without this assumption we would not be able to resolve the commitment problem linked to the Diffie-Hellman substep for adaptive adversaries.

Assumption A2 closely matches the requirements from [BFK09] for the Map2Point operation in PACE. As pointed out also in the section above, the important aspect is that an adversary must not find a discrete logarithm relationship in between two points generated by the mapping algorithm for two separate input strings.

5.4.1 Game G_0 : Real Game

G_0 is the real game in the random-oracle model using the functionality $\mathcal{F}_{\mathrm{RO}}$ from figure 7. The parties P_i receive NewSession queries from all simulated honest parties. These queries contain the passwords provided by the environment $\mathcal{Z}$. P_i then executes the actions of initially honest parties in the protocol. In the event of corruptions, the internal state of the parties is passed to the real-world adversary algorithm $\mathcal{A}$. The subroutine library $\mathcal{F}_0$ is empty.

5.4.2 Game G_1 : Simulation of the random oracle

Here we modify the previous game by replacing the calls $\mathsf{H}_n(q)$ to the original $\mathcal{F}_{\mathrm{RO}}$ hash ITI by a separate implementation. We let $\mathcal{S}_1$ maintain an initially empty list Λ of value pairs (n, q, g, r). For any hash query $\mathsf{H}_n(q)$ such that $(n, q, *, r)$ appears in Λ from any of the ITI libraries, the returned answer is r. In case that no query q has yet occurred, we handle separately the cases of $n = 1$ and $n \neq 1$. In case of $n \neq 1$ we implement the conventional random-oracle model by choosing a new random r of length k, by storing $(n, q, 0, r)$ in Λ and by returning r to the calling ITI.

For $n = 1$ instead, we aim at generating a random string r such that the discrete logarithm of the point $\mathsf{Map2Point}(r)$ is known. For this purpose we first generate a random point G whose discrete logarithm is known and use the inverse map $\mathsf{Map2Point}^{-1}(G, k)$ for converting it into a bit string of length k. We use the guarantee, that for any random point G the string $r = \mathsf{Map2Point}^{-1}(G, k)$ is indistinguishable from a random value.

For calculating the random point G, we first choose a random nonzero value g being smaller than the order of the group. We calculate the point on $\mathcal{J}, G = B^g$. We then test whether G is in the image of Map2Point, $\mathcal{J}_m$. If G is not in the image, we restart with a new random value g until $G \in \mathcal{J}_m$. This is guaranteed to succeed in probabilistic polynomial time because $\mathcal{J}_m$ is large by assumption A2.

Then we calculate $r = \mathsf{Map2Point}^{-1}(G, k)$, record $(1, q, g, r)$ in Λ and return r.

Since the inverse map returns a string indistinguishable from a random string by the assumption A1 G_0 and G_1 are indistinguishable for the environment.

5.4.3 Considering the chance that an adversary guesses sk by chance. (former paper versions: G_2)

In this step, we consider the probability of obtaining the session key sk without querying the hash oracle H_2 for K. We assume that the probability of successfully guessing ISK is the same as the probability of guessing K. (In earlier versions of this proof we did want to make the simulator to follow different strategies, however it turned out that we could not always detect this when facing the UC dummy adversary and do not actually need the adaptions of making the simulator abort. G_2 was retained in the proof in order to keep the numbering consistent with the earlier versions. G_2 is now un-modified in comparison to G_1 and thus indistinguishable.

5.4.4 Game G_3 : Restrict the access to the password.

Here, when receiving the passwords for party P_i from the environment, we let P_i pass them directly to the subroutine library $\mathcal{F}_3$ and allow the rest of the program $\mathcal{S}_3$ no longer access the password unless the simulated party P_i gets corrupted. We let $\mathcal{S}_3$ inform $\mathcal{F}_3$ in case that a party got corrupted such that $\mathcal{F}_3$ returns the password in this case. We add also an implementation of the TestPwd query to $\mathcal{F}_3$ and implement it according to the spec. of $\mathcal{F}_{\mathrm{ipwKE}}$. In this game, we preliminarily also add a SamePwd query to $\mathcal{F}_3$ that returns true if the passwords match.

As a result, the password-derived generator G from figure 2 is no longer known to the ITI P_i . We therefore cannot start the original protocol from the beginning and need to refactor the ITM for the parties as well.

Instead we let the ITI P_i calculate the protocol messages Y_a and Y_b to be random multiples of the group's base point B, $Y_a = B^{z_a}$ and $Y_b = B^{z_b}$. We also let P_i maintain an initially empty list Γ and store the secret scalars of simulated honest parties $(sid, P_i, y \text{ unknown}, z_a)$ and $(sid, P_j, y \text{ unknown}, z_b)$ in this list together with the respective session id.

Since powers of G from game G_2 generate $\mathcal{J}$, the resulting points Y_a, Y_b take any value on $\mathcal{J}$ with equal probability (except for the neutral element) for honest parties. So do the points generated in G_3 . The public messages in G_2 and G_3 are, thus, indistinguishable for the environment.

If the simulated honest points Y_a or Y_b have been modified by the adversary in transmission or generated by the adversary, we use the Interrupt query of $\mathcal{F}_{\text{ipwKE}}$.

In case that a party P_i gets corrupted before calculating K, we need to hand over $\mathcal{A}$ something being consistent with the internal state from P_i in G_2 , notably the values y (y_a or y_b respectively for client and server).

In the case of corruption, $\mathcal{F}_3$ grants us access to the secretly stored pw from its internal state. $\mathcal{S}_3$ may then take the password and the session id and make a corresponding hash query to H_1. We then retrieve the secret scalar value g from Λ. We fetch the party P_i's secret scalar z from Γ and calculate $y = z/g$. We add the party's secret scalar z to the record in Γ with y and hand over y, pw to $\mathcal{A}$.

In case that any party gets corrupted after calculating K but before calculating the final H_2, we perform the secret scalar correction above and recalculate a new $K = Y_r^y$ by using the received point value Y_r and pass K to $\mathcal{A}$.

The code for the verification handling for the received points Y_r can remain unchanged in comparison to Game G_2 .

In case that the point verification fails for any party, we do not generate a session key and do not need to calculate the final hash $\mathsf{H}_2(sid||K)$. In case that the final hash H_2 needs to be calculated for the first of the parties P_i and P_i is still honest, we need to provide a session key to P_i. (Note that this could be either server or client.) We distinguish three cases.

- If the other party was corrupted earlier, we know the other party's password pw'. We then may issue a TestPwd query to $\mathcal{F}_3$. If the guess was correct, we learn the local secret scalar value y by the method described above and calculate $K = Y_r^y$ with the received remote point Y_r. We query $sk = \mathsf{H}_2(sid||K)$ for the corrected value of K and return the result to P_i. If the guess was wrong, we sample a new random key sk and return it to P_i.

- If the other party is still honest, we sample a new random key sk and send it to P_i and record this session key together with the session id and the party identifiers (sid, P_i, P_j, sk).

- If the other party is impersonated by $\mathcal{A}$ we also sample a new random key sk and send it to P_i and record this session key together with the session id and the party identifiers (sid, P_i, P_j, sk). Note that (according to the previous game), we will be returning a distinguishable key sk iff $\mathcal{A}$ somehow managed to guess the value $K = Y_r^y$ in G_2 . We will calculate the corresponding probability GuessK in section 5.5.

The remaining task is to calculate the session key sk for a second party P_j if it is not corrupted until the very end or corrupted before calculating sk. In any of these two cases, we know that two messages Y_a and Y_b must have been delivered by $\mathcal{A}$and we know that the received points are not from a low order sub-group (or the neutral element). Also,

because we know that we have to simulate session key generation for the second time, we know that the first party was honest until the end of the protocol.

