AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT

Björn Haase and Benoît Labrique

Endress+Hauser Conducta GmbH&Co. KG, Germany
bjoern.haase@conducta.endress.com

Abstract.
Increasingly connectivity becomes integrated in products and devices that previously operated in a stand-alone setting. This observation holds for many consumer applications in the so-called "Internet of Things" (IoT) as well as for corresponding industry applications (IIoT), such as industrial process sensors. Often the only practicable means for authentication of human users is a weak password. The security of password-based authentication schemes frequently form the weakest point of the security infrastructure.
In this paper we first expose, why a tailored protocol designed for the IIoT use case is considered necessary. The differences between IIoT and to the conventional Internet use-cases result in largely modified threats and require special procedures for allowing both, convenient and secure use in the highly constrained industrial setting.
Specifically the use of a verifier-based password-authenticated key-exchange (V-PAKE) protocol as a hedge against public-key-infrastructure (PKI) failures is considered important. Availability concerns for the case of failures of (part of) the communication infrastructure makes local storage of access credentials mandatory. The larger threat of physical attacks makes it important to use memory-hard password hashing.
This paper presents a corresponding tailored protocol AuCPace together with a security proof within the Universal Composability (UC) framework considering fully adaptive adversaries. We also introduce a new security notion of partially augmented PAKE that provides specific performance advantages and allows, thus, for suitability for a larger set of IIoT applications.
We also present an actual instantiation of our protocol, AuCPace25519, and present performance results on ARM Cortex-M0 and Cortex-M4 microcontrollers. Our implementation realizes new speed-records for PAKE and X25519 Diffie-Hellman for the ARM Cortex M4 architecture.

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

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) as well as in the corresponding industry setting, the "Industrial IoT" (IIoT). Often communication technology and security protocols are employed that were not originally tailored and designed for the resource-constrained setting and the specific threat model.

In comparison to conventional un-connected devices, security becomes a crucial aspect to consider, specifically in the IIoT. Often the only practicable means for authentication of human users still is a weak password. In fact, the security of password-based authentication

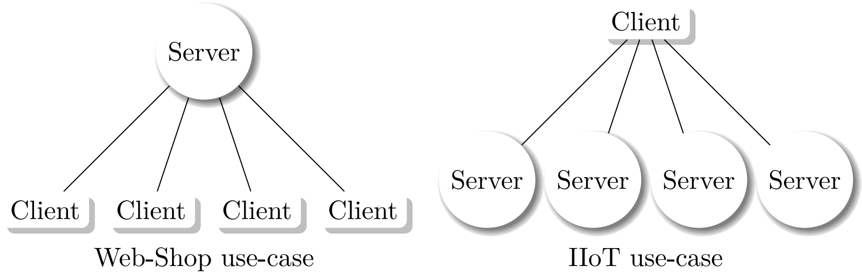

Figure 1: Use-cases for conventional Internet applications and IIoT.

schemes frequently form the weakest point of the security infrastructure. Users tend to use short and easily memorable passwords. For this reason emerging industry standards, such as from the IEC-62443 family rightfully require two-factor authentication for higher security levels (SL-3, SL-4). Still then, however, suitable protection of the second factor "password" remains important.

Today, in most Internet communication and many IIoT applications protocols such as TLS based web servers are used, that were not originally designed for the IIoT use-case. Mostly, a trustworthy and failure-free Public-Key-Infrastructure (PKI) is indispensable for providing even basic protection of passwords, e.g. against phishing or man-in-the-middle attacks. However in the IIoT setting, specifically for industrial installations not fully operating according to standards such as IEC-62443, today integration of the devices in a PKI is not always available.

1.1 Differences between conventional web security and IIoT

One of the most remarkable difference between the conventional web server setting (here referred to as "web shop" use case) and the typical IIoT setting corresponding, e.g. to an industrial plant is illustrated in Figure 1. In the former case few servers, e.g. web servers, interface to many clients, e.g. web browsers which come with pre-installed configuration for certificate authorities trusted by the browser supplier. In the latter case, one single client, e.g. a tablet-computer based human machine interface (HMI), might be used for configuring many servers, e.g. sensors or control valves.

Unlike in the "web shop" use case, in the IIoT setting a dramatically larger number of server certificates needs to be configured and maintained. Often self-signed certificates are used for servers, leading to the significant risk that the essential corresponding configurations on the client side (e.g. browsers) are omitted. Users used to the more convenient "web shop" setting might not even be aware that such configuration is mandatory for security.

As a consequence, the threat of PKI failures should be considered very carefully for remote HMI access to (I)IoT units. This is one of the reasons, why since recently strategies for password protection as a hedge against PKI failures have regained academic and industrial interest [JKX18, PW17, HL17].

Moreover many IIoT devices, notably battery-driven devices, will not be permanently "online". Availability concerns for the case of (partial) failure of networking infrastructure often make it mandatory to locally store user credentials, typically in unprotected memory. Furthermore, IoT devices might be much more exposed to physical attacks. For all of these reasons, protection of passwords forms a crucial point of any (I)IoT security solution.

1.2 Strategies for protecting passwords

For the protection of passwords two complementary approaches could be distinguished. On one hand, memory-hard password hash algorithms such as scrypt [PJ] and Argon2

[BDKJ16] aim at increasing the cost of offline dictionary attacks. They do so by designing the hashing algorithm such that a large, parametrisable amount of memory, e.g. 32 Megabytes or more, is mandatory for efficiently calculating the password hash. This prevents important classes of accelerating techniques based on parallelisation on low-cost and low-power hardware such as application specific integrated circuit (ASICs), field-programmable gate arrays (FPGA) and graphic processing units (GPU).

On the other hand password-based key exchange (PAKE) protocols allow for establishing a secure, high-entropy shared session key over an insecure communication channel. This holds even if only a low-entropy secret key, the password pw is shared.

One of the important advantages of PAKE protocols published since the early works by Bellovin and Merritt and Jablon [BM92, Jab96] is the fact that neither a public-key infrastructure (PKI) nor a trusted hardware component capable of securely storing high-entropy keys is required as prerequisite. PAKE protocols, thus, match very closely the needs of the IIoT use-case.

PAKE protocols essentially come in two variants. Firstly, so-called balanced or symmetric PAKE protocols, for instance [BM92, CHK+05, BFK09, Jab96, BMP00], are designed such that both, initiator and responder parties require that the same password pw is available on both sides. Secondly, so-called verifier-based PAKE protocols (also known as V-PAKE, asymmetric or augmented PAKE) protocols could be distinguished, where the server entity is given access only to a password-verifier W and the clear-text password is only available to the client party, e.g. [W+98, PW17, GMR06, Jab97]. In all known protocols V-PAKE comes with significant additional computational overhead in comparison to PAKE.

A peculiarity in the industrial IIoT setting is that we should be expecting to find the very same password in use on many small devices. Due to password re-use the compromise of one small server might affect the security of a larger infrastructure. On the other hand, availability concerns for the case of failures of (part of) the network infrastructure often make it mandatory to locally store this sensitive data and often no protected storage media, such as smart-card circuits, are available. For this reason verifier-based PAKE using memory hard password hashing should be considered to provide the best possible security strategy regarding HMI authentication for IIoT applications.

1.3 Why industrial instrumentation needs a specially tailored V-PAKE protocol

Unfortunately, it is not uncommon that IoT and industrial devices have only very limited energy and computational resources available, specifically regarding battery-driven wireless devices or if they have to be conformable with the constraint set that applies for intrinsically safe explosion protection (IEC60079-11) [HL17].

For important classes of devices, use of memory-hard password hashing is precluded due to the limited memory and computational capacities. Also the computational complexity of some established V-PAKE protocols that were originally developed for office information technology might prevent actual use in the IIoT. Note that all of the most efficient known augmented protocols (requiring three exponentiations) or components of the protocols, such as AugPake [SKI10] and OPAQUE [JKX18] are covered by patents. It is important to consider that also devices even smaller than typical IIoT devices need sound password protection, such as e.g. legacy fieldbus systems with bluetooth-based wireless HMI interfaces [HL17].

In all of these settings, efficiency considerations and pending patents will be a crucial factor ultimately deciding upon whether or not sound password protection could actually be implemented by manufacturers. Note that specifically for the smallest devices such as e.g. temperature sensors, a significant commercial cost constraint applies. All of, power efficiency, code size and ease-of-implementation are crucial factors that will decisive for actual roll-out of a more secure or more insecure solution. I.e. efficiency is of utmost importance.

Regarding efficiency of V-PAKE solutions, optimisations need to apply on all levels of the security implementation, protocol design, algorithmic optimisations (e.g. group operations on elliptic curves) and low-level arithmetics. Unfortunately, most protocols today were mainly designed with typical office environments in mind.

Regarding the optimisation of V-PAKE on the protocol level for both, security and efficiency, the framework used for the security analysis is of major importance. More idealised assumptions regarding the cryptographic primitives, such as specifically the random-oracle model [BR93], often allow for significantly more efficient constructions. It is also important to consider that authentication protocols are often embedded in a larger system, involving, e.g., protocols allowing for distributing password verifiers to server entities from a central management infrastructure. Note that in the IEC-62443-4-2 draft standard, centralised user account management systems are declared mandatory even at the lowest security level SL-1, providing significant challenges for all devices that are not permanently "on-line" or isolated by purpose from the main company network.

Another example for such an application is given in [KM16] where as mitigation against dishonest server units the clear-text password never leaves the client party. While it is out of the scope of this paper, we consider it highly desirable to prepare a path for formal security proofs for more complex composed solutions such as V-PAKE in conjunction with centrally distributed user credentials. One of the proof strategies simplifying analysis of inclusion of a PAKE substep in a larger system is the Universal Composability framework (UC) by Canetti [Can01], specifically because it allows for modular re-use of reduction proofs.

1.4 Contribution of this paper

This paper aims at contributing to the project of securing industrial IoT applications by comprehensively addressing the efficiency, implementation and patent issues that, today, often still hamper resolution of the prevalent and notorious pitfalls regarding password-based human operator authentication.

We aim at doing so by introducing an efficiency-optimised V-PAKE protocol "Augmented Composable Password Authenticated Connection Establishment" (AuCPace) and an actual instantiation AuCPace25519 that is, to the best of our knowledge, freely usable without inflicting intellectual property rights and specifically tailored for the exact subset of devices where security is most likely to be discarded: Small extremely resource-constrained and/or low-cost devices where integrating sound security is particularly difficult.

Our scheme arranges for use of memory-hard password hashes also on smallest devices with little memory, since it is deferring the costly password hashing to the client entities.

Our work builds up upon the work in [HL17] by explicitly providing formal security proofs for the specific optimised design choices therein, e.g. regarding the cheaper point verification techniques employing twist security and avoidance of costly and memory-consuming point compression. Note that such aspects were just not relevant for the conventional office IT setting for which most PAKE protocols so far have been designed.

We do so by providing a security proof for AuCPace in the UC framework with joint state [CR03]. We base our analysis on the UC-based security model of [GMR06]. We show that our protocol, unlike most Diffie-Hellman-based constructions, could be proven secure in the UC model, even when considering fully adaptive adversaries. We extend this model for allowing a new operation mode that we coined *partially augmented* PAKE. Partial augmentation allows for significant efficiency advantages in comparison to the conventional "full" verifier-based PAKE. We show, how it is possible to implement AuCPace in its partially augmented variant without any computational overhead in comparison to conventional "balanced" PAKE for the resource-constrained server, while maintaining all of the most relevant security guarantees in the IIoT setting.

Our protocol could be clearly modularised into a balanced sub-protocol that we coined CPace (Composable Pace) since it shares important design features with the PACE protocol from [BFK09] and an augmentation layer allowing for both conventional augmentation and our newly introduced partial augmentation.

As one concrete instantiation we present a protocol AuCPace25519 which adds up further optimisations also on the group arithmetics and field arithmetics level. We present performance results for both, AuCPace and Diffie-Hellman protocols for ARM Cortex M0 and Cortex M4 microcontrollers. We hope that the new speed-records for constant-time implementation of both, PAKE and the X25519 Diffie-Hellman Protocol on the ARM Cortex M4, that we report in this paper will make it possible to enlarge the set of targets that could afford integrating state-of-the-art security technology.

1.5 Organisation of this paper

This paper is organised as follows. In section 2 we first review related work on PAKE protocols and review the definitions of security used for their respective security analysis. Since AuCPace may be considered a SPEKE [Jab96] variant, we will concentrate on this protocol family in more detail. We then give a short introduction to the concepts and methods used for security proofs in the framework of Universal Composability (UC) and describe our proof strategy.

Subsequently, in section 3 we will explain the full protocol AuCPace. There we will also explain the design guidelines that motivated specific choices.

We then present our modular security proof of indistinguishability, first handling the case of the balanced sub-protocol CPace without explicit authentication (section 4). In section 5 we then use the composition theorem for proving security of the verifier-based PAKE protocol in an UC hybrid model. In the subsequent section 6 we expose how to halve the computational complexity of the login process of our protocol when discarding security guarantees of only minor practical relevance in the IIoT setting. For this purpose we introduce the concept of "partial augmentation" with a corresponding ideal functionality $\mathcal{F}_{\text{papwKE}}$. In section 7 we compare our proposal AuCPace with other efficient PAKE protocols from the IIoT perspective.

Subsequently in section 8 we describe the implementation strategy on Cortex M0 and M4 microcontrollers for our reference implementation AuCPace25519. We conclude the paper by presenting actual performance benchmarks on different microcontrollers and embedded bluetooth-transceiver microcontroller platforms in section 9.

In the appendix we present an outline, how a variant of our protocol could also be implemented on conventional elliptic curves in short Weierstrass form and how the very similar balanced protocol in [HL17] could be proven secure in the UC model.

2 Review of PAKE protocols and their security analysis

2.1 Overview on PAKE protocols

Despite the clear-cut security advantages, there are a couple of reasons that hampered use of PAKE protocols in a number of applications [EKSS09]. Among the many reasons patent pitfalls did play a major role. Notably, protocols such as EKE [BM92] and SPEKE [Jab96] were patented until very recently and also some of the most efficient protocols in the IEEE 1363 standard family are patented. This resulted in protocols such as SRP[W$^+$98], J-PAKE [HR10] and PACE [BFK09] which did include additional complexity solely for patent circumvention.

Many protocols, including EKE and SPEKE, have been first suggested without a formal security proof. Doing so comes at the risk of accidentally including serious design-flaws. Many PAKE protocols have later been shown to be insecure. A recent example showing the need of thorough security analysis is the case of zkPAKE presented in [MRA15] that has been shown vulnerable to offline guessing attacks in [BSv17].

As a result of patent circumvention, a large number of different PAKE protocols has been presented. This did hamper thorough security analysis. Firstly proofs became more complex or impracticable because of the additional complexity of patent circumvention steps. Secondly the number of protocol variants grew significantly, reducing the amount of analysis effort spent on each individual variant.

As a general observation, protocols with a security proof that relies on random oracles are more efficient than protocols whose security is based on standard assumptions. Some of the additional complexity may be attributed to technical aspects of the proof strategy. The efficiency difference is significant. For instance, SPEKE [Jab96] requires only two exponentiations for generating the session key, while the KOY protocol [KOY01] requires 15 exponentiations for the server. For a recent comprehensive overview over different PAKE protocols and proof strategies see, e.g. [SOAA15].

Many of the most efficient protocols base security on the conventional computational Diffie-Hellman (CDH) problem or decisional Diffie-Hellman (DDH) problem. When staying in the random-oracle model, this simplifies the security analysis in comparison to other efficient constructions, such as AugPace [SKI10] whose security is based on problems that are not as well understood and studied.

For the scope of this present paper targeting resource-constrained applications, we saw the need to focus on the most-efficient constructions that are not covered by patents. For efficiency and patent reasons, we did concentrate our research on SPEKE and one of its variants, PACE [BFK09]. Since the SPEKE patent has expired recently, some of the circumvention strategies used for PACE became obsolete. The protocol CPace, as presented in this paper, has been developed in the process of removing some of the patent circumvention steps. We did observe that the resulting protocol changes allowed for a natural way of agreeing on a session id before entering the protocol, and opened a path for a proof strategy within the UC framework. We will elaborate on the specific design objectives and corresponding advantages of AuCPace in the section presenting the protocol.

It is worth noting that a very interesting recent and highly efficient construction for PAKE augmentation using oblivious pseudo-random functions OPAQUE has been presented in [JKX18]. OPAQUE is particularly interesting for elliptic curves having an efficient integrated mapping available. We will come back to the respective advantages and disadvantages in of this construction in actual applications in the discussion sections.

2.2 Security models

Probably one of the most widely used security models used for analysing PAKE protocols is the game-based approach introduced in the early work of Bellare, Pointcheval and Rogaway (BPR) [BPR00]. This model later has been extended to the so-called "find-then-guess" (FTG) model[AFP05]. Other groups have used simulation-based proof techniques, such as introduced by Boyko, MacKenzie and Patel (BMP) in [BPR00].

In 2005 Canetti, Katz, Lindell and MacKenzie [CHK+05] have introduced an alternative approach, based on the framework of Universal Composability (UC) [Can01], specifically in its joint-state version [CR03]. Unfortunately, it has been proven that PAKE constructions being secure in in the UC model could not be realised without either, idealised assumptions, such as random oracles or a common reference string [CHK+05]. One of the advantages of analysing 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.

2.3 Review of SPEKE and SPEKE variants

SPEKE [Jab96] is one of the earliest published protocols. Over time it has been analysed in a number of papers. The first security analysis of SPEKE has been given by MacKenzie within the BMP simulation based model [Mac01]. As a result of the analysis at most two passwords may be guessed for one on-line impersonation attack. Later it has been shown that in fact multiple password guessing with one impersonation is feasible [Zha04] when instantiating SPEKE on large-characteristic prime fields (as in the original 1996 SPEKE paper).

SPEKE variants inherit the property from Diffie-Hellman key exchange, that a man-in-the-middle attacker has the possibility to modify both honest party's resulting DH key unless the whole transcript of the communication is used for generating the session key (or authenticator messages respectively) [HS14]. If the attacker replaces both intermediate exponentiations B^a and B^b of the honest participants by a fixed power $(B^a)^c$ and $(B^b)^c$ both parties will calculate the same session key. Note that this "attack" does not actually affect security in practice [HS14]. In fact, preventing it might result in some overhead with respect to storage for the transcript and computation.

SPEKE variants operating on large characteristic prime fields suffer from exponential equivalence of passwords and also hashing of the password as suggested in [Jab97] does not fully resolve this issue [LW15]. Note that the length of the bit stream generated by commonly used hash functions is way shorter than the length of the prime.

In principle, SPEKE could also be implemented using groups on (hyper)-Elliptic curves. However, until very recently [PW17] we are not aware of any security analysis where SPEKE derivates (except for PACE [BFK09]) have been proven to be secure in this setting.

It was long conjectured that SPEKE and its variants inherit the property of forward secrecy from Diffie-Hellman. However we are only aware of one protocol, VTBPEKE, for which this has been analysed in detail [PW17].

One notable advantage of SPEKE and some SPEKE variants such as PACE [BFK09] when implemented on elliptic curves with integrated mapping [CGIP12] is that no full group operations are required, e.g. allowing also for so-called x-coordinate-only implementations. This feature provides some advantages of SPEKE and SPEKE variants in comparison with, e.g., SPAKE [AP] and its variants and PAK[BMP00]. I.e. this way no full group operations are required for implementations but only a less-strict notion of a group modulo negation, that allows for scalar multiplication and differential addition but not for arbitrary point additions.

Verifier-based variants of SPEKE have been suggested already in 1997 [Jab97]. However the first verifier-based SPEKE variant VTBPEKE that came with a formal security proof dates to our best knowledge from 2017 [PW17].

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. The reader interested in more details is referred to the updated version of [Can01] in [Can00].

The general idea of UC is to define security in terms of idealised 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 analysed. 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 $\mathcal{F}_{\mathrm{pwKE}}$ 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 its internal memory. In the literature different types of corruption could be distinguished. Unlike in so-called static corruption models where corruptions are only allowed before starting with an actual protocol execution, the so-called adaptive or Byzantine corruption models 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 model.

In the UC framework there may be many copies of the ideal functionality running in parallel, each one being uniquely characterised by a session id *sid*. Every time a message has to be sent to a specific copy of $\mathcal{F}$, such a message should contain the *sid* of the copy it is intended for. We follow here the approach of [CHK+05] and assume that each protocol that implements $\mathcal{F}$ expects to have the session id included in each message.

The original UC theorem from [Can01] allows to analyse 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 a hash function functionality for instance. For this reason, for our application in mind, 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]. Unfortunately the alternative approach of [CJS14] for modelling a global random oracle $\mathcal{F}_{gRO}$ functionality could not be used, because we need to program the random oracle to chosen values in our proof.

It is worth noting, that the integration of a Diffie-Hellman substep typically provides significant technical difficulties in the UC framework [KR17]. Specifically, when considering adaptive adversary models allowing for corruptions at any time, it is often infeasible for the simulator to provide the adversary secret scalars that are consistent with the previously published group elements. For this reason, many security proofs previously were forced to restrict the analysis to the weaker static adversary model, for instance, in [GMR06].

2.5 Overall proof strategy

In this paper we follow a modular approach and try to avoid the introduction of new security notions and new ideal functionalities where possible.

For this reason, we first prove that the balanced sub-protocol CPace securely implements the ideal functionality $\mathcal{F}_{\mathrm{pwKE}}$ from [CHK+05] which we repeat for reference in figure 4.

We finally show by using the UC composition theorem that the combination of the sub-protocols securely implements $\mathcal{F}_{\mathrm{apwKE}}$ from [GMR06] and thus provides conventional "full" augmentation.

We then show that when the ephemeral Diffie-Hellman secret scalar x within the AuCPace augmentation layer is replaced by a static scalar value $\tilde{x}$ that is re-used over several protocol instances, the computational complexity for the server is essentially halved, while preserving most of the security guarantees. We will present the corresponding idealised functionality $\mathcal{F}_{\text{papwKE}}$ together with the security proof for partially augmented AuCPace.

3 The AuCPace protocol

3.1 Design rationales for the AuCPace protocol

One major guideline for the protocol choices beside the security targets already introduced above has been server-side efficiency. In a typical use case of a remote human-machine-interface for a resource-constrained target, we could assume the client-side units to have much more powerful computing capabilities.

There is no simple means for performance assessment of cryptographic protocols. A large number of competing objectives are involved. The assessment needs, e.g., to consider computational resources, RAM memory requirements, ROM memory requirements, message transport latency, maximum transport-layer packet size and power consumption. For different target hardware the relative importance varies.

Often, PAKE protocols were assessed with respect to performance in a simplified way by counting the number of required exponentiations (sometimes also distinguishing fixed-based and variable base cases) and by counting the number of communication rounds.

With AuCPace, we target the industrial IoT use-case, where a large set of servers is assumed to be operating with the same passwords. For integrating these servers into a larger user-credential distribution framework, it is mandatory to meet the computational constraints also of the smallest devices. Therefore, the design should focus on this specific subset, assuming that larger devices could then implement the corresponding mechanisms without any difficulty.

We assessed power consumption to be one major issue in line with the results of [HL17]. We observed that not only minimising the number of required group exponentiations is important. Beside the choice of a high-speed elliptic curve, a number of further parameters influence the efficiency. Also point compression and point verification substeps should be considered. We observed, e.g., that x-coordinate-only Diffie-Hellman implementations could provide significant advantages, both, regarding speed, ease-of-implementation and memory-consumption. We therefore searched for protocols compatible with implementations not requiring full group operations. One possible advantage of using x-coordinate-only implementations of an elliptic curve is, that the complexity of point compression/decompression and verification could be drastically reduced, specifically on curves offering twist security (allowing for efficient strategies for fending off small-subgroup attacks [LL97]). Otherwise point decompression typically requires one costly field exponentiation for implementing a square root.

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

For AuCPace, we also aimed at minimising memory requirements. E.g., it was considered desirable, not to require that a full transcript of all communication is to be kept in memory until the termination of the protocol. Throughout the calculation, we aimed also at minimising the memory requirement for temporary variables. E.g. for calculating scalar multiplications on elliptic curves we searched for efficient strategies not requiring large pre-computed tables, such as are typically required for window-based algorithms. Also storage in Flash memory, e.g. for password database entries was considered important. As one example the Ω-Method from [GMR06] needs to store a full public-private key pair in addition to the hashed password and the salt. We aimed to improve on this in our solution. For constrained servers, we assumed also that the ROM limitation might not allow for easy integration of tables typically needed for fixed-base point scalar multiplication speedups.