If the second party is also honest until the very end, we make a SamePwd query introduced temporarily to $\mathcal{F}_3$. If the passwords match and if the session is not marked as DHFails, we return the same sk value to P_j as for the first party, otherwise we sample a new random key sk' and return this one to P_j.

In case that the second party P_j gets corrupted after calculating K we first correct K using the secret exponent g retrieved from Λ. In case that we did not recognize a non-destructive modification of the Diffie-Hellman points by the mutual-compensation marker for the session, we just sample a new value for sk' by the interface of the random oracle $sk' = \mathsf{H}_2(sid||K||Y_a||Y_b)$ and pass sk' to $\mathcal{A}$. There is only a negligible chance of collision with the key sk sent to the first party, since both sk and sk' have been randomly sampled. There is also only a negligible chance that $\mathcal{A}$ managed to make both parties issue the same session key despite different passwords by modification of the transmitted points. For this reason G_3 and G_2 are indistinguishable for this case.

If the Diffie-Hellman points have not been modified, the session key issued in G_2 depends on the password. We learn the party's password pw from $\mathcal{F}_3$. We then may issue a TestPwd query to $\mathcal{F}_3$. If the guess was correct, we have to provide the same session key to $\mathcal{A}$ as for the first party if DHFails is not recognized. For this purpose, we program the value $\mathsf{H}_2(sid||K||Y_a||Y_b) := sk$ to the session key returned to the first party. This could fail only, if the oracle H_2 already has been queried for $(sid||K||Y_a||Y_b)$, again corresponding to the probability GuessK that we deal with in section 5.5. (Note that it is for this re-programming operation that we will later need to be granted access to the session key issued to the client by the ideal functionality $\mathcal{F}_{\text{ipwKE}}$. Otherwise we could not give $\mathcal{S}$ access to the session key that would have been calculated by honest parties for corruptions occurring just after executing the hash function.)

The messages Y_a, Y_b generated in Game G_2 and G_3 are indistinguishable for the environment because they come from the same distribution. (Note that for this precise aspect, the appropriate co-factor handling was mandatory!) Also the session keys are sampled from an indistinguishable uniform distribution in both cases. Session keys delivered to parties P_i and P_j match under the same conditions as in G_2 . Inserting points on the group's twist by the adversary always leads to different session keys for both parties, just as in G_2 . G_2 and G_3 are, thus, indistinguishable for $\mathcal{Z}$.

In G_2 the adversary is able to make an attempt to impersonate a honest party with a password pw'. I.e. the adversary follows the protocol just as a honest party. If the passwords match, the adversary will be calculating the same session key as the honest party. We need to simulate this situation also in G_3 in a way that is indistinguishable for the environment. The behaviour is symmetric for initiator and responder, so we assume here that the adversary takes over the responder role.

We proceed as follows. We simulate the first message Y_a as $Y_a = B^{y_a}$, as discussed above. We will observe that the adversary queries the random oracle H_1 for the specific sid, party identifiers and the password pw'. The simulator observes these queries and sets up a list G_l of candidate generators for this session id. Since game G_1 the simulator knows the corresponding secret exponents g_l such that $G_l = B^{g_l}$. The adversary follows the protocol specification and calculates $Y_b = G_1^{y_b}$ which we learn from the communication. Subsequently the adversary will query the random oracle $\mathsf{H}_2(sid||K_1)$ for the point $K_1 = Y_a^{y_b} = B^{y_a \times y_b}$. The simulator does observe this query to H_2. In case that K_1 matches $Y_b^{y_a \times (1/g_l)}$ for one of the candidates, we learn the password pw'. We then use the TestPwd query for finding out whether the passwords actually match. Depending on the result we provide the same or different session keys. We have control over the session keys passed to the honest party by $\mathcal{F}_{\text{ipwKE}}$ and the ability to program a chosen session key for H_2. Note that in order to allow

for this simulation strategy, we need to be able to call the TestPwd query also in case that a key already has been delivered to the honest party. This requires the extension of the relaxed version of $\mathcal{F}_{\text{ipwKE}}$ allowing for one single DelayedTestPwd query also for a session that is already completed. In both games, both parties will be given matching session keys if the honest party and the adversary used the same password. If they did not use the same password, they will obtain different session keys with overwhelming probability in both games.

Moreover, we need to rule out the case that an active adversary comes up with a second point K_2 such that it is matching another one of the values $K_2 = Y_b^{y_a \times (1/g_k)}$ from the password candidate list. However, this is ruled out by the SDH assumption (A0).

5.4.5 Game G_4 : Merge the key generation procedures to the functionality $\mathcal{F}_4$.

In this game we essentially only do code-refactoring and move the code responsible for session key generation to the ITM $\mathcal{F}_4$. We make $\mathcal{F}_4$ implement exactly the functionality $\mathcal{F}_{\text{ipwKE}}$. (Note that in line with conventions in UC $\mathcal{F}_4$ gives access to the passwords pw in case of corruptions.) We remove the SamePwd query from the list of queries for $\mathcal{F}_4$ because now, $\mathcal{F}_4$ could easily check itself for password identity in its internal storage. Within $\mathcal{S}_4$ we finally replace the sampling of the session keys by calls to the NewKey query of the ideal functionality.

Since between G_3 and G_4 no functionality change is present, G_3 and G_4 are indistinguishable for $\mathcal{Z}$.

5.5 Proof that probability GuessK in G_3 is negligible

5.5.1 Passive adversary

In G_2 an eavesdropping adversary has access to the published points Y_a and Y_b generated by honest parties. In G_2 he could make guesses for the password and derive the corresponding generator G from the known sid. If the adversary was able to calculate K in G_2 from known G, Y_a and Y_b without knowing y_a and y_b she would be able to solve the computational Diffie-Hellman problem, which is implied by the SDH assumption A0.

5.5.2 Active adversary

Now let us consider the case that the adversary did provide one of the points Y_a and Y_b while the other point was calculated from G by a honest party. The case is symmetric for client and server, so let us assume here that the adversary provided a chosen Y_a. We will base the analysis here on an information-theoretic reasoning.

G_3 can be distinguished by $\mathcal{Z}$ from G_2 iff the impersonating adversary succeeds in obtaining the real-world protocol's honest party point $K = Y_a^{y_b}$ in G_2 . This implies that the adversary is able to calculate K from the following set of information: Y_b, a chosen Y_a and a guessed G. Note that upon the unknown choice of the honest party y_b any point in the prime-order subgroup of $\mathcal{J}$ could result for K with uniform probability. The amount of required information for calculating K is thus governed by the order of the prime-order subgroup q and corresponds, e.g. to $\log_2(q)$ bits.

The only information that the adversary disposes of is the fact that for the correct password-derived generator G the equation $Y_b = G^{y_b}$ holds. Let us consider an adversary $\mathcal{A}'$ which knows the discrete logarithm of Y_b with respect to a base point B. I.e. she knows z such that $Y_b = B^z$. $\mathcal{A}'$ is clearly more powerful than $\mathcal{A}$ because she could calculate $Y_b = B^z$. $\mathcal{A}'$ has now access to the equation $g \times y_b = z$ which contains more information than $Y_b = G^{y_b}$. Still since g is unknown to the adversary by assumption A2, G could be any point in the image of Map2Point with close to uniform probability by assumption A2.

The functionality $\mathcal{F}_{\text{RO}}$ proceeds as follows, running on security parameter k with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$:

$\mathcal{F}_{\text{RO}}$ keeps a list L (which is initially empty) of pairs of bit strings.
Upon receiving a value (sid, m) with $(m \in \{0,1\}^*)$ from some party P_i or from $\mathcal{S}$, do:

- If there is a pair $(m, (\tilde{h}))$ for some $\tilde{h} \in \{0,1\}^k$ in the list L, set $h := \tilde{h}$.
- If there is no such pair, choose uniformly $h \in \{0,1\}^k$ and store the pair $(m, h) \in L$.