On the other hand, we assessed that in comparison to memory and power consumption, the number of message rounds might not be of primary importance, specifically if content of messages from different protocol levels (protocol handshake, authentication protocol, secured channel protocol) could be combined. Typically the main objective of reducing communication rounds is reduction of the impact of networking latency. However, for constrained servers, the main issue for the user-experienced delay might not be packet transmission latency but lacking power or lacking computational resources respectively [HL17].

3.2 Parameters of the AuCPace protocol

The AuCPace protocol is depicted in figures 2 and 3. The protocol is parametrised by

- A password based key derivation function PBKDF_σ that is itself parametrised by algorithm settings σ, specifying e.g. the memory consumption for the password hash or an iteration count. PBKDF_σ calculates a random string from the password pw, a username and a so-called "salt" value. For reasons to detail in the following paragraphs, AuCPace alternatively also considers use of a server-specific so-called "pepper" that never changes for a specific server instead of ephemeral "salt" values that are chosen randomly upon each password change. For our reference implementation of AuCPace, we use the memory-hard script [PJ] password hash parametrised for a memory consumption of 32 MByte.

- A (hyper-)elliptic curve $\mathcal{C}$ with a group $\mathcal{J}$ and a Diffie-Hellman (DH) protocol operating on both, $\mathcal{J}$ and its quadratic twist $\mathcal{J}'$. We denote the DH base point with B. We don't require full group structure but could also instantiate AuCPace if only group operations modulo negation are available. This might 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 the 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 use a simple exponentiation notation (even if, as in the case of X25519, additional co-factor handling and clamping might apply for the scalars to guarantee operations to take place in the appropriate sub-groups).

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

- A verification algorithm that checks whether the order of a point Y within $\mathcal{J}$ or $\mathcal{J}'$ 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 $X25519(x, Y) \neq 0$.

- A $\mathsf{Map2Point}$ operation and its inverse map $\mathsf{Map2Point}^{-1}$. $\mathsf{Map2Point}(s)$ is required to map a string s on a point from a cryptographically large subgroup $\mathcal{J}_m$ of $\mathcal{J}$, 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 $X \in \mathcal{J}_m$ on 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 Hamburg, Bernstein, Krasnova and Lange in [BHKL13] on Curve25519, where the sign of the inverse map result is chosen at random and the

Elligator2 inverse map's result is padded with random bits to yield the required total length l.

- Hash functions $\mathsf{H}_0 \ldots \mathsf{H}_4$. For our reference implementation we use SHA512 where the hash function index is prepended as four-byte word.

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 2. The second sub-protocol is shown in figure 3 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 parametrisations 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. Possibly authorisation information is also transmitted. The server then checks whether the parametrisation σ and the authorisation setting to attribute to the user are acceptable and stores the verifier W in the database together with the salt and the authorisation settings.

While all of the protocols in this paper allow for choosing ephemeral fresh "salt" values upon password changes, in some settings we see reasons for using a fixed "pepper" that is chosen once for each server and otherwise fixed. Firstly, when using an ephemeral salt value, we could not avoid some leakage of information. Specifically, we could not hide whether or not a specific username entry is available in the server's database or whether a user has changed his password. For some applications, this information might provide real-world attackers more advantage than a facilitated offline attack. Note also that an additional or alternative "pepper" could serve as kind of common reference string for honest parties. Specifically, if the "pepper" value required for the password hash is not openly communicated over the vulnerable channel, some attackers might not be able to mount an online attack.

In 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 that the authorisation for writing password file entries has been verified (as well as the authorisations to be attributed to the user). More formally, we do not consider adversaries $\mathcal{A}$ that read or modify messages of this sub-protocol or impersonate parties.

Within the protocol, the password is never given in clear-text to the server. This also implies that the computationally complex PBKDF function is only calculated on the client entity. Note also that by letting the client choose the "salt" value, we provide a path for distributing password verifiers from a centralised user credential server by use of an offline ticket mechanism.

Note that with respect to the generation of the password verifier W our protocol shares some similarities with the AugPake scheme by Shin and Kobara [SKI10] and VTBPEKE by Pointcheval and Wang [PW17]. In all of these constructions a group element $W = B^\xi$ is used as verifier where password-derived key ξ is used as secret exponent for a known base point.

Store password operation for AuCPace

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

Figure 2: Protocol for password configuration.

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

Establishment of the session key sk is realised by a sequence of several steps shown in figure 3. First a subsession id $ssid$ is established. This is done by hashing together client and server-side generated random values s, t.

Secondly, a password related string PRS is established. We refer to this sub-protocol as the AuCPace Augmentation layer. Establishing PRS involves one message round. After learning the user name, the server transmits its "salt" string together with the value σ required for parametrising the password hash PBKDF_{σ} and an ephemeral public key P 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 party has to calculate PBKDF_{σ}. 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. Doing so 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. There, first an ephemeral generator G is calculated by use of the $\mathsf{Map2Point}$ algorithm.

Calculation of G involves a "channel identifier" CI which is hashed together with the PRS. In the context of a TCP/IP based communication, the channel identifier CI might be constructed by concatenating unique representations of the server's and client's IP address and TCP port numbers. Incorporating the channel identifier into G allows us to fend off certain types of relay attacks. By incorporating the $ssid$ into the calculation, the generator G is guaranteed to be ephemeral.

After deterimining G the two parties implement a Diffie-Hellman protocol by exchanging two messages Y_a and Y_b and derive a shared Diffie-Hellmann secret point K. Note that it is essential for the receiving party to verify the points Y_a and Y_b to be on a large prime-order subgroup of $\mathcal{J}$ or its twist. Then a first session key $sk1$ is derived from the shared Diffie-Hellman secret K.

As last sub-protocol, optionally explicit authentication is added by exchange of two authenticator messages T_a and T_b. Finally the session key $sk1$ is refreshed to yield the final session key sk.

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 $\in \mathcal{J}$.

AuCPace

Server		**Client**
	Agree on $ssid$	
$s \leftarrow_{\$} \{0,1\}^{k_1}$		$t \leftarrow_{\$} \{0,1\}^{k_1}$
	$\xrightarrow{\quad s \quad}$	
	$\xleftarrow{\quad t \quad}$	
$ssid = \mathsf{H}_0(s\|\|t)$		$ssid = \mathsf{H}_0(s\|\|t)$
	AuCPace Augmentation layer	
do until $X \neq 0$:		
$\quad x \leftarrow_{\$} \{0,1\}^{k_2}, X = B^x$		
	$\xleftarrow{\quad \text{user} \quad}$	
$W = \text{lookup}W(\text{user})$		
	$\xrightarrow{\mathcal{J}, X, \text{salt}, \sigma}$	
		$w = \mathsf{PBKDF}_\sigma(pw, \text{user}, \text{salt})$
if lookup failed $PRS \leftarrow_{\$} \{0,1\}^{k_2}$,		abort if X invalid
else $PRS = W^x$		$PRS = X^w$
	CPace substep	
$g' = \mathsf{H}_1(ssid\|\|PRS\|\|CI)$		$g' = \mathsf{H}_1(ssid\|\|PRS\|\|CI)$
$G = \mathsf{Map2Point}(g')$		$G = \mathsf{Map2Point}(g')$
do until $Y_a \neq 0$:		do until $Y_b \neq 0$:
$\quad y_a \leftarrow_{\$} \{0,1\}^{k_2}, Y_a = G^{y_a}$		$\quad y_b \leftarrow_{\$} \{0,1\}^{k_2}, Y_b = G^{y_b}$
	$\xrightarrow{\quad Y_a \quad}$	
	$\xleftarrow{\quad Y_b \quad}$	
$K = Y_b^{y_a}$		$K = Y_a^{y_b}$
abort if Y_b invalid		abort if Y_a invalid
$sk_1 = \mathsf{H}_2(ssid\|\|K)$		$sk_1 = \mathsf{H}_2(ssid\|\|K)$
	Explicit mutual authentication	
$T_a = \mathsf{H}_3(ssid\|\|sk1)$		$T_a = \mathsf{H}_3(ssid\|\|sk1)$
$T_b = \mathsf{H}_4(ssid\|\|sk1)$		$T_b = \mathsf{H}_4(ssid\|\|sk1)$
	$\xleftarrow{\quad T_b \quad}$	
	$\xrightarrow{\quad T_a \quad}$	
verify T_b		verify T_a
$sk = \mathsf{H}_5(ssid\|\|sk1)$		$sk = \mathsf{H}_5(ssid\|\|sk1)$

Figure 3: Protocol AuCPace.

Instead we impose a less strict requirement that could be implemented more efficiently on some curves, notably if they have secure twists: We require that it has to be verified that the order of the respective points is large with respect the required complexity assumption for the CDH problem for the security parameter k, such as to prevent collisions in the result points K and X^w due to small subgroup attacks.

4 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 3.

In this proof we show that this sub-protocol emulates the functionality $\mathcal{F}_{\mathrm{pwKE}}$ from [CHK+05]. $\mathcal{F}_{\mathrm{pwKE}}$ (repeated in figure 4) 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 the passwords match. In case that any party gets corrupted or messages were modified by the adversary, the adversary is given control over the party's session key sk.

In [CHK+05] $\mathcal{F}_{\mathrm{pwKE}}$ has been used in the context of static adversaries only. Here, in the context of adaptive adversaries we observed the need to more clearly specify the behaviour of $\mathcal{F}_{\mathrm{pwKE}}$ in case of adaptive corruptions. As in the static corruption model used in [CHK+05], we let $\mathcal{F}_{\mathrm{pwKE}}$ give party P_i's password to $\mathcal{S}$ upon corruption. In addition, we need to handle the case that $\mathcal{S}$ corrupts a party P_j after a session key sk has been already sent to party P_i. If P_i and P_j use the same password pw, we send $\mathcal{S}$ the honest party's session key sk, otherwise (in case of different passwords) we send $\mathcal{S}$ a randomly sampled key sk'. This is required for giving $\mathcal{S}$ the possibility to continue behaving just as a honest party.

The sub-protocol CPace from figure 3 receives password-related strings PRS as input and returns session keys $sk1$ as result. Note that PRS is formed as the concatenation of the subsession id and a password-related component (W^x and X^w respectively).

For the security analysis in this section we need to map the inputs and outputs of the protocol from figure 3 to the notation used for defining the ideal functionality $\mathcal{F}_{\mathrm{pwKE}}$ in figure 4. The password related components (W^x and X^w respectively) correspond to the passwords pw and the resulting session key sk from figure 4 corresponds to the intermediate session key $sk1$ from figure 3. For the purpose of the proof, we define the channel identifier CI to be the concatenation of the identifiers of the parties P_i and P_j. (Note that in order to have both sides work with the same CI, we place the party with the smaller index (i, j) first.)

4.1 Proof strategy

Our proof closely follows the strategy from [ACCP08]. Here we also use a sequence of games G_0 to G_4 where the simulator algorithms $\mathcal{S}_0$ to $\mathcal{S}_4$ are executed. We organise 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 $\mathcal{F}_{\mathrm{RO}}$ 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 $\mathcal{S}_n$.

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 behaviour 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 $\mathcal{F}_{\mathrm{pwKE}}$.

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 $\mathcal{F}_{\mathrm{pwKE}}$.

4.2 Game-based proof

4.2.1 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 5. 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.

4.2.2 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 an own 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 $\mathsf{Map2Point}$, $\mathcal{J}_m$. 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 required to be a large subset of $\mathcal{J}$.

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 guarantees of Map2Point and due to the birthday paradox, G_0 and G_1 are indistinguishable for the environment.

4.2.3 Game G_2 : Handle the case that an impersonating adversary wins by chance

Here we handle the case that an impersonating adversary succeeds in calculating the session key sk without querying the hash oracle H_2. In this case we let $\mathcal{S}_2$ abort. This case occurs with negligible probability. G_2 is, thus, indistinguishable from G_1 .

4.2.4 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 get's 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 $\mathcal{F}_{\text{pwKE}}$. 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 3 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 $(sid, ssid, P_i, y$ unknown $, z_a)$ and $(sid, ssid, P_j, y$ unknown $, z_b)$ in this list together with the respective session id.

Since G from game G_2 was a generator of the whole cryptographic group $\mathcal{J}$, 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.

As soon as both messages Y_a and Y_b have been delivered by $\mathcal{A}$ we calculate the Diffie-Hellman results using the received points and the local exponents z_a and z_b. If the respective result points differ, we know that $\mathcal{A}$ has modified the messages in a destructive way and record for this session a flag DHFails .