Once h is set, reply to the activating machine (i.e., either P_i or $\mathcal{S}$) with (sid, h).

Figure 7: Ideal functionality $\mathcal{F}_{\text{RO}}$.

This means that there is no significant amount of information that $\mathcal{A}'$ could learn (at most $\log_2(q/r)$ bits, where r is the number of points in the image of Map2Point). This amount of information is not sufficient for calculating K.

Under the given assumption set, the probability GuessK is, thus, negligible. [1] This makes the real-world execution of the CPace protocol indistinguishable from the ideal world. □

5.6 Remarks regarding the ordering of the messages and efficiency

Note that in this proof we have assumed that the server party starts with the communication round. In fact, since the services provided to the two parties by the ideal functionality are identical and since the protocol is perfectly symmetric, we could interchange the server and client roles for the balanced PAKE sub-protocol CPace. This might be useful, specifically in case that the scalar multiplication takes comparable time as message delivery.

6 Proof for the augmented protocol AuCPace

6.1 Technical details

For carrying out the proof, the previous security analysis in [HL18b] aimed at re-using functionalities from previous papers wherever possible, specifically the ideal functionality $\mathcal{F}_{\text{apwKE}}$ from [GMR06]. As pointed out By J. Hesse in [Hes19] and as a result of the CFRG review process regarding OPAQUE and AuCPace, adaptions of the functionality in [GMR06] were considered necessary.

Specifically, $\mathcal{F}_{\text{apwKE}}$ from [GMR06] aimed at flexibly and simultaneously considering protocols with and without explicit key confirmation. The chosen approach for $\mathcal{F}_{\text{apwKE}}$, however, turned out to have significant drawbacks.[Hes19]

For the AuCPace protocol, we specifically see applications with and without explicit key confirmation messages. The earlier paper [HL18b] did only consider the case of explicit authentication. The security proof in this revised paper considers an implicitly authenticated version of AuCPace without key confirmation messages.

6.1.1 Adaptions to $\mathcal{F}_{\text{apwKE}}$, $\mathcal{F}_{\text{aipwKE}}$

$\mathcal{F}_{\text{apwKE}}$ as formulated in [GMR06] allows for a TestPwd query only once and only if a session is not yet completed. Unfortunately, for implicitly authenticated protocols, password tests remain possible for the adversary also after session keys have already been delivered

[1] Note that if $\mathcal{A}$ knows g with $G = B^g$ there is an obvious attack. When providing a chosen point $Y_a = B$, K could be calculated as $K = Y_b^{1/g}$.

The functionality $\mathcal{F}_{\text{aipwKE}}$ is parametrized by a security parameter k. It interacts with an adversary $\mathcal{S}$ and a set of parties via the following queries:

Password storage and authentication sessions

Upon receiving a query (StorePWfile,sid, P_i, P_j, pw) **from party** P_i or P_j **:**
If this is the first StorePWfile query, store password data record (file, P_i, P_j, pw) and mark it uncompromised.

Upon receiving a query (CltSession,$sid, ssid, P_i, pw$) **from party** P_i **:**
Send (CltSession, $sid, ssid, P_j, P_j$) to $\mathcal{S}$, and if this is the first CltSession query for $ssid$, store session record $(ssid, P_i, P_j, pw)$ and mark it fresh.

Upon receiving a query (SvrSession,$sid, ssid$) **from party** P_j **:**
If there is a password data record (file,P_i, P_j, pw) then send (SvrSession,$sid, ssid, P_i, P_j$) to $\mathcal{S}$, and if this is the first SvrSession query for $ssid$, store session record $(ssid, P_j, P_i, pw')$ and mark it fresh.

Stealing password data

Upon receiving a query (StealPWfile,sid) **from adversary** $\mathcal{S}$**:**
If there is no password data record, reply to $\mathcal{S}$ with "no password file". Otherwise do the following. If the password data record (file,P_i, P_j, pw) is marked uncompromised, mark it as compromised. If there is a tuple (offline ,pw') stored with $pw = pw'$, send pw to $\mathcal{S}$, otherwise reply to $\mathcal{S}$ with "password file stolen" and the list of all client parties that have a filerecord stored in $\mathcal{F}_{\text{aipwKE}}$.

Upon receiving a query (OfflineTestPwd,sid, pw') **from adversary** $\mathcal{S}$**:**
If there is no password data record, or if there is a password record (file,P_i, P_j, pw) that is marked uncompromised, then store (offline ,pw'). Otherwise, do: If $pw = pw'$, reply to $\mathcal{S}$ with "correct guess". If $pw \neq pw'$, reply with "wrong guess".

Active session attacks

Upon receiving a query (Interrupt,$sid, ssid, P$) **from the adversary** S **:**
If there is a record of the form $(ssid, P, P', pw)$ which is fresh then do: Mark the record interrupted and set **dPT** $(ssid, P) := 1$.

Upon receiving a query (TestPwd,$sid, ssid, P, pw'$) **from adversary** $\mathcal{S}$**:**
Set **dPT**$(ssid, P) := 0$. If there is a session record of the form $(ssid, P, P', pw)$ which is fresh, then do: If $pw = pw'$, mark the record compromised and reply to $\mathcal{S}$ with "correct guess". Otherwise, mark the session record interrupted and reply with "wrong guess".

Upon receiving a query (DelayedTestPwd,$sid, ssid, P, pw'$) **from adversary** $\mathcal{S}$**:** If there is a record of the form $(ssid, P, P', pw)$ which is completed and **dPT**$(ssid, P) = 1$ do: Set **dPT**$(ssid, P) := 0$. If $pw = pw'$ reply with "correct guess", otherwise reply with "wrong guess".

Upon receiving a query (SvrImpersonate,$sid, ssid$) **from adversary** $\mathcal{S}$**:**
If there is a session record of the form $(ssid, P_i, P_j, pw)$ which is fresh, then do: If there is a password data record (file, P_i, P_j, pw) that is marked compromised, mark the session record compromised and reply to $\mathcal{S}$ with "correct guess", else mark the the session record interrupted and reply with "wrong guess".

Key Generation

Upon receiving a query (NewKey,$sid, ssid, P, key$) **from adversary** $\mathcal{S}$**, where** $|key| = k$**:**
If there is a record of the form $(ssid, P, P', pw)$ that is not marked completed, then:

- If this record is compromised, or either P or P' is corrupted, then output $(sid, ssid, key)$ to P.
- If this record is fresh, there is a session record $(ssid, P', P, pw')$,$pw' = pw$, a key key' was sent to P', and $(ssid, P', P, pw)$ was fresh at the time, then let $key'' = key'$ and output $(sid, ssid, key'')$ to P.
- , else pick a random key key'' of length k and output $(sid, ssid, k'')$ to P.

Finally, mark the record $(ssid, P, P', pw)$ as completed.

Figure 8: Ideal functionality $\mathcal{F}_{\text{aipwKE}}$ for verifier-based PAKE with explicit authentication. In comparison to [GMR06] we did apply a wording change (underlined) by replacing Impersonate with SvrImpersonate for making it more explicit that this message models impersonation of the *server*. We removed the TestAbortquery. Also we allowed the password for the server be registered by both, the server *and* the client (green shaded extension). When ommitting the cyan section, we obtain the "*strong*", pre-computation-attack resistant version $\mathcal{F}_{\text{SaipwKE}}$. The grey-shaded extensions allow for one single delayed password test also for completed sessions, as needed for capturing the security properties of implicitly authenticated key exchange.

The functionality $\mathcal{F}_{\text{tunnel}}$ provides the service of confidential and authentic message delivery from party P_i to P_j:

Upon receiving a query (TunnelMsg,sid,P_i,P_j,$data$) **from party** P_i**:**
Send (TunnelMsg,sid,P_i,P_j,$data$) to P_j.