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 $\mathsf{H}_2(ssid||K)$. 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 $\mathsf{H}_2(sid||K)$ 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 identifier (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 identifier (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 $K = Y_r^y$. We will calculate the corresponding probability GuessK in section 4.3.

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, thus, have access to the $\mathsf{DHFails}$ marker and 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 $\mathsf{SamePwd}$ query introduced temporarily to $\mathcal{F}_3$. If the passwords match and if the session is not marked as $\mathsf{DHFails}$, 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 recognised destructive modification of the Diffie-Hellman points by the $\mathsf{DHFails}$ marker for the session, we just sample a new value for sk' by the interface of the random oracle $sk' = \mathsf{H}_2(sid||K)$ 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 in a destructive way ($\mathsf{DHFails}$ not recognised), 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 $\mathsf{TestPwd}$ 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 $\mathsf{DHFails}$ is not recognised. For this purpose, we program the value $\mathsf{H}_2(sid||K) := sk$ to the session key returned to the first party. This could fail only, if the oracle H_2 already has been queried for $(sid||K)$, again corresponding to the probability GuessK that we deal with in section 4.3. (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 $\mathcal{F}_{\text{pwKE}}$. 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. Also the session keys are sampled from an un-distinguishable 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 the environment $\mathcal{Z}$.

4.2.5 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 $\mathcal{F}_{\text{pwKE}}$. Note that we need maintain the queries within $\mathcal{F}_4$ that give access to the passwords pw in case of corruptions. We also need to add a query returning the session key delivered to the client in case of late adaptive corruptions of the second party (server), as discussed above. We remove the $\mathsf{SamePwd}$ 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 the environment $\mathcal{Z}$.

4.3 Proof that probability GuessK in G_3 is negligible

For the proof we did use the conjecture that an impersonating adversary that has no access to the password of a honest party *pw* is not able to predict the honest party's calculated point K.

The reasoning is identical for both, server and client, so we consider the case of the server here. 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 without knowing the honest player's password. Note that Y_a is publicly known and may be either on J or on it's twist J'. We know, however, that the order of the point Y_a is large, thus DLP and CDH could be conjectured to be difficult.

In G_2 and under the assumption of the hardness of the discrete logarithm problem DLP, $\mathcal{A}$ can derive K iff he knows y_b. The only information that he disposes of is the remote side's sent point $Y_b = G^{y_b}$. If we assume that the password-derived generator $G = B^g$ used by the honest party is unknown because the password is unknown, the problem of calculating the honest party's secret ephemeral scalar y_b from the known $Y_b = B^{g \times y_b}$ is simply undefined, since one equation cannot be used for two unknowns (g and y_b). Note that the secret scalar g is known to exist, but known to no party according to the security guarantees required for the Map2Point primitive. I.e. even an attacker being able to solve the discrete log problem could not solve it. If one more conservatively assumes that the adversary managed to control the generator G, possibly by biasing the *sid* and by exploiting two simultaneous flaws in both, the hash and the Map2Point primitive, we obtain the PACE-DH problem from [BFK09]. In the very same paper this one is shown to be as hard as the DH problem in the generic model of Shoup [Sho97] and conjectured to be as hard as the computational Diffie-Hellman problem.

In any case the probability GuessK is, thus, negligible.

4.4 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. The ordering of the message exchange of Y_a and Y_b is, thus, irrelevant for the security. Note that this could be used for speedups for actual implementations, specifically in case that the scalar multiplication takes comparable time as message delivery.

5 Proof for the augmented protocol AuCPace

5.1 Technical details

For implementing the actual proofs we need to consider a number of technical details. We aim at re-using functionalities from previous papers wherever possible, specifically the ideal functionality $\mathcal{F}_{\text{apwKE}}$ from [GMR06].

However, for our protocol we could not use it as-is because firstly, $\mathcal{F}_{\text{apwKE}}$ aborts in case that the server does not find a password entry in its file. In our protocol, we aim at keeping the information which users have a database entry somewhat more confidential by continuing the protocol with a random string. This confidentiality could be fully realised only when using a server-specific "pepper". If a random "salt" is transmitted for the password hashing, we could, e.g., not hide the information that a user has changed the password. On the other hand, by using a random string, we could hide the information whether an entry is available in the database at least for the low-motivation and low-skill attacker. Secondly, AuCPace supports password changes, while $\mathcal{F}_{\text{apwKE}}$ only allows for

The functionality $\mathcal{F}_{\text{pwKE}}$ is parametrised 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,role) 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 (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: 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 (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 4: Ideal functionality $\mathcal{F}_{\text{pwKE}}$ from [CHK+05] re-presenting balanced PAKE without explicit authentication.

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

The functionality $\mathcal{F}_{\text{apwKE}}$ is parametrised 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 **:**
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".

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 (TestPwd ,$sid, ssid, P, pw'$) **from adversary** $\mathcal{S}$ **:**
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 (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 and Authentication

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 $key'' = key'$, else pick a random key key'' of length k. Output $(sid, ssid, key'')$ to P.
- In any other case, pick a random key key'' of length k and output $(sid, ssid, key'')$ to P.

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

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

- If this record is fresh , there is a record $(ssid, P', P, pw')$,and $pw' = pw$, let $b' =$ succ .
- In any other case let $b' =$ fail

Send b' to $\mathcal{S}$. If $b' =$ fail, send (abort ,$sid, ssid$) to P, and mark record $(ssid, P, P', pw)$ completed .

Figure 6: Ideal functionality $\mathcal{F}_{\text{apwKE}}$ for verifier-based PAKE with explicit authentication from [GMR06]. Note that we applied a single wording change (underlined) by replacing Impersonate with SvrImpersonate for making it more explicit that this message models impersonation of the *server*.

configuring passwords once for each session id sid. f We first considered re-phrasing the functionality to our needs, but finally refrained from doing so. E.g. regarding the mitigated information leakage when transmitting the "salt" would have added significant complexity. We came to the conclusion that it is best to try to avoid this complexity by rather slightly modifying our protocol for the purpose of the security proof. Specifically, for the purpose of the proof we don't continue the protocol if no password database entry is available for the given username by a random password related string PRS, but abort instead and allow for setting the password only once (in line with [GMR06, JKX18]). This change in the protocol allows for carrying out the proof with an un-modified functionality $\mathcal{F}_{\text{apwKE}}$.

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

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 authorisations. Instead we assume that the client's identifier P_i takes over the role of the user name and ignore the concept of authorisation here. The full protocol used for the proof is shown in figure 7.

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

5.2 Proof strategy for the augmented protocol

With respect to simulation, we need to distinguish password storage and session establishment. During password storage we don't actually give the adversary $\mathcal{A}$ any power. We allow $\mathcal{A}$ for compromising the server after configuration of the password. For this reason, the simulation of this sub-step does not provide any difficulty. We let $\mathcal{S}$ just forward the StorePWfile query to the $\mathcal{F}_{\text{apwKE}}$ functionality and send an empty StorePWfileSvr message to the server.

With respect to the session establishment, we again consider fully-adaptive adversaries. The most complex part of the proof will be handling of compromise of the server database. Just as for the proof of the balanced sub-protocol CPace, we proceed by using a sequence of games where G_0 corresponds to the real world and G_4 corresponds to the ideal world. In each of these games, we consider simulators $\mathcal{S}_0$ to $\mathcal{S}_4$ which implement part of their functionality in a subroutine library $\mathcal{F}_0$ to $\mathcal{F}_4$, where $\mathcal{F}_4$ exactly implements the ideal functionality $\mathcal{F}_{\text{apwKE}}$. Throughout this proof, we show that all of the individual games are indistinguishable for $\mathcal{Z}$.

5.3 Game-based proof

5.3.1 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 5 for calculating the password hash PBKDF. Honest parties P_i execute the actions of the real-world protocol until eventually getting corrupted. 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. The subroutine library $\mathcal{F}_0$ is empty.

The Asymmetric AuCPace protocol

Setup: This protocol uses a random oracle functionality $\mathcal{F}_{\text{RO}}$ for all of the hash functions H_3, H_4, H_5 and the password hash PBKDF_σ (H_6) with a parametrisation σ and salt size of m_s bits. The protocol also uses a balanced PAKE functionality $\mathcal{F}_{\text{pwKE}}$ 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 $\mathcal{J}\,'$.

Password storage protocol:
When P_i (who is a client) is activated using 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 $w = \mathsf{H}_6(\text{salt}||\sigma||pw||P_i)$ by using $\mathcal{F}_{\text{RO}}$. He then calculates a Diffie-Hellman point $W = B^w$.
He sends a message (StorePWfileSvr $, sid, P_i, \text{salt}, \sigma, W$) to P_j. When P_j which is a server receives (StorePWfileSvr $, sid, P_i, \text{salt}, \sigma, W$) from P_i for the first time, he sets file $[sid] = (sid, \text{salt}, \sigma, W, P_i)$.

Protocol steps for session establishment:

1. When P_j receives input (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 (username $, sid, ssid, P_i$) message.
2. When P_i receives input (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 (username $, sid, ssid, P_i$) to P_j and awaits a response.
3. When P_j receives input (username $, sid, ssid, P_i$), he obtains the tuple stored in file $[sid]$ (aborting and marking the session record as completed if this value is not properly defined). He then samples a fresh nonzero exponent x with $0 < x < m_\mathcal{J}$ and calculates $X = B^x$. P_j then sends (hashingParams $, sid, ssid, \sigma, \text{salt}, X$) to P_i. P_j then calculates W^x. He then sends (NewSession $, (sid, ssid), P_j, P_i, (sid, ssid, W^x)$) to $\mathcal{F}_{\text{pwKE}}$ and awaits a response.
4. When P_i receives input (hashingParams $, sid, ssid, \sigma, \text{salt}, X$) he verifies X and calculates $w = \mathsf{H}_6(\text{salt}||\sigma||pw||P_i)$. He then calculates X^w. He then sends (NewSession $, (sid, ssid), P_i, P_j, (sid, ssid, X^w)$) to $\mathcal{F}_{\text{pwKE}}$ and awaits a response.
5. When P_j receives input $((sid, ssid), sk1)$ he calculates $T_a = \mathsf{H}_3(sk1), T'_b = \mathsf{H}_4(sk1)$ and $sk = \mathsf{H}_5(sk1)$ and adds T_a, T'_b, sk to the session record. He then sends (Authenticator $, sid, ssid, T_a$) to P_i and awaits a response.
6. When P_i receives input $((sid, ssid), sk1)$ he calculates $T'_a = \mathsf{H}_3(sk1), T_b = \mathsf{H}_4(sk1)$ and $sk = \mathsf{H}_5(sk1)$ and adds T'_a, T_b, sk to the session record. Then he sends (Authenticator $, sid, ssid, T_b$) to P_j and outputs $(sid, ssid, sk)$. He then waits for a response.
7. When P_i receives a message (Authenticator $, sid, ssid, T_a$) he compares T'_a with T_a and aborts in case of differences. Else P_i outputs $(sid, ssid, sk)$.
8. When P_j receives a message (Authenticator $, sid, ssid, T_b$) he compares T'_b with T_b and aborts in case of differences. Else P_j outputs $(sid, ssid, sk)$.

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

Figure 7: AuCPace Protocol definition for the proof of indistinguishability.

5.3.2 Game G_1 : Modeling the random oracle for the hash

In G_1 we replace calls to $\mathcal{F}_{\mathrm{RO}}$ by an own 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 due to the birthday paradox.

5.3.3 Game G_2 : Getting rid of the case where the adversary $\mathcal{A}$ wins by chance.

This game is almost as the previous, only we allow the simulator to abort in case that the adversary manages to guess one of the authenticator messages T_a or T_b or the final session key sk without querying the random oracles for $sk1$. This happens with negligible probability, so Game G_1 and G_2 are indistinguishable for the environment $\mathcal{Z}$.

5.3.4 Game G_3 : Handle mutual authentication.

In this game we deal with mutual authentication but still allow the simulator access to the clear-text password pw upon server compromise events. I.e., in this game, we don't give the simulator access to the password but instead pass the password from a StorePWfile and CltSession query to code within $\mathcal{F}_3$ with an implementation according to $\mathcal{F}_{\mathrm{apwKE}}$. Temporarily, we allow $\mathcal{F}_3$ to return the clear-text password upon the StealPWfile query.