Figure 9: Ideal functionality $\mathcal{F}_{\text{tunnel}}$ as used for password registering in the real-world protocol.

to a honest party. In order to capture this property, the latest version of the OPAQUE paper [JKX19] introduces an interrupt query which will allow the adversary for a delayed password test. Here, we follow the aproach of the revised OPAQUE paper.

We refer to our functionality as $\mathcal{F}_{\text{aipwKE}}$, augmented *implicitly* authenticated password-based key exchange.

6.1.2 Password hash

The second technical aspect to consider is the handling of the $\mathsf{PBKDF}_\sigma(pw$,username,salt) function. In the proof, we treat PBKDF as a separate hash function H_6 and model it as a random oracle $\mathsf{PBKDF}_\sigma(pw$,username,salt) = $\mathsf{H}_6(pw||\sigma||$ username $||$ salt).

The third technical aspect stems from the fact, that the UC simulation model based on Turing machines does not naturally allow for the concept of human users with "user names" and authorizations. Instead we assume that the client's identifier P_i takes over the role of the user name and ignore the concept of authorization here. The full protocol as used for the proof is shown in figure 10.

6.1.3 Keeping the clear-text password secret for the server

The fourth aspect is related to password registration. In the original formulation of $\mathcal{F}_{\text{apwKE}}$ in [GMR06] the password is registered on a server by giving it the clear-text password pw. We have provided the extension that we also allow the password to provided only to the client in clear-text form. Our version $\mathcal{F}_{\text{aipwKE}}$ thus provides also for the option of registering the password on the server by a query that is issued by the client (green shaded extension in figure 8).

In order to formalize the process of password registration in a setting where only the client has access to the clear-text password, we introduced the ideal functionality $\mathcal{F}_{\text{tunnel}}$ in figure 9.

6.1.4 Difference between server corruption and server compromise

We adhere to the convention from [GMR06] and use "server compromise" for the event of stealing the server's persisted state. We use the terminology denote "corruption" for events where the adversary gains control over a party during session establishment.

6.2 Proof

Theorem 2. *The protocol from figure 10 securely realizes $\mathcal{F}_{aipwKE}$ in the ($\mathcal{F}_{RO}$, $\mathcal{F}_{ipwKE}$,$\mathcal{F}_{tunnel}$) hybrid model in the presence of adaptive adversaries under the assumption of the hardness of the computational Diffie-Hellman problem in $\mathcal{J}$ (A3).*

6.2.1 Game G_0 : Real Game

G_0 is the real game in the random-oracle model using the functionality $\mathcal{F}_{\text{RO}}$ from figure 7 for calculating the password hash PBKDF. Honest parties P_i execute the actions of

The implicitly authenticated AuCPace protocol

Setup: This protocol uses a random oracle functionality $\mathcal{F}_{\mathrm{RO}}$ for the password hash PBKDF_σ (H_6) with a parametrization σ and salt size of m_s bits. The protocol also uses a balanced PAKE functionality $\mathcal{F}_{\mathrm{ipwKE}}$ as well as a Diffie-Hellman key exchange protocol (written in exponentiation notation X^y) operating on base point B and group order $m_{\mathcal{J}}$ working on a cryptographic sub-group $\mathcal{J}$ of an elliptic curve and its quadratic twist $\mathcal{J}'$. For password registration it uses the ideal functionality $\mathcal{F}_{\mathrm{tunnel}}$.

Password storage protocol:
When P_i (who is a client) is activated using $\mathsf{StorePWfile}(sid, P_j, pw)$ for the first time, he does the following. He samples a fresh value salt $\leftarrow\!\!\$\, \{0,1\}^{m_s}$, calculates the password hash $w = \mathsf{H}_6(\mathrm{salt}||\sigma||pw||P_i)$ by using $\mathcal{F}_{\mathrm{RO}}$. He then calculates a Diffie-Hellman point $W = B^w$. P_i then sends a query $(\mathsf{TunnelMsg}, sid, P_i, P_j, (\mathsf{StorePWfileSvr}, \mathrm{salt}, \sigma, W))$ to $\mathcal{F}_{\mathrm{tunnel}}$.
When P_j which is a server receives $(\mathsf{TunnelMsg}, sid, P_i, P_j, (\mathsf{StorePWfileSvr}, \mathrm{salt}, \sigma, W))$ from $\mathcal{F}_{\mathrm{tunnel}}$ for the first time, he sets $\mathsf{file}[sid] = (sid, \mathrm{salt}, \sigma, W, P_i)$.

Protocol steps for session establishment:

1. When P_j receives input $(\mathsf{SvrSession}, sid, ssid, P_i)$, he sets up a session record $(sid, ssid, P_i)$ and marks it as fresh. He then waits for a $(\mathsf{username}, sid, ssid, P_i)$ message.

2. When P_i receives input $(\mathsf{CltSession}, sid, ssid, P_j, pw)$ he sets up a session record $(sid, ssid, P_j)$ and marks it as fresh. He then sends message $(\mathsf{username}, sid, ssid, P_i)$ to P_j and awaits a response.

3. When P_j receives input $(\mathsf{username}, sid, ssid, P_i)$, he obtains the tuple stored in $\mathsf{file}[sid]$ (aborting and marking the session record as $\mathsf{completed}$ if this value is not properly defined). He then samples a fresh nonzero exponent x with $0 < x < m_{\mathcal{J}}$ and calculates $X = B^x$ and W^x. He then continues with sending $(\mathsf{hashingParams}, sid, ssid, \sigma, \mathrm{salt}, X)$ to P_i and waits for $\mathsf{contimueAfterHashingParams}$.

4. When P_i receives input $(\mathsf{hashingParams}, sid, ssid, \sigma, \mathrm{salt}, X)$ he verifies X and calculates $w = \mathsf{H}_6(\mathrm{salt}||\sigma||pw||P_i)$. He then calculates X^w. He then sends $(\mathsf{NewSession}, (sid, ssid), P_i, P_j, (sid, ssid, X^w))$ to $\mathcal{F}_{\mathrm{ipwKE}}$ and awaits a response.

5. When P_j receives input $(\mathsf{contimueAfterHashingParams}, sid, ssid, P_i)$ he sends $(\mathsf{NewSession}, (sid, ssid), P_j, P_i, (sid, ssid, W^x))$ to $\mathcal{F}_{\mathrm{ipwKE}}$

6. When P_j receives input $((sid, ssid, ISK)$ from $\mathcal{F}_{\mathrm{ipwKE}}$he outputs $(sid, ssid, ISK)$.

7. When P_i receives input $((sid, ssid, ISK)$ from $\mathcal{F}_{\mathrm{ipwKE}}$he outputs $(sid, ssid, ISK)$.

Stealing the password file: When P_j (who is a server) receives a message $(\mathsf{StealPWfile}, sid, P_j, P_i)$ from the adversary $\mathcal{A}$, if $\mathsf{file}[sid]$ is defined, P_j sends it to $\mathcal{A}$.

Figure 10: AuCPace Protocol definition for the proof of indistinguishability.

the real-world protocol until eventually getting corrupted. Note that we need to let the adversary re-activate the server party with a contimueAfterHashingParams message after receiving the NewSession response from $\mathcal{F}_{\text{ipwKE}}$ for the client. Specifically client entities P_i receive StorePWfile and CltSession queries from the environment $\mathcal{Z}$ and return session keys upon success. Server entities P_j receive SvrSession queries. On the event of corruptions, all the internal state of the parties is passed to the real-world adversary algorithm $\mathcal{A}$, specifically for server corruptions, the password verifier W is returned. Honest parties use the API of $\mathcal{F}_{\text{ipwKE}}$ for generating their session keys and also the adversary is able to interface to $\mathcal{F}_{\text{ipwKE}}$, e.g. when impersonating peers or by TestPwd queries.