This way, the simulator may no longer access the password for the message StorePW-file sent from the client to the server. Since we assume that neither impersonation nor eavesdropping or message modification is feasible for $\mathcal{A}$ in this sub-step, simulation of the message provides no difficulties. We just sample a new random salt value and let the client send a message (StorePWfile , sid,salt$, \sigma, P_j$) with only the hashing parameter but without password verifier W. Since $\mathcal{A}$ is not allowed to eavesdrop this is indistinguishable from game G_2 for the environment $\mathcal{Z}$.

Simulation of the (username , $sid, ssid, P_i$), does only include publicly known information and is simulated as in the real world protocol. The same holds for the server's reply (hashingParams , $sid, ssid, \sigma$,salt$, X$) we sample a fresh random secret scalar x and salt value and calculate the public key in the message as $X = B^x$. Point verification for X may be implemented just as in G_2 .

In case of compromising the server's password file, we have to return password verifiers W in order to maintain indistinguishability with game G_2 . In game G_3 we do so, by retrieving the clear-text password pw from $\mathcal{F}_3$ and by calculating the password verifier as in the original protocol.

For simulating the authenticator messages T_a and T_b we sample two random values and transmit these. Since also in game G_2 these values came from a uniform distribution, the authenticator messages from game G_3 are indistinguishable from G_2 for the environment $\mathcal{Z}$.

After sending the authenticator messages, we call the TestAbort queries of $\mathcal{F}_3$ for both parties and call a NewKey query upon success. In case that the adversary did destructively modify the hashingParams or the authenticator messages, we let protocol parties abort.

Games G_2 and G_3 are indistinguishable for the environment. In both games, the client aborts, if the group order of the point X is small. The Diffie-Hellman points W^x and X^w match, thus, iff W has been calculated from B^w and X has been calculated from B^x. Therefore any modification of X by $\mathcal{A}$ leads to different PRS strings. I.e. the input to $\mathcal{F}_{\mathrm{pwKE}}$ matches iff the passwords used for the StorePWfile request for the client match the one from the CltSession request. As a consequence the session keys returned by

$\mathcal{F}_{\text{pwKE}}$ match only if the very same passwords match. Verification of the authenticators in G_2 succeeds iff $\mathcal{F}_{\text{pwKE}}$ returned the same session key to both parties. Upon any modification of the authenticator messages by $\mathcal{A}$ the parties abort in both games.

5.3.5 Game G_4 : Keeping the password secret

In this game we disallow the simulator to access the clear-text password upon server compromise events. In this game, we add a re-program-offline-query list Λ_1 to the implementation of the random oracle in addition to it's list Λ. We change the implementation as follows. Upon a query q to the hash oracle, if no corresponding entry is found in Λ, we first parse salt value P_i, pw and σ from the query q's encoding. For all entries (sid, P_j, w) contained in Λ_1 we execute a OfflineTestPwd query on $\mathcal{F}_4$ for all P_j in case of a "correct guess" result, we program the record in Λ for the query to the value w from Λ_1, remove the entry (sid, P_j, w) from Λ_1 and return w. If after parsing the full list Λ_1 no "correct guess" result is returned, we sample a fresh random value r', program it to Λ and return r'.

Upon server compromise, we proceed as follows. We first make a StealPWfile query to $\mathcal{F}_4$. Subsequently, we iterate through the PBKDF's random oracle list entries in Λ, parse the stored queries for the client id, salt, σ and the password and execute OfflineTestPwd queries to $\mathcal{F}_4$. In case of a "correct guess" reply, we learned the password pw and can, thus, calculate the password verifier W and send it to the adversary $\mathcal{A}$. Otherwise, the password hash oracle has not yet been queried. In this case, we sample a new random hash result w, setup a new re-program-offline-query entry (sid, P_j, w) for the hash oracle Λ_1. In this case we calculate the password verifier as $W = B^w$ and send it to the adversary.

This procedure allows us to later on arrange for matching password verifiers W and password hashes w.

We also have to handle the case of impersonation. If $\mathcal{A}$ uses the stolen password verifier in his attack strategy for impersonating a server, we let the simulator make calls to SvrImpersonate .

The only difference to game G_3 shows up with respect to the way that the password verifier W is calculated. Irrespectively, whether the adversary had queried the hash oracle before the server compromise operation or after, the simulator always returns (w, W) pairs matching to the respective passwords. Also in both games the distribution of password hashes and verifiers w and W is the same. Game G_3 and G_4 are, thus, indistinguishable for the environment $\mathcal{Z}$.

The real-world protocol AuCPace, thus emulates the ideal functionality $\mathcal{F}_{\text{apwKE}}$ in the $\mathcal{F}_{\text{pwKE}}$, $\mathcal{F}_{\text{RO}}$ hybrid model.

6 Partial augmentation

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

In order to allow for the proof we introduce a new concept for *partial* augmentation of a PAKE protocol. The corresponding functionality is depicted in figure 8. In comparison to $\mathcal{F}_{\text{apwKE}}$ partial augmentation ($\mathcal{F}_{\text{papwKE}}$) 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, which is not possible. 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*. 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 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 $\mathcal{F}_{\text{papwKE}}$ does not actually provide any meaningful advantage in comparison to $\mathcal{F}_{\text{pwKE}}$.

The advantage, however, becomes obvious when considering the IIoT setting with many servers sharing the same user credentials and passwords. 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 $\mathcal{F}_{\text{papwKE}}$ 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 might not be a relevant attack scenario, such that the additional capability of the adversary to impersonate the client on this specific server might 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 $\mathcal{F}_{\text{apwKE}}$ 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 $\mathcal{F}_{\text{papwKE}}$.

Similar security guarantees of $\mathcal{F}_{\text{papwKE}}$ could also be realised 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. In such a setting a central user credential database server is storing password verifiers for all users. (We refer to this other type of server as "the database" in order to avoid confusion with the V-PAKE server entities that receive password verifiers from the database). Note, however 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 parametrisation σ.

Finally, we would like to mention that it might be possible to relax the requirement that a new key pair has to be chosen upon each password configuration within the server's file. In fact it would be desireable to choose a key pair only once and forever for one specific server. In the UC model, however, we presently see the need for new key pairs upon each password configuration in order to avoid common state in concurrent executions. This aspect might be worth to be further analysed, specifically regarding the real-world case of a server implementation being restricted to one single concurrent login instance by resource-constraints.

6.2 Proof

We implement the proof for the partially augmented protocol in the UC hybrid model, just as for the fully augmented variant. However, here leave the black-box model for $\mathcal{A}$ for

The functionality $\mathcal{F}_{\text{papwKE}}$ is an extension to the functionality $\mathcal{F}_{\text{apwKE}}$ from figure 6. It implements all of the $\mathcal{F}_{\text{apwKE}}$ 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 8: Ideal functionality $\mathcal{F}_{\text{papwKE}}$ for partial verifier-based PAKE with explicit authentication.

simplicity.

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 upon server compromise. For this reason, we also have to consider adversaries $\mathcal{A}$ which base their attack strategy on this knowledge. In this case, the adversary is able to calculate the password related string PRS and, thus, impersonate the client. In case of such an attack strategy, we let the simulator use the CltImpersonate query of $\mathcal{F}_{\text{papwKE}}$ in order to make the ideal and real world indistinguishable for the environment $\mathcal{Z}$.

7 Performance assessment of the AuCPace protocol

In this section, we will discuss the properties of AuCPace in comparison to other verifier-based protocols and aim to assess the respective suitability for resource-constrained servers for industrial applications. Regarding suitability a number of aspects is to be considered. In our opinion, all of server-side efficiency, ease-of implementation, code-size, intellectual property rights, flexibility regarding password registration, memory requirements for password-verifier storage, number of communication rounds, message length and security guarantees are important parameters.

Our analysis is based on own literature research and the recent comparison regarding efficiency and proven security guarantees in [PW17]. Since the work of [PW17] one new interesting construction, OPAQUE, has been presented in [JKX18].

Giving the conclusion of our assessment result already at the beginning, we believe that AuCPace overall is best suited for constrained servers in comparison a very small set of other possibly suitable V-PAKE constructions, which is formed essentially only by VTBPEKE [PW17], OPAQUE [JKX18] and augmented versions of PAK [GMR06].

7.1 Security guarantees

Typically the confidential channel used for password verifier registering for a V-PAKE protocol is setup by the V-PAKE protocol itself. For this reason, we consider forward secrecy to be crucial and restrict our analysis here to protocols specifically coming with a corresponding security proof. Unfortunately some otherwise interresting constructions, such as AugPake [SKI10] don't offer this feature.

To some extend, the security guarantees depend on the framework used for the security analysis. E.g. in the UC model approach used here and in [GMR06, JKX18] no assumptions regarding password distribtions apply unlike for with the game-based approach used, e.g., in [PW17]. Also for AuCPace, we did consider fully adaptive adversaries during session establishment, while in the other UC-based approaches [GMR06, JKX18] only static server corruptions were considered. However, according to our assessment in the view of practical relevance for real-world applications all of these differences are of minor importance.

Within the group of augmented protocols, we only identified two specific feature differences. Firstly, the unique feature of OPAQUE is that it allows for starting with the offline attack only after compromising the server (referred to as pre-computation resistance). OPAQUE, thus provides somewhat stronger guarantees than all of the other protocols. Secondly, the unique feature of AuCPace is that it optionally allows for partial augmentation, i.e. a somewhat weaker security guarantee.

7.2 Computational efficiency for servers

With respect to the fully augmented setting we come to the conclusion that OPAQUE allows for the lowest known computational complexity on the server by needing only three scalar multiplications. Moreover one of these scalar multiplications could be pre-computed prior to login sessions and uses a fixed base-point.

According to [PW17] and in line with our own analysis, the most efficient known conventional verifier-based PAKE protocol that allows for forward-secrecy within the analysis [PW17] is VTBPEKE, requiring four exponentiations in total, just as the fully augmented variant of AuCPace. However unlike VTBPEKE, for AuCPace one of these four ($X = B^x$) could be pre-calculated before the login starts. The perceived delay on the HMI interface due to the complex scalar multiplications will correspond, thus, to only three scalar multiplications in the case of AuCPace in contrast to four in the case of VTBPEKE.

In comparison to AuCPace, VTBPEKE and OPAQUE on the one side, UC-secure constructions based on [GMR06] on the other side should be computationally somewhat more complex due to the required digital signature verification substep (specifically when discarding speedup strategies based on large RAM-tables).

For important applications in the IIoT setting, we conjecture that conventional full augmentation is not essential and that suitable security could be obtained when implementing partial augmentation. Specifically here we consider settings where server compromise involves stealing of hardware and highly invasive physical attacks likely to destroy hardware. In the partially augmented setting, AuCPace has a computational complexity of two exponentiations and provides the most efficient solution among all known verifier-based PAKE protocols. Attacks on un-compromised servers are prevented even if these are working with the same password verifier W as, e.g. distributed by a centralised database server.

When considering efficiency to be one of the most important parameters, we concluded that the remaining subset of suitable known augmented protocols beside AuCPace are OPAQUE [JKX18], augmented variants of PAK using [GMR06] and VTBPEKE [PW17].

7.3 Implementation effort

All of these protocols in contrast to AuCPace require the full group structure for actual implementation. I.e. both, X and Y coordinates are necessary and implementations have to deal with the conventional point compression and point verification issues if they aim at reducing the message and/or password verifier length. AuCPace, in contrast could also be implemented by using x-coordinate-only Diffie-Hellman algorithms such as X25519. This could be used for sparing both code-ROM and RAM memories and could facilitate secure (e.g. efficient constant-time) implementation.

Unlike the other considered protocols [GMR06] requires implementation of a digital signature primitive and, thus, needs larger implementation effort. While beside the elliptic-curve operations AuCPace only requires a cryptographic hash, the other protocols also all require a symmetric encryption primitive. AuCPace is, thus, somewhat simpler, however, this is only a very minor advantage, since most application based on PAKE protocols might anyway require symmetric authenticated encryption.

VTBPEKE and constructions based on [GMR06] have the advantage that they could be implemented without requiring hashing to elliptic curve points. Note that in the light of intellectual property rights, this could be a significant advantage, specifically regarding standardized curves in short-Weierstrass form. Note, however, that when discarding the aspect of patents, hashing to elliptic curves could be implemented with little effort since the field arithmetics used for the elliptic curve operations could be re-used.

For VTBPEKE, unlike AuCPace, the server also needs to implement inversions with respect to the group order. I.e. in addition to the field arithmetics a second set of modulo reductions needs to be implemented also for the server.