6.2.2 Game G_1 : Modeling the random oracle for the hash

In G_1 we replace calls to $\mathcal{F}_{\text{RO}}$ by a separate implementation of the random oracle for PBKDF and the hash functions in a straight-forward way. Again we maintain an initially empty list Λ of value pairs (n, q, r). For any hash query $\mathsf{H}_n(q)$ such that (n, q, r) appears in Λ from any of the ITI, the returned answer is r. In case that no query q has yet occurred we implement the conventional random-oracle model by choosing a new random r of length k, by storing $(n, q, 0, r)$ in Λ and by returning r to the calling ITI.

This game is indistinguishable from game G_0 since the hash results both come from the same uniform distribution.

6.2.3 Game G_2 : Getting rid of the case where the adversary $\mathcal{A}$ correctly guesses a $\mathcal{F}_{\text{RO}}$ output.

This game is almost as the previous, only here we disregard cases where the adversary manages to come up with a value w that has been generated by the PBKDF hash for a given password and salt value, prior to any corresponding query to the PBKDF random oracle. This happens with negligible probability, so Game G_1 and G_2 are indistinguishable for $\mathcal{Z}$. As a consequence, any value w used in the protocol executions has a corresponding query to the PBKDF password hash for a given clear-text password in the $\mathcal{F}_{\text{RO}}$ records.

6.2.4 Game G_3 : Ruling out derivation of w from W

In this game we consider an adversary that is able to calculate w from knowledge of a verifier $W = B^w$ without querying the PBKDF random oracle. An adversary who would come up with such a w would be able to solve the DLP problem, whose hardness is implied by the CDH assumption.

As a consequence, in this game the environment-adversary needs to first query himself the PBKDF random oracle for a given password in order to come up with a secret exponent w that is used by honest parties.

6.2.5 Game G_3 : Handle password inputs to $\mathcal{F}_{\text{ipwKE}}$.

In this game we consider an adversary that is able to calculate X^w in the client role such that it matches the value used by a honest server that has previously produced $X = B^x$ in the real world. In the real world, x is unknown to the adversary. If the adversary is able to come up with W^x, without knowing either w or x, the adversary would have been able to solve the CDH problem.

Correspondingly an adversary in the server role is able to provide a point X of his choice but needs to know either w or W in order to come up with the same W^x that a honest client has calculated in the real world.

As a consequence, the only chance for an adversary to come up with a W^x value that matches the one of a honest party is to follow the protocol with the same password and salt value as used by the honest party (implies queries for w). Alternatively the adversary

may implement a strategy based on stealing the password verifier W by server compromise and impersonate a party in the server role. Note that for this assertion it is a necessary precondition that honest parties abort the protocol if verification of the group element W^x or X^w fails.

6.2.6 Game G_4 : The ideal world

In this game we replace the behaviour of the honest parties with implementations that only use $\mathcal{F}_{\text{aipwKE}}$. With the reasoning above, the only possibility of an adversary to come up with the same session keys generated by $\mathcal{F}_{\text{ipwKE}}$ is to insert the same X^w or W^x values on both sides, which is only possible by following the protocol as a client or by stealing the password verifier W in the server role.

In this game, we need to simulate all protocol messages of honest parties. We also need to simulate the behaviour of the functionality $\mathcal{F}_{\text{ipwKE}}$ that could be accessed and used by the environment-adversary in the previous game. Similarly, the simulator receives all requests by the environment-adversary to $\mathcal{F}_{\text{RO}}$.

- When we receive (CltSession, $sid, ssid, P_j, P_j$) from $\mathcal{F}_{\text{aipwKE}}$, a (username, $sid, ssid, P_i$) message is simulated on behalf of the honest client. This message is indistinguishable from the one from the previous game.
- If the environment-adversary issues a (username, $sid, ssid, P_i$) message and previously a (SvrSession,$sid, ssid, P_i, P_j$) message has been received from $\mathcal{F}_{\text{aipwKE}}$ for the same session id, a database entry needs to be simulated. If the simulator does not yet have a corresponding record for P_i in its simulated database and the given session id, the simulator samples a fresh w, and sets up a new record $(ssid, sid, P_i, w, B^w, salt)$. The simulator then generates a key pair x, X for this session and simulates the (hashingParams, $sid, ssid, \sigma$,salt, X) message on behalf of a honest server party. The salt, σ and X values come from the same distributions in both games.
- If the simulator sees a salt value from his simulated password database in a $\mathcal{F}_{\text{RO}}$ query for the PBKDF password hash, the simulator parses the hash query for the password and issues a (OfflineTestPwd,sid, pw') query to $\mathcal{F}_{\text{aipwKE}}$.
- When receiving a (hashingParams, $sid, ssid, \sigma$,salt, X) message from the adversarial environment, the simulator records the contents.

An adversial environment needs to be given access to the interface that is provided by the $\mathcal{F}_{\text{ipwKE}}$ functionality by a party P_i as in the previous game. The simulator needs to respond to the corresponding queries in an indistinguishable way.

- If a NewSession query is generated for $\mathcal{F}_{\text{ipwKE}}$ by the environment on behalf of a corrupted party P_i in the client role, the simulator parses the password parameter as WX. The simulator moreover queries its simulated password database of the server for the given user and extracts salt values for the given session. If a salt value was communicated in the hashingParams message for the given session id and user, the simulator queries the the random oracle list for a query for any password/salt combination with this salt. The simulator ends up with a list of candidate scalars w. If for any of the candidate scalars, X^w matches WX, the simulator learned the password of an adversary that followed the real-world protocol. As a result, we use the TestPwd query in order to let the ideal world $\mathcal{F}_{\text{aipwKE}}$ issue the same key as our simulation of $\mathcal{F}_{\text{ipwKE}}$.
- Simulation of the TestPwd queries to $\mathcal{F}_{\text{ipwKE}}$ by the environment also receive an input parameter WX which is handled just as the corresponding input to NewSession queries.

If the password was learned by the simulator, a corresponding TestPwd query is issued to $\mathcal{F}_{\text{aipwKE}}$. If the password was not learned, we query $\mathcal{F}_{\text{aipwKE}}$ with TestPwd for a random string as password.

- Upon corruption of a server, we pass a (StealPWfile,sid) to $\mathcal{F}_{\text{aipwKE}}$ and possibly reveceive a list of clear-text passwords from previous (OfflineTestPwd,sid, pw') queries and a list of clients that have registered a password on the server (without getting the passwords themselves). The simulator needs to provide the environment with the password verifiers W and salt values that a real-world server would have stored. If we already have learned the password for a client party P_i from a previous offline test query, we derive w from the password hash and return $W = B^w$ as password verifier. If we did not yet learn the password, we sample a fresh w for this client party, record this value and return B^w as password verifier. We record w in an offline-reprogram value list for the pair of parties (P_i, P_j). If a subsequent password hash query is issued to $\mathcal{F}_{\text{RO}}$ and the offline test query of $\mathcal{F}_{\text{aipwKE}}$ reveals the password for this party P_i, we reprogram the random oracle to yield the w value that belongs to the verifier $W = B^w$ that we did commit to earlier. The values of W,salt and σ come from the same distribution as in the previous game.

- The adversary may also issue Interrupt queries to the simulated $\mathcal{F}_{\text{ipwKE}}$. The simulator converts these to corresponding Interrupt queries to $\mathcal{F}_{\text{aipwKE}}$. Both implementations ($\mathcal{F}_{\text{ipwKE}}$ and $\mathcal{F}_{\text{aipwKE}}$) allow for delayed password tests under the same conditions.

- Subsequent DelayedTestPwd queries for the simulated $\mathcal{F}_{\text{ipwKE}}$ are simulated by calling the corresponding queries in $\mathcal{F}_{\text{aipwKE}}$. If the password was learned by the simulator, a corresponding DelayedTestPwd query is issued to $\mathcal{F}_{\text{aipwKE}}$. If the password was not learned, we query $\mathcal{F}_{\text{aipwKE}}$ with TestPwd for a random string as password.

- If the simulator receives a NewSession query for the simulated $\mathcal{F}_{\text{ipwKE}}$ for a party P_j in the server role, the simulator parses the password parameter of the query as WX. The simulator also fetches the adversially generated X value from a previous hashingParamsmessage. Then the simulator visits two sources for possible values w such that $WX = X^w$. Firstly it visits all of the passwords learned from OfflineTestPwd queries for the given session and party id. If the password had been learned, the simulator issues a TestPwd query to $\mathcal{F}_{\text{aipwKE}}$. If there is no such previously learned password, the simulator visits the offline-reprogram value list of the w values. If WX matches X^w in this case a SvrImpersonate query is issued to $\mathcal{F}_{\text{aipwKE}}$.

- When receiving a NewKey query from the simulated $\mathcal{F}_{\text{ipwKE}}$ we issue a NewKey query to $\mathcal{F}_{\text{aipwKE}}$. If previous TestPwd or server impersonate queries were sucessful, we return matching session keys to both parties.

Generated session keys match in both games under the same conditions. The simulated messages stem from the same distributions in both games. The real-world protocol AuCPace, thus emulates the ideal functionality $\mathcal{F}_{\text{aipwKE}}$ in the $\mathcal{F}_{\text{ipwKE}}$, $\mathcal{F}_{\text{RO}}$, $\mathcal{F}_{\text{tunnel}}$) hybrid model.

□

7 Partial augmentation

7.1 The ideal functionality $\mathcal{F}_{\text{paipwKE}}$ for modeling *partial* augmentation.

For the proof we introduce a new concept for *partial* augmentation of a PAKE protocol. The corresponding functionality is depicted in figure 11. In comparison to $\mathcal{F}_{\text{aipwKE}}$ partial

augmentation ($\mathcal{F}_{\text{paipwKE}}$) gives the attacker the possibility to also impersonate the *client* after having succeeded in compromising the server.

In the partially augmented variant of our protocol, we replace the server-chosen ephemeral key-pair (x, X) by a long-term key pair that is re-used over several login sessions (same *sid*, different *ssid*). We would have liked to choose the key-pair only once at the point, where the server's Turing machine is first instantiated and not upon each password configuration. Unfortunately this is technically not possible in the UC framework, since this would correspond to a shared state over several *sid*. For this reason, we need to let the server choose (x, X) upon password configuration and store W^x together with X in the password file, i.e., just as the password verifier, the public key X becomes part of the shared state for session *sid*.

After receiving the password verifier W through the tunnel, the server calculates the ephemeral pair (x, X) and the point W^x and records the pair of (X, W^x) what we call *personalization* of W for the specific server. The server then may discard x. Note that using a long-term public key essentially halves the computational complexity of AuCPace for the server for the case of the login sessions.

At a first glance, since after a server P_j's compromise, none of the security guarantees with respect to the adversary are maintained, it might be argued that $\mathcal{F}_{\text{paipwKE}}$ does not actually provide any meaningful advantage in comparison to $\mathcal{F}_{\text{ipwKE}}$.

The advantage, however, becomes obvious when considering the IIoT setting with more than two servers sharing the same user credentials. In fact after executing a StealPWfile query on server P_j the adversary has full control over P_j. Note, however that the adversary is *not* given the clear-text password *pw* from P_j upon server compromise. He is only granted the capability to execute OfflineTestPwd queries.

In settings where the adversary may expect other server entities P_k to operate with the same password *pw* as P_j, client impersonation for connections with P_k is still precluded.

Note that this is occurring exactly in the use-case of industrial control plants. There user credentials (password verifiers) may be shared by many small server entities, which may be comparably easily stolen/compromised. In this setting, server compromise might most likely be implemented by invasive attacks on the hardware, e.g., by stealing a first server, un-soldering microcontroller or memory chips and by side-channel attacks that re-open debug ports. In this setting $\mathcal{F}_{\text{paipwKE}}$ provides very meaningful protection to the honest subset of servers. It might be likely to detect theft of the device and the partial augmentation feature might provide a sufficiently large time-window allowing for changing user credentials on the plant.

Also undetected re-insertion of a compromised server in a plant may not be a relevant attack scenario, such that the additional capability of the adversary to impersonate the client on this specific server does not actually degrade the security in practice. Moreover, as we will show, the AuCPace scheme allows for a server-specific configuration for partial and full augmentation. A server entity where non-invasive attacks allowing for a re-insertion into an installation should be considered feasible might choose to implement $\mathcal{F}_{\text{aipwKE}}$ using AuCPace with ephemeral key pair (x, X) while a server where a more invasive attack is presumed necessary in order to compromise the database (leading to device destruction) might choose to use a long-term secret x and as a consequence $\mathcal{F}_{\text{paipwKE}}$.

Similar security guarantees of $\mathcal{F}_{\text{paipwKE}}$ could also be realized if any server uses a different "salt" value for each client, e.g., by letting the server provide a random salt value upon password configuration. This, however precludes mechanisms offering an off-line user credential distribution because the set of all server identities need to be known at the time when the user configures his password. It would not allow for the flexibility to asynchronously add further servers to a plant after password registration. Note also that this way upon password changes, the complex PBKDF_σ password hash would have to be calculated once for each server, significantly reducing the feasible strength of the workload

The functionality $\mathcal{F}_{\text{paipwKE}}$ is an extension to the functionality $\mathcal{F}_{\text{aipwKE}}$ from figure 8. It implements all of the $\mathcal{F}_{\text{aipwKE}}$ queries and extends the capabilities of the adversaries by the following query:

Upon receiving a query (CltImpersonate,$sid, ssid$) **from adversary** $\mathcal{S}$**:**
If there is a session record of the form $(ssid, P_i, P_j, pw)$ which is fresh, then do: If there is a password data record (file, P_i, P_j, pw) that is marked compromised, mark the session record compromised and reply to $\mathcal{S}$ with "correct guess", else mark the the session record interrupted and reply with "wrong guess".

Figure 11: Ideal functionality $\mathcal{F}_{\text{paipwKE}}$ for partial verifier-based PAKE with explicit authentication.

parameters σ.

For the same reason "personalizing" a password hash for a server by hashing it together with the server ID provides weaker security guarantees than partially augmented AuCPace. Either the capability to add new servers to a plant after password registration is lost or a central password distribution server would be required to hold information allowing for impersonating any user. In the AuCPace context the distribution server would only hold information on W not allowing for impersonation attacks without offline dictionary attacks because entering the protocol in client role requires the password-derived scalar w.

7.2 Proof

Theorem 3. *The protocol from figure 10 with using a long-term key-pair (x, X) instead of the ephemeral key pair from step 3 in figure 10 securely realizes $\mathcal{F}_{paipwKE}$ in the ($\mathcal{F}_{RO}$,$\mathcal{F}_{ipwKE}$,$\mathcal{F}_{tunnel}$) hybrid model in the presence of adaptive adversaries under the assumption of the hardness of the computational Diffie-Hellman problem in $\mathcal{J}$.*

We implement the proof for the partially augmented protocol in the UC hybrid model, just as for the fully augmented variant.