AuCPace, thus, allows in comparison to the other candidates for significantly improved ease-of-implementation.

7.4 Bandwidth and latency aspects

At a first glance, AuCPace requires a comparably large number of communication rounds, specifically regarding the initial establishment of the session id ($ssid$) by exchange of the messages t and s. The security model of the UC framework assumes that the $ssid$ is fixed prior to initiating the protocol. This requirement should also apply for other UC-secure constructions such as OPAQUE and augmented protocols based on [GMR06].

Note, however, that any reasonable communication protocol needs a mandatory protocol handshake phase and that the message exchanges for generating the $ssid$ could be integrated in this phase for optimisation.

Also the typical use-case of a PAKE protocol is establishment of a secure (encrypted and authenticated) channel using the session key. In this case, the final authenticator messages are optional, just as for OPAQUE. (Note, that in both cases the two explicit authentication messages are not mandatory for UC-securely implementing the $\mathcal{F}_{\text{apwKE}}$ functionality.)

If using explicit mutual authentication, T_a and T_b messages may alternatively be prepended to the first encrypted payload data or "change cipher spec." packet respectively. Also note, that in case that the optional mutual authentication is desired, no specific ordering requirement is imposed on the T_a and T_b messages.

In comparison to AuCPace, OPAQUE and VTBPEKE require one and two messages less, respectively. As a result, message latency will add up more significantly for AuCPace. However, in comparison to OPAQUE and [GMR06], messages used for AuCPace and VTBPEKE are significantly shorter. Specifically, it is not necessary to transfer encrypted (and authenticated) versions of public-private key pairs. Note that this provides an advantage, when using the PAKE protocol over a low-bandwidth wireless link, specifically if very small packets are used on the physical layer, such as characteristic, e.g., for the bluetooth-low-energy standard.

Note also, that the shorter messages allow for reduced buffer sizes and all-over reduced RAM memory requirements.

AuCPace allows for pipelining message transfer and cryptographic calculation, improving upon user-experienced latency. For instance, the server may interleave transmission of the last message from the augmentation layer (X, σ, salt) and calculation of the public point Y_a, such that Y_a is transmitted later in a separate message. In settings where message delivery latency is significant and computation is fast, however, the server may choose to include Y_a in the earlier message.

7.5 Intellectual property rights

We believe that pending patents on algorithms and algorithmic substeps might seriously hamper actual use of a crptographic protocol. AuCPace was specifically designed for avoiding all patents known to the authors. The only aspect where we are aware of the potential of conflicts is the Map2Point substep where efficient algorithms might possibly have to be analyzed in detail, specifically for curves in short Weierstrass form. (For this case, we will sketch a circumvention approach in the appendix.)

Unfortunately, an important part of the efficiency advantage of OPAQUE could be attributed to the use of the highly efficient HMQV [Kra05] construction, for which unfortunately according to the author's knowledge patents apply in some countries. We believe that for patent reasons, important applications might not be considering use of OPAQUE, specifically regarding industrial installations where identically-constructed plants might be constructed in various countries throughout the world.

7.6 Registering verifiers

Regarding password verifier registration on the server we identify two main aspects: Size of the password verifiers on the server and flexibility regarding password-verifier registration protocols.

AuCPace is characterized by requiring only very little persistent storage for the password verifiers. I.e. in addition to the user identifiers, the encoding of the user's authorization and the salt value (which might require roughly 32 bytes), only one (two) group elements requiring typically 32 bytes need to be stored for full (partial) augmentation. In total an amount of 64 (96) bytes sufficies. (Possibly further future analysis, specifically in a game-based BPR model could reduce the verifier size also for partially augmented variants to 64 bytes.)

The other protocols, specifically [GMR06, JKX18] require significantly longer verifiers. E.g. OPAQUE requires two group elements in addition to two secret scalars in an encrypted authenticated strucure (typically requiring additional nonce and mac fields). Even when considering point compression, this could easily add up to a total verifier size of 180 bytes.

Note that the size of password verifiers is important. Some microcontrollers include small amounts (e.g. 1 kByte) of somewhat protected memory (e.g. tamper-protected RAM) meant to be used for storing sensitive information such as cryptographic keys or password-related information. Excessive size of password verifiers might require additional complexity in the application or make implementers be tempted to use conventional unprotected memory also on devices that might be exposed to physical attacks.

Regarding OPAQUE, it is worth to draw attention to a side-effect of the security property of pre-computation attack resistance. This desirable feature is realised by executing the complex password hash after finishing the protocol step of the oblivious pseudo-random function (OPRF). This requires as a consequence to maintain an online connection to the server entity upon password changes. First an interaction is required between client and server for calculating the OPRF which is required as pre-requisite for calculating the computationally complex PBKDF password hash. In other words, for OPAQUE password registration requires at least three messages and bi-directional communication.

Otherwise we would have to let the client entity choose the secret server scalar for the OPRF upon password configuration, which we do not consider ideal from a security perspective (in line with the suggestion of the authors of [JKX18]). As a consequence, if the server is not capable of calculating the PBKDF this effectively precludes user credential distribution protocols based on offline ticket generation allowing for only one single (uni-directional) message.

For this reason, we conclude that OPAQUE might probably not be best-suited for applications, where online connections to the server are not guaranteed to be permanently available. We assess, that in this case offline password database update protocols become helpful, notably for battery-driven devices and devices operating in separated industrial sub-networks that have been intentionally isolated (i.e. no bi-directional communication) for security reasons from the office IT network where the user credential database server is located.

7.7 Summary regarding the protocol assessment

Summing up, we conclude that overall AuCPace provides significant advantages for constrained IIoT servers in comparison to other candidate protocols for V-PAKE for resource-constrained industrial installations.

While we do acknowledge some clear advantages of other candidate protocols, notably OPAQUE, for some isolated sub-aspects, in our opinion AuCPace fares at least reasonably well for all of the metrics discussed above and provides the best overall figure-of-merit.

8 Implementation on ARM Cortex M0 and M4 microcontrollers

One important target platform for resource-constrained (I)IoT devices are 32 bit microcontrollers, such as from the ARM Cortex M0 and Cortex M4 series. We have implemented AuCPace25519 in its partially augmented variant on nRF51 and nRF52 microcontrollers from the company Nordic Semiconductors and three different microcontrollers from the manufacturer ST Microelectronics.

In this section we first describe the high-level strategy for implementing the X25519 Diffie-Hellman protocol and Elligator2. Then we will elaborate on our strategy for implementing the field arithmetics.

8.1 Implementation of X25519

AuCPace uses X25519 for generating the password verifier and the session key $sk1$. We make use of the constant-time Montgomery ladder algorithm from [DHH$^+$15]. Both fixed-point and variable-point scalar multiplication require 1287 field multiplications and 1274 field squarings. We did implement two variants. Firstly for the sake of comparison with related work, we implemented a synchronous version of the X25519 function. Secondly, we implemented a second, asynchronous version of X25519. For this second implementation, we defined an asynchronous cryptographic engine (ACE) object that stores the intermediate state of the scalar multiplication. This allows our implementation to suspend and resume calculations after each ladder step in case that the power budget requires the microcontroller to enter a sleep mode.

We came to the conclusion, in line with findings from [Ham12], that the constant-time Montgomery ladder is, most probably the most efficient known algorithmic choice available for Diffie-Hellman on Curve25519, if we aim at avoiding at the same time memory consuming pre-computed tables.

8.2 Implementation of Elligator2

In order to remain consistent with the notation used in [BHKL13] we denote the Legendre symbol that records quadratic residuosity of a mod q (with q being an odd prime number) with χ:

$$\chi(a) \triangleq \left(\frac{a}{q}\right) \equiv a^{\frac{q-1}{2}} \tag{1}$$

Remember that the Elligator2's decoding function for a Weierstrass curve $E : y^2 = x^3 + Ax^2 + Bx$ is the function $\psi : R \to E(\mathbb{F}_q)$ defined as follows: $\psi(0) = (0, 0)$; if $r \neq 0$

then $\psi(r) = (x, y)$ (see [BHKL13]). For a set R defined as

$$R \triangleq \{r \in \mathbb{F}_q : 1 + ur^2 \neq 0, A^2ur^2 \neq B(1 + ur^2)^2\} \tag{2}$$

the following elements of $\mathbb{F}_q$ are defined (we use only X coordinates):

$$v = \frac{-A}{1 + ur^2} \tag{3}$$

$$\varepsilon = \chi(v^3 + Av^2 + Bv) \tag{4}$$

$$x = \varepsilon v - (1 - \epsilon)\frac{A}{2} \tag{5}$$

In case of Curve25519, $q = 2^{255} - 19$, $A = 486662$ and $B = 1$. We take $u = 2$. If we would calculate x directly, we would need two exponentiations, one for the inversion (3) and one for the Legendre symbol χ (4). We will show that computing a single exponentiation is enough, using the inverse square root algorithm. Let us substitute v in $v^3 + Av^2 + Bv$. We obtain, in a projective representation, the fraction

$$\frac{a}{b} \triangleq \frac{A^3ur^2 + AB(1 + ur^2)^2}{(1 + ur^2)^3} \tag{6}$$

As a property of the Legendre symbol we have:

$$\chi(\frac{a}{b}) = \chi(ab) \tag{7}$$

since $\chi(\frac{a}{b}) \equiv a^{\frac{q-1}{2}} b^{\frac{-(q-1)}{2}} \equiv a^{\frac{q-1}{2}} b^{\frac{q-1}{2}} \equiv \chi(ab)$ with $1 \equiv a^{q-1} \pmod q$ (Fermat's little theorem). Now let's define

$$c \triangleq ab \tag{8}$$

$$d \triangleq 1 + ur^2 \tag{9}$$

$$s \triangleq (cd^2)^{\frac{q-3}{2}} \tag{10}$$

We have $s^2cd^2 = (cd^2)^{q-2} = \frac{1}{cd^2}$ using Fermat's little theorem again. So the inverse is given by:

$$\frac{1}{d} = s^2cd^2cd \tag{11}$$

iff $c \neq 0$ and $d \neq 0$. If $c = 0$ then the point is $(0, 0)$ or ∞. If $d = 0$ the point is ∞ and we return 0 in whatever case. On the other hand we can write $scd^2 = (cd^2)^{\frac{q-1}{2}} \equiv \chi(cd^2)$.Furthermore it holds that by definition of the Legendre symbol:

$$\chi(ab) = \chi(a)\chi(b) \tag{12}$$

and

$$\chi(d^2) = 1 \tag{13}$$

unless $d = 0$. So

$$scd^2 \equiv \chi(cd^2) = \chi(c) = \chi(ab) = \chi(\frac{a}{b}) \tag{14}$$

By placing (14) into (11) we get

$$\frac{1}{d} = \chi(\frac{a}{b})scd \tag{15}$$

This means that we have calculated the Legendre symbol and the inverse (and finally Elligator2) by means of a single exponentiation (10). In case of Elligator2 for Curve25519 the algorithm requires a total of 254 field squarings and 23 field multiplications.

8.3 Implementation of the field arithmetics

The implementations for Cortex M0 and M4 share all of the group arithmetics and high-level algorithms but rely on separate field arithmetics for addition, subtraction, negation, multiplication and squaring.

Our implementation for the Cortex M0 uses the same highly optimised field arithmetics as in [DHH+15, HL17]. Here register pressure and the limited capability of the multiplier engine providing only 32 bits of a result make it beneficial to employ three cascaded Karatsuba stages.

The implementation for the Cortex M4 makes use of a new, yet unpublished optimised implementation. Just as for the M0, we use a packed radix 32 representation. Throughout the implementation we reduce field elements modulo $2^{256} - 38$.

Our implementation on the M4 uses the much more powerful `MULAL` and `MLAAL` instructions allowing for simultaneously multiplying two 32 bit words and accumulating up to two 32 bit words. Due to the significant advantage of using several stages of Karatsuba multiplication for the M0, we have implemented several variants of Karatsuba multiplication also for the Cortex M4. Experiments, however, have shown that here the reduced register pressure (the "upper" registers R8 ... R12 and R14 may be used without restrictions) in addition to the fact that accumulation comes essentially for free made schoolbook multiplication faster when register-allocation is carefully tuned. Based on our experiments, we presume that for carefully optimised code on the M4, Karatsuba techniques might become beneficial again for integer sizes above 512 bits.

In order to optimise the register allocation, we have generated the assembly sources by a code generator handling register allocation and spill register storage on the stack. For the accumulation of intermediate results during the multiplication we also make use of the `MULAL` and `MLAAL` instructions. Note that when one register with the value 1 is available, `MLAAL` allows for implementing three 32 bit additions in one single cycle ($r := a + b + 1 * c$) yielding a 64 bit result. For subtraction (and for squaring), we made use of a specific architectural property of the M4 architecture. There two distinct ways of handling addition carries are possible. In addition to the `MLAAL` -based method above, flag-based add and add-with-carry (`ADDS` ,`ADCS`) and subtract-with-borrow (`SUBS` ,`SBCS`) instructions are available. The multiplication instructions are specified not to modify the addition/subtraction carry flag. We did use this for merging reduction with subtraction of field elements. We first do accumulate both the value to subtract and a multiple of the prime (that stems from reduction) by using multiply-accumulate instructions. Then we use subtract with borrow to simultaneously subtract both results. Note that this will result in remarkable speed differences between addition and subtraction. Addition could be implemented very efficiently by using the powerful multiplication engine. Subtraction is somewhat slower because `SUBS` and `SBCS` have to be used.

For addition and subtraction, we did use the powerful inline assembly capabilities of both, GCC and CLANG, that allowed us to avoid a significant amount of call overhead. In order to avoid operand fetches and stores wherever possible, we also made use of an inline assembly function that merges addition of a first operand with the curve-constant's multiple of a second operand ($r := a + b * 121666$). Addition and subtraction of field elements follows the strategy from [DHH+15] by first processing the most significant word and then merging reduction of the two most significant bits and addition(subtraction) operation for the remaining seven words. I.e. we make use of the available carry bit 255 to obtain an implementation with only one single carry chain.

One additional optimisation strategy was to bundle load and store operations together as much as possible in blocks. This way the pipeline latency on the M4 could be reduced. Isolated load and store operations account for two clock cycles each, while a sequence of n such operations only accounts for $n + 1$ cycles.

Table 1: Sequence of executing the 64 partial products of words $A_i \times B_j$ used for schoolbook multiplication of 256 bit operands.

	A0	A1	A2	A3	A4	A5	A6	A7
B0	1	5	10	15	20	25	28	48
B1	0	6	11	16	21	26	29	31
B2	2	7	12	17	22	27	30	32
B3	3	8	13	18	23	49	50	51
B4	4	9	14	19	24	52	53	54
B5	33	36	39	42	45	55	56	57
B6	34	37	40	43	46	58	59	60
B7	35	38	41	44	47	61	62	63

8.3.1 Field multiplication for ARM Cortex M4

Table 1 depicts the sequence $(0 \ldots 63)$ in which each of the 64 partial products of the schoolbook multiplication of the input operand words A0 ... A7 and B0 ... B7 is executed. Our optimisation of the multiplication strategy does not seem to follow a regular pattern at first sight.

We use this sequence several reasons. Firstly, we observed that keeping as many input operands in registers as possible is equally important as avoiding stack spills of intermediate multiplication results. Secondly, it is worth noting that a multiplication actually costs *less* instructions if two intermediate results are to be accumulated at the same time. If only one intermediate result is to be accumulated, a MULAL instruction has to be used, which typically requires clearing of a scratch register (+1 instruction).

Basically four subblocks may be distinguished. Initially input operands B0 to B4 are cached in registers and multiplied one after the other with input operands A0 to A4. In the process of multiplication increasingly more registers were required for holding intermediate multiplication results. Completed result words that had been fully accumulated were spilled on the stack in order to free registers for more temporary results. Still starting with the multiplication with input A5, the input operand registers B3 and B4 were required as scratch registers for storing temporaries (multiplication steps 25 to 32). Ultimately also B0 had to be discarded. Then, in order to complete words 5 to 6 of the multiplication results (for freeing completed result words by writes to the stack), multiplication of input operands B5 to B7 with A0 to A4 is performed (33 to 47). Subsequently the multiplication result word 7 could be completed after the multiplication of A7 and B0 (Step 48). Finally the multiplications of input words A6 to A7 with B3 to B7 is calculated. Here the values A5 to A7 were cached in registers.

During the multiplication only 8 register spills were necessary for storing the lower-most result words temporarily on the stack. After multiplication the upper 8 result words of the 512 bit multiplication result were reduced within the register set before storing the reduced result back to memory.

8.3.2 Field squaring for ARM Cortex M4

For squarings we again make use of a special property of the Cortex-M4 instruction set which allows for two different types of carry chain. Either the `ADDS` and `ADCS` instructions may be employed (storing the carry bit in the status register) or `MULAL` and `MLAAL` instructions (which do not modify the carry bit). The latter instructions store carries in full registers. We make use of this by using addition instructions for doubling the off-diagonal parts (with the exception of the product term A1*A0), while we make use of integrated multiply-

Table 2: Sequence of executing the partial product words $A_i \times A_j$ used for schoolbook squaring of 256 bit operands .

	A0	A1	A2	A3	A4	A5	A6	A7
A0	1	2						
A1	0	3						
A2	5	6	15					
A3	4	11	12	19				
A4	8	9	16	23	32			
A5	7	13	20	24	27	34		
A6	10	17	21	25	28	30	35	
A7	14	18	22	26	29	31	33	36

accumulate operations everywhere else.

In comparison to multiplication, we were able to hold more input operands in registers. The following table depicts the sequence (0..36) in which the partial products were calculated.

Throughout the calculation we distinguish between off-diagonal multiplication results (which require subsequent doubling) and diagonal multiplication results which were accumulated by use of multiplication instructions. Just as for multiplication, squaring is merged with reduction. This way only 5 register spills to the stack were required for storing intermediate multiplication results.

8.4 Implementation of the hash functions

For the calculation of SHA512 we use the optimised assembly code for the Cortex M0 architecture on both targets. This code fully unrolls the inner loop of the add-rotate-xor algorithm. We also make use of the special instructions for endianness-change. For the Cortex M4, a further speedup would be possible, when exploiting the availability of the "upper" registers.

9 Experimental results

In the following sections we will report on experimental results obtained from several different microcontroller targets, nRF51822, nRF52832, STM32F407, STM32F411 and STM32L476. We decided to include figures for all of these instead of selecting one particular chipset since in the course of our analysis, we observed that for the Cortex M4 architecture a major difficulty arises regarding speed benchmarking. Unlike for smaller architectures we observed that the highly target-specific performance of the flash memory subsystem plays a major role for the actual speed.

We did observe the most remarkable effect for the microcontroller STM32L476 targeting specifically ultra-low-power applications. Note that for ultra-low-power operation a suitable compromise between increased microcontroller speed and increased power consumption due to speculative flash accesses has to be found. We attribute our finding that the cycle counts for the analysed primitives (depending on the power-consumption configuration) could increase by almost 40% when increasing the clock frequency from 16 MHz to 80 MHz mainly to such type of optimisation. Obviously this makes fair speed benchmarking very difficult.

For the high-performance-family devices STM32F411 and STM32F407 from the same manufacturer, we observed that the influence of the clock frequency on performance is still present, but much smaller. Specifically for the STM32F411 almost no speed reduction was observed also at its highest clock frequency. In addition to flash timing issues, it is worth noting that for the STM32F407 device we observed some further dependence on the RAM memory configuration. This device disposes of so-called core-coupled memory (CCM). The timings reported here were obtained when placing the execution stack to CCM. Speed figures were observed to be somewhat faster than when placing the stack in conventional RAM region.

As a result, we conclude that for speed benchmarking for cryptography implementations it is best to compare results obtained at lower clock frequencies. According to our results then some variations between different microcontroller suppliers still do exist, however the resulting values are at least of the same order of magnitude. We suggest, that the STM32F411 as a typical medium-size implementation for IIoT applications might be well suited as kind of reference platform for speed benchmarking.

9.1 Field arithmetics

In table 3 the speed results for the field arithmetics are summarised. Despite the mentioned difficulties regarding benchmarking, we come to the conclusion that our field arithmetics is significantly more efficient than the previously best published results on the Cortex M4 microcontroller in [FA17], specifically regarding multiplication and squaring of field elements.

The speedup obtained for the field arithmetics in comparison to reports from D. Aranha and H. Fujii in [FA17] in our opinion might stem from the following differences. Firstly for multiplication and squaring we did merge multiplication and squaring functions with reductions. This allowed us to hold more operands in registers. Secondly with the realised level of optimisation regarding multiplication and squaring, the performance of addition and subtraction within the X25519 calculations starts becoming important as well. For these simpler operations call overhead becomes significant and use of inline assembly functions highly beneficial.

For the purpose of comparison, we also did add timings for the field $\mathbb{F}_{(2^{127}-1)^2}$ as used in the construction Four$\mathbb{Q}$ in [LLP+18]. Note that our timings for $\mathbb{F}_{(2^{255}-19)}$ are significantly faster despite the fact that the group order is comparable. As a result, we expect that a large fraction of the algorithmic speedup that is made possible by the endomorphisms of Four$\mathbb{Q}$ is lost by less efficient field arithmetics. Note, that the most relevant figure for the speed of Diffie-Hellman on Four$\mathbb{Q}$ is field multiplication, where the difference to our results is particularly large.

9.2 X25519 Diffie-Hellman

In table 4 we summarise the results for the X25519 function for different microcontrollers and different clock frequencies. Our fastest result for X25519 on the M4 executes in as little as 609.779 cycles and is, thus, roughly 3 and 2.5 times faster than the reports in [dG15] (1816351) and [DSS16] (1563852) respectively and also significantly faster than the previously fastest result (907.240 cycles) from [FA17]. It is worth noting that in contrast to [FA17] we did use (in line with [HL17]) constant-time swaps of pointers instead of swapping full field elements. Note that for internal memories of Cortex M4 and M0 access timing is deterministic. When swapping pointers we expect both, more speed and less side-channel leakage. Note however, that our implementation requires (unlike [FA17]) to be run with using internal RAM memory with constant access times. Our implementation optionally also allows for swapping field elements instead of pointers. According to our own measurements the penalty of doing so accounts roughly for additional 50.000 clock cycles.

Table 3: Field arithmetics on different targets at different frequencies f (/MHz). Columns $*A_0$ $(+*A_0)$ contain clock cycles for multiplication with the field constant $A_0 = 121666$ and merged addition and multiplication $x + y * A_0$. Cycle count for the nRF51 target was obtained with the CLANG compiler with compile switch -O2 while for the ST Microelectronics microcontroller we did use GCC 4.9.2 with optimisation setting -O2.

Target	f	$x+y$	$x-y$	$*A_0$	$+*A_0$	x^2	$x*y$	
nRF51822	16	120	147	193	-	998	1478	$\mathbb{F}_{(2^{255}-19)}$, this work
STM32F411	?	73	77	129	-	563	631	$\mathbb{F}_{(2^{255}-19)}$, [DSS16]
MK20DX	72	86	86	76	-	252	276	$\mathbb{F}_{(2^{255}-19)}$, [FA17]
STM32F411	16	55	72	-	58	153	222	$\mathbb{F}_{(2^{255}-19)}$, this work
STM32L476	16	52	65	-	55	153	220	$\mathbb{F}_{(2^{255}-19)}$, this work
STM32L476	80	95	124	-	95	168	237	$\mathbb{F}_{(2^{255}-19)}$, this work
STM32F407	84	86	-	-	-	215	358	$\mathbb{F}_{(2^{127}-1)^2}$ [LLP+18]

Table 4: Speed of X25519 scalar multiplication (Four$\mathbb{Q}$) (on different targets, clock and memory configurations. The timings marked with (p) were obtained with enabled flash pre-fetch engines which increase current consumption.

Target	f / MHz	X25519	
nRF51822	16	3.474.201	this work
STM32F411	?	1.816.351	[dG15]
STM32F411	?	1.563.852	[DSS16]
MK20DX	72	907.240	[FA17]
STM32L476	16, $80^{(p)}$, 80	609.779, 857.002, 971.272	this work
nRF52832	64	634.567	this work
STM32F411	16, $100^{(p)}$, 100	625.347, 625.449, 734.554	this work
STM32F407	16, $168^{(p)}$, 168	625.358, 655.891, 847.048	this work
STM32F407	$84^{(p)}$	560.500 (Four$\mathbb{Q}$)	[LLP+18]

Again we also have added speed benchmarks for Four$\mathbb{Q}$ from [LLP+18] for reference. Note that comparing of the fundamentally different algorithms X25519 and Diffie-Hellman on Four$\mathbb{Q}$ is difficult. For instance, when using the endomorphisms in Four$\mathbb{Q}$ quite large tables in RAM are required (required stack size is unfortunately not reported in [LLP+18]). Also note that the code size is about a factor of three larger than for our X25519 implementation. Despite the fact that our X25519 implementation (ca. 625.500 cycles) is much more adapted to small targets, our observed speed is very competitive in comparison to the reported result for Diffie-Hellman on Four$\mathbb{Q}$ (560.500 cycles including point decompression).