Since we assume that the key pair (x, X) is used for several protocol runs, we give the adversary access to the secret exponent x and the value W^x upon server compromise. For this reason, we also have to consider adversaries $\mathcal{A}$ which base their attack strategy on this knowledge. We detect this case, that the adversary was giving a password related string PRS containing W^x as input to the balanced ideal functionality $\mathcal{F}_{\text{ipwKE}}$. In case of such an attack strategy, we let the simulator use the CltImpersonate query of $\mathcal{F}_{\text{paipwKE}}$ in order to make the ideal and real world indistinguishable for the environment $\mathcal{Z}$. □

8 Security of the *strong* AuCPace protocol

We follow the same proof strategy as used for the augmented protocol, however for the proof we need the additional assumptions $A1$ and $A2$ from section 5.4 in order to be able to simulate the random oracle H_1 according to the strategy from section 5.4.2. Additionally we need the "One-More" Diffie-Hellman assumption from [JKKX16], just as required for the generic "compiler" approach for the OPRF in [JKX18].

The protocol is described in 12.

Theorem 4. *The protocol strong AuCPace from figure 12 securely realizes $\mathcal{F}_{aipwKE}$ (without the blue highlighted sections) in the $\mathcal{F}_{RO}$ hybrid model in the presence of adaptive adversaries under the assumptions*

The *strong* implicitly authenticated AuCPace protocol

Setup: This protocol uses a random oracle functionality $\mathcal{F}_{\mathrm{RO}}$ for all of the hash function H_7 and the password hash PBKDF_σ (H_6) with a parametrization σ and salt size of m_s bits. The protocol also uses a balanced PAKE functionality $\mathcal{F}_{\mathrm{ipwKE}}$ as well as a Diffie-Hellman key exchange protocol (written in exponentiation notation X^y) operating on base point B and group order $m_{\mathcal{J}}$ working on a cryptographic sub-group $\mathcal{J}$ of an elliptic curve and its quadratic twist $\mathcal{J}'$. For password registration it uses the ideal functionality $\mathcal{F}_{\mathrm{tunnel}}$.

Password storage protocol:
When P_i (who is a client) is activated using $\mathsf{StorePWfile}(sid, P_j, pw)$ for the first time, he does the following. He samples a fresh value $q \leftarrow\!\!\$\ \{0,1\}^{m_s}$ and derives a password salt with $\mathrm{Z} = (\mathsf{Map2Point}(\mathsf{H}_7(pw||P_i))$ and salt $= Z^q$. With this salt he calculates the password hash $w = \mathsf{H}_6(\mathrm{salt}||\sigma||pw||P_i)$ by using $\mathcal{F}_{\mathrm{RO}}$. He then calculates a Diffie-Hellman point $W = B^w$. P_i then sends a query $(\mathsf{TunnelMsg}, sid, P_i, P_j, (\mathsf{StorePWfileSvr}, q, \sigma, W))$ to $\mathcal{F}_{\mathrm{tunnel}}$.
When P_j which is a server receives $(\mathsf{TunnelMsg}, sid, P_i, P_j, (\mathsf{StorePWfileSvr}, q, \sigma, W))$ from $\mathcal{F}_{\mathrm{tunnel}}$ for the first time, he sets $\mathsf{file}[sid] = (sid, q, \sigma, W, P_i)$.

Protocol steps for session establishment:

1. When P_j receives input $(\mathsf{SvrSession}, sid, ssid, P_i)$, he sets up a session record $(sid, ssid, P_i)$ and marks it as fresh. He then waits for a $(\mathsf{username}, sid, ssid, U, P_i)$ message.

2. When P_i receives input $(\mathsf{CltSession}, sid, ssid, P_j, pw)$ he sets up a session record $(sid, ssid, P_j)$ and marks it as fresh. He then samples a fresh value r and calculates $U = \mathsf{Map2Point}(\mathsf{H}(pw||P_i)^r$. He then sends message $(\mathsf{username}, sid, ssid, U, P_i)$ to P_j and awaits a response.

3. When P_j receives input $(\mathsf{username}, sid, ssid, P_i)$, he obtains the tuple stored in $\mathsf{file}[sid]$ (aborting and marking the session record as $\mathsf{completed}$ if this value is not properly defined). He then samples a fresh nonzero exponent x with $0 < x < m_{\mathcal{J}}$ and calculates $X = B^x$, $UQ = U^q$ and W^x. He then continues with sending $(\mathsf{hashingParams}, sid, ssid, \sigma, UQ, X)$ to P_i and waits for $\mathsf{contimueAfterHashingParams}$.

4. When P_i receives input $(\mathsf{hashingParams}, sid, ssid, \sigma, UQ, X)$ he verifies X and UQ and calculates $w = \mathsf{H}_6(UQ^{(1/r)}||\sigma||pw||P_i)$. He then calculates X^w. He then sends $(\mathsf{NewSession}, (sid, ssid), P_i, P_j, (sid, ssid, X^w))$ to $\mathcal{F}_{\mathrm{ipwKE}}$ and awaits a response.

5. When P_j receives input $(\mathsf{contimueAfterHashingParams}, sid, ssid, P_i)$ he sends $(\mathsf{NewSession}, (sid, ssid), P_j, P_i, (sid, ssid, W^x))$ to $\mathcal{F}_{\mathrm{ipwKE}}$

6. When P_j receives input $((sid, ssid, ISK)$ from $\mathcal{F}_{\mathrm{ipwKE}}$he outputs $(sid, ssid, ISK)$.

7. When P_i receives input $((sid, ssid, ISK)$ from $\mathcal{F}_{\mathrm{ipwKE}}$he outputs $(sid, ssid, ISK)$.

Stealing the password file: When P_j (who is a server) receives a message $(\mathsf{StealPWfile}, sid, P_j, P_i)$ from the adversary $\mathcal{A}$, if $\mathsf{file}[sid]$ is defined, P_j sends it to $\mathcal{A}$.

Figure 12: strong AuCPace Protocol definition for the proof of indistinguishability.

- *that the number of elements $X \in \mathcal{J}_m$ is not significantly smaller than the order of the group and that for any given randomly sampled group element $X \in \mathcal{J}_m$ the inverse map $\mathsf{Map2Point}^{-1}(X, l)$ is indistinguishable from a random string of length l (A1)*
- *that for any base point B and uniformly distributed random string s, $\mathsf{Map2Point}(s)$ is close to uniformly distributed within the image of $\mathsf{Map2Point}$ and the probability for finding an exponent y such that $B^y = \mathsf{Map2Point}(s)$ is negligible. (A2).*
- *of the hardness of the CDH problem in $\mathcal{J}$ (A3)*
- *of the hardness of the One-More-Diffie-Hellmann problem from [JKKX16] in $\mathcal{J}$ (A4)*

We follow the same approach as for the conventionally augmented protocol.
The following additional considerations apply.

- In game G_1 we add an implementation for the password hash H_7 that makes hash queries observable for the simulator and in a way that allows the simulator to derive the secret exponent of the values Z, just as it was used in case of the balanced CPace protocol for H_1. Here we need the assumption (A1).
- In game G_2 we add the consideration that for deriving the password hash w as used by honest parties, it is also a pre-requisite that previously H_7 has been queried for the clear-text password and user combination. The probability that an adversary guesses w without this additional query is neglectable.
- In game G_3 we add the consideration, that an adversary that could calculate UQ from a given U would be able to solve the One-More-Diffie-Hellmann problem which is of neglectable probability according to assumption A4.
- Moreover, in game G_3 we add the consideration, that an adversary that could calculate Z^q only if previously H_7 has been queried for the clear-text password and by information on q. As a consequence the probability that an adversary could calculate the salt value and the password hash could be neglected as long as the value q is not available (e.g. by a server compromise).

In game G_4 we modify the simulator as follows (in comparison to the conventionally augmented protocol). In this game the password is no longer accessible. We need to provide the curve points U and UQ in a way indistinguishable from G_3 , such that we could provide consistent data to the adversary in case of corruptions.

In G_4 we simulate U as a random multiple $B^{(u\, c_{\mathcal{J}})}$ of the group's base point and assign the server a scalar q for each user/$ssid$ pair. Note that in order to be indistinguishable from G_3 , the transmitted points need to come from the same distribution in both games. In order to guarantee this, we need the explicit co-factor treatment for generating the point U in the real world protocol. The server's response point UQ is calculated as $U^{(q\, c_{\mathcal{J}})}$ as in the real-world protocol. We record the value of u for the event of corruptions.

If the client is corrupted, we learn the password and retrieve the secret exponent z for the point $Z = B^z = \mathsf{Map2Point}(\mathsf{H}_7(\text{ username}, pw))$ from our programmed value for the hash function H_7. We then return $r = u/z$ to the adversary (same approach as within the proof for CPace). We then could calculate the salt value as $Z^{(q\, c_{\mathcal{J}})}$.

In case that the server is corrupted, we just return the value of q for the secret exponent of the OPRF.

We re-program the random-oracle for the PBKDF function such that the secret scalar w matches the password verifier W that might have been disclosed earlier to the environment by corrupting the server. The environment could distinguish G_4 from G_3 iff this re-programming operation fails, because the PBKDF hash has previously been queried for

$Z^{(q\, c_{\mathcal{J}})}$. Since r is unknown to the environment and the group order of the point UQ has verified to be large, successful guessing of $Z^{(q\, c_{\mathcal{J}})}$ happens with negligible probability.

Finally, it is necessary to show that an active adversary that impersonates the client is not able to calculate $Z^{(q\, c_{\mathcal{J}})}$ offline even if she knows an arbitrarily large set of pairs $(V, V^{(q\, c_{\mathcal{J}})})$ with $Z \neq V$. This case is ruled out when assuming the hardness of the "One-More" Diffie-Hellmann assumption from [JKKX16].

Regarding the PBKDF password hash, we don't modify the simulator in comparison to the conventionally augmented AuCPace, however as the blue text from the $\mathcal{F}_{\text{aipwKE}}$ definition is ommitted, the OfflineTestPwd will have any effect unless the event of the server compromise. After compromising the server and learning q, however, the adversary could start with password hash queries and offline tests in both games.

This makes games G_4 and G_3 indistinguishable for the environment also in case of the strong augmented protocol. □

9 Discussion and conclusion

In this paper we have presented an updated analysis regarding the candidates CPace and AuCPace for the second round of the CFRG PAKE selection process. In this updated security analysis, we have considered the suggestions gathered in the discussions on the CFRG mailing list.

Specifically, we changed the establishment of the session ID according to the suggestion of [Can19] on the CFRG list. This step removed one message round for the CPace protocol. Secondly, we have added the SDH problem to the list of assumptions in confirmation of the analysis of Stanislav Jarecki. We also have re-phrased the ideal functionalities to have more consistency with the corresponding functionalities as used for the security analsysis of OPAQUE in the latest paper version [JKX19]. Finally, we now have included the transcripts of the messages into the session key calculation by CPace, as recommended by some of the CFRG reviewers. This allowed for a significantly simplified definition of the ideal functionalities.

As a result of this analysis, we conclude that the security is provided for the protocols in the form as they are published in the internet drafts [Haa20c, Haa20b]. A reference implementation of the protocols discussed here is published at [HL18a, Haa20a]. We moreover did not identify intellectual property rights problems with these protocols.

10 Acknowledgements

The authors would like to thank the CFRG reviewers for their dedication and constructive feedback. Specifically, we would like to thank Björn Tackmann, Julia Hesse, Stanislav Jarecki and Hugo Krawczyk.

References

[ACCP08] Michel Abdalla, Dario Catalano, Céline Chevalier, and David Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pages 335–351. 2008. 4, 11

[AFH19] N. Sullivan R. Wahby C. Wood A. Faz-Hernandez, S. Scott. Hashing to elliptic curves, 2019. https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/. 2

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84. Springer, 2005. 3

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2005. 2

[Ber06] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer-Verlag Berlin Heidelberg, 2006. http://cr.yp.to/papers.html#curve25519. 5

[Ber14] Daniel J. Bernstein. 25519 naming. Posting to the CFRG mailing list, 2014. https://www.ietf.org/mail-archive/web/cfrg/current/msg04996.html. 5

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Proceedings*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2009. 2, 15

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013. 6, 15

[BLR04] Boaz Barak, Yehuda Lindell, and Tal Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004. https://eprint.iacr.org/2004/006. 7

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 4-6, 1992*, pages 72–84. IEEE Computer Society, 1992. 2

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In Preneel [Pre00], pages 156–171. 2

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Preneel [Pre00], pages 139–155. 3

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive*, 2000:67, 2000. 4

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001. 3, 4

[Can19] Ran Canetti. SIDs in UC-secure PAKE and KE. Posting to the CFRG mailing list, 2019. https://mailarchive.ietf.org/arch/msg/cfrg/fgwMYLfaa1ZbF_UgKcurse211KE. 7, 31

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 404–421, 2005. 3, 6, 13, 14

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003. 3, 4

[EKSS09] John Engler, Chris Karlof, Elaine Shi, and Dawn Song. Is it too late for PAKE? In *Proceedings of the IEEE Web 2.0 Security and Privacy Workshop*, volume 5, page 17, 2009. 2

[GMR06] Craig Gentry, Philip D. MacKenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2006. 4, 5, 20, 21, 22

[Haa20a] Björn Haase. Repository for reference implementations for CPace and AuCPace, 2020. https://github.com/BjoernMHaase/AuCPace. 31

[Haa20b] Björn Haase. CPace, a balanced composable PAKE, 2020. https://datatracker.ietf.org/doc/draft-haase-cpace/. 2, 31

[Haa20c] Björn Haase. (strong) AuCPace, an augmented PAKE, 2020. https://datatracker.ietf.org/doc/draft-haase-aucpace/. 1, 31

[Hes19] Julia Hesse. Separating standard and asymmetric password-authenticated key exchange. Cryptology ePrint Archive, Report 2019/1064, 2019. https://eprint.iacr.org/2019/1064. 13, 20

[HL17] Björn Haase and Benoît Labrique. Making password authenticated key exchange suitable for resource-constrained industrial control devices. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 346–364. Springer, 2017. 5

[HL18a] Björn Haase and Benoît Labrique. Repository for optimized X25519 field arithmetics code for ARM cortex M4 microcontrollers, 2018. https://github.com/BjoernMHaase/fe25519. 31

[HL18b] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286, 2018. https://eprint.iacr.org/2018/286. 4, 7, 9, 20

[Jab96] David P. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996. 2

[JKKX16] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). Cryptology ePrint Archive, Report 2016/144, 2016. https://eprint.iacr.org/2016/144. 28, 30, 31

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 456–486. Springer, 2018. 3, 6, 9, 10, 13, 28

[JKX19] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: An asymmetric pake protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163, Revision from October 22th 2019, 2019. https://eprint.iacr.org/2018/163. 13, 22, 31

[KTR13] Ralf Kuesters, Max Tuengerthal, and Daniel Rausch. The IITM model: a simple and expressive model for universal composability. Cryptology ePrint Archive, Report 2013/025, 2013. https://eprint.iacr.org/2013/025. 9

[LL97] Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroupp. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 249–263. Springer, 1997. 5

[PJ12] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function., 2012. http://tools.ietf.org/html/josefsson-scrypt-kdf-00.txt. 5

[Pre00] Bart Preneel, editor. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000. 32

[PW17] David Pointcheval and Guilin Wang. VTBPEKE: verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 301–312. ACM, 2017. 2, 3, 6, 13, 15

[RS17] Joost Renes and Benjamin Smith. qdsa: Small and secure digital signatures with curve-based diffie-hellman key pairs. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information*

Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II, volume 10625 of *Lecture Notes in Computer Science*, pages 273–302. Springer, 2017. 5

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997. 15

[SKI10] SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Security proof of Aug-PAKE. *IACR Cryptology ePrint Archive*, 2010:334, 2010. 6