9.3 Partially augmented AuCPace25519

We have implemented AuCPace by using an asynchronous execution engine as suggested in [HL17]. Table 5 summarises the speed results for individual substeps for the Cortex M0 (nRF51822) and different Cortex M4 microcontrollers.

We observed a speedup of roughly a factor of two in comparison to the results of [HL17] regarding the Elligator2 substep. In [HL17] the Elligator2 mapping algorithm was calculated by use of two separate exponentiations for inversion and calculation of the Legendre symbol χ. In our work, we make use of the inverse square root algorithm

Table 5: Cycle counts for the nRF51822 Cortex M0 (STM32F411 Cortex M4) microcontrollers running at 16 MHz for SHA512, Elligator2, a X25519 Montgomery ladder step (LS), Inversion ($1/x$) and a complete partially augmented AuCPace protocol run.

Target	SHA512	Elligator2	LS	$1/x$	AuCPace
nRF51822	21.564	289.276	13.521	258.291	7.345.820
STM32F411	21.130	46.032	3.163	42.590	1.351.381

Table 6: Memory consumption in bytes for asynchronised implementation of AuCPace (ACE) and X25519 for Cortex M0 and M4 microcontrollers. Results were obtained with arm-none-eabi-gcc -O2 (gcc version 4.9.3). RAM consumption is separated in static memory (stack memory) respectively.

Target Target	RAM ACE	ROM ACE	RAM X25519	ROM X25519	
Cortex-M0	264 (396)	11252	0 (572)	6108	this work
Cortex-M4	264 (268)	8896	0 (444)	3324	this work
Cortex-M4				4152	[FA17]
Cortex-M4				3786	[DSS16]

for calculating Elligator2 with one single field exponentiation. In total this accounts for roughly 4 percent of a speedup regarding the balanced PACE (CPace) protocol runs on the Cortex M0.

Our results for the Cortex-M4 microcontroller family are faster by a factor of 5.4 in comparison to the Cortex M0, showing that this microcontroller architecture with its signal-processing instructions is by far better suited and likely also much more power-efficient for implementing complex asymmetric cryptography.

In table 6 the memory consumptions for the asynchronous execution object ACE from [HL17] and the stand-alone algorithm X25519 are summarised. The figures for the ACE object also include a salsa20-20 based pseudo-random-number generator and the implementation for SHA512. For the Cortex M4 version, the total RAM requirement amounts to 532 bytes including static memory and stack. The stand-alone synchronous X25519 implementation (no state in static memory) for the Cortex M4 needs 444 bytes of stack memory and 3.324 bytes of flash and improves, thus upon previous work [DSS16, FA17].

All of our code avoids secret-dependent branches and is, thus, executing in constant time on target-platforms with deterministic RAM memory access timings, such as typically found in ARM Cortex M0 and M4 microcontrollers.

10 Discussion and conclusion

In this paper we have presented a comprehensive analysis regarding possible optimisations for verifier-based password-authenticated key exchange for the setting of resource-constrained servers. Our analysis did cover all of, protocol design, protocol security proof, algorithmic optimisation regarding group operations and field arithmetics and assembly-level fine-tunings.

Our construction allows for particular advantages in IIoT settings where a large number of small server nodes should be expected to operate with the same passwords, such as e.g. the case in industrial plants. In addition to the conventional notion of verifier-based PAKE, our construction also allows for a partial augmentation operation mode that essentially halves the computational complexity of the password verification step.

Our construction with full augmentation imposes a complexity of four exponentiations in total on the server, one of which could be pre-computed prior to each login. The user-perceived login delay, thus is governed by the time consumed for calculating three scalar multiplications.

Our construction, is one exponentiation faster for the server than all previously known verifier-based PAKE constructions when instantiated in its partially augmented variant. In the setting of [HL17] where one scalar multiplication accounts for about two seconds this results in a clearly perceivable usability gain in comparison to previously known protocols requiring at least three scalar multiplications.

Moreover our construction inherently allows for using strong memory-hard password hashing also on small servers since the costly memory-consuming operations are deferred to the clients.

The composability of the AuCPace security guarantees facilitates security analysis for use of AuCPace, e.g. as a building block in larger constructions, such as a centralised ticket-based user-credential distribution framework for industrial plants.

In contrast to most previous Diffie-Hellman based V-PAKE constructions with analysis in the UC framework, our security proof provides guarantees also in the stronger fully adaptive adversary model which allows for corruptions at any time during session establishment.

Finally, we have presented performance benchmarks of an instantiation targeting common microcontroller platforms coined AuCPace25519 which instantiates our protocol with using the primitives X25519, Elligator2 and SHA512.

The protocol runs in only 1.351.381 (7.345.820) cycles for a partially augmented protocol run on an ARM Cortex-M4 (M0) microcontroller respectively. On the M4 AuCPace requires only 8896 (532) bytes of flash (RAM) memory. There the X25519 Diffie-Hellman protocol sub-step executes in as little as 609.779 cycles. Our implementation, thus, sets up new speed records for both, (V)-PAKE protocols and X25519 Diffie-Hellman key exchange on this important embedded CPU architecture platform. This illustrates also that on the Cortex M4 X25519 could be implemented very competitively, even in comparison to constructions that exploit additional structure in elliptic curves, such as endomorphisms.

Summing up, we believe that all of the individual components presented in this paper in combination might yield a solution particularly tailored for the needs of real-world resource-constrained IIoT environments, such as notably explosion protected industrial instrumentation.

11 Acknowledgements

The authors acknowledge inspiring discussions with Daniel Rausch, Ralf Küsters, Denis Kügler, Marc Fischlin, Mike Hamburg and Peter Schwabe.

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*, pages 335–351. Springer, 2008. 8, 14

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In *International Workshop on Public Key Cryptography*, pages 65–84. Springer, 2005. 6

[AP] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA*, volume 3376, pages 191–208. Springer. 7

[BDKJ16] Alex Biryukov, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. The memory-hard argon2 password hash and proof-of-work function. Technical report, Internet-Draft draft-irtf-cfrg-argon2-00, Internet Engineering Task Force, 2016. Work in Progress, 2016. 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. 10

[Ber14] Daniel J. Bernstein. 25519 naming. Posting to the CFRG mailing list, 2014. https://www.ietf.org/mail-archive/web/cfrg/current/msg04996.html. 10

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the pace key-agreement protocol. In *ISC*, volume 5735, pages 33–48. Springer, 2009. 3, 5, 6, 7, 18, 43

[BHKL13] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980. ACM, 2013. 10, 30

[BM92] Steven M Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 72–84. IEEE, 1992. 3, 5

[BMP00] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology—Eurocrypt 2000*, pages 156–171. Springer, 2000. 3, 7

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology—EUROCRYPT 2000*, pages 139–155. Springer, 2000. 6

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993. 4

[BSv17] José Becerra, Petra Sala, and Marjan Škrobot. An offline dictionary attack against zkpake protocol. Cryptology ePrint Archive, Report 2017/961, 2017. https://eprint.iacr.org/2017/961. 5

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067. 7

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001. 4, 6, 7, 8

[CGIP12] Jean-Sébastien Coron, Aline Gouget, Thomas Icart, and Pascal Paillier. Supplemental access control (pace v2): security analysis of pace integrated mapping. In *Cryptography and Security: From Theory to Applications*, pages 207–232. Springer, 2012. 7

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Phil MacKenzie. Universally composable password-based key exchange. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 404–421. Springer, 2005. 3, 6, 8, 14, 19

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical uc security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 597–608. ACM, 2014. 8

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Crypto*, volume 2729, pages 265–281. Springer, 2003. 4, 6, 8

[dG15] Wouter de Groot. *A Performance Study of X25519 on Cortex-M3 and M4*. PhD thesis, Master thesis, Eindhoven University of Technology (Sep 2015), 2015. 36

[DHH+15] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. High-speed curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers. *Designs, Codes and Cryptography*, 77(2-3):493–514, 2015. 30, 32

[DSS16] Fabrizio De Santis and Georg Sigl. Towards side-channel protected x25519 on arm cortex-m4 processors. In *SPEED-B Software performance enhancement for encryption and decryption, and benchmarking*, 2016. 35, 36, 37

[EKSS09] John Engler, Chris Karlof, Elaine Shi, and Dawn Song. Is it too late for pake? *indicators*, 5(9):17, 2009. 5

[FA17] Hayato Fujii and Diego F Aranha. Curve25519 for the cortex-m4 and beyond. *Progress in Cryptology-LATINCRYPT*, 2017. 35, 36, 37

[GMR06] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. *Advances in Cryptology-CRYPTO 2006*, pages 142–159, 2006. 3, 4, 8, 9, 18, 20, 21, 26, 27, 28, 29, 43

[Ham12] Mike Hamburg. Fast and compact elliptic-curve cryptography. volume 2012, page 309, 2012. 30

[HL17] Björn Haase and Benoît Labrique. Making password authenticated key exchange suitable for resource-constrained industrial control devices. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 346–364. Springer, 2017. 2, 3, 4, 5, 9, 10, 32, 36, 37, 38, 44

[HR10] Feng Hao and Peter Ryan. J-pake: authenticated key exchange without pki. In *Transactions on computational science XI*, pages 192–206. Springer, 2010. 5

[HS14] Feng Hao and Siamak F Shahandashti. The speke protocol revisited. In *International Conference on Research in Security Standardisation*, pages 26–38. Springer, 2014. 7

[Jab96] David P Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996. 3, 5, 6, 7

[Jab97] David P Jablon. Extended password key exchange protocols immune to dictionary attack. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1997. Proceedings., Sixth IEEE Workshops on*, pages 248–255. IEEE, 1997. 3, 7

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: An asymmetric pake protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163, 2018. https://eprint.iacr.org/2018/163. 2, 3, 6, 21, 26, 27, 29, 43

[KM16] Franziskus Kiefer and Mark Manulis. Blind password registration for verifier-based pake. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pages 39–48. ACM, 2016. 4

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 475–494. Springer, 2001. 6

[KR17] Ralf Küsters and Daniel Rausch. A framework for universally composable diffie-hellman key exchange. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 881–900. IEEE, 2017. 8

[Kra05] Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *Annual International Cryptology Conference*, pages 546–566. Springer, 2005. 29

[LL97] Chae Hoon Lim and Pil Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Annual International Cryptology Conference*, pages 249–263. Springer, 1997. 9

[LLP+18] Zhe Liu, Patrick Longa, Geovandro Pereira, Oscar Reparaz, and Hwajeong Seo. Fourq on embedded devices with strong countermeasures against side-channel attacks. *IEEE Transactions on Dependable and Secure Computing*, 2018. 35, 36

[LW15] Hanwook Lee and Dongho Won. Prevention of exponential equivalence in simple password exponential key exchange (speke). *Symmetry*, 7(3):1587–1594, 2015. 7

[Mac01] Philip MacKenzie. On the security of the speke password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. https://eprint.iacr.org/2001/057. 7

[MRA15] Karina Mochetti, Amanda C Davi Resende, and Diego F Aranha. zkpake: A simple augmented pake protocol. In *Brazilian Symposium on Information and Computational Systems Security (SBSeg)*, 2015. 5

[PJ] C Percival and S Josefsson. The scrypt password-based key derivation function. 2012. *URL http://tools. ietf. org/html/josefsson-scrypt-kdf-00. txt*. 2, 10

[PW17] David Pointcheval and Guilin Wang. Vtbpeke: Verifier-based two-basis password exponential key exchange. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 301–312. ACM, 2017. 2, 3, 7, 11, 26, 27, 43

[RS17] Joost Renes and Benjamin Smith. qdsa: Small and secure digital signatures with curve-based diffie–hellman key pairs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 273–302. Springer, 2017. 10

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 256–266. Springer, 1997. 18

[SKI10] SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Security proof of augpake. *IACR Cryptology ePrint Archive*, 2010:334, 2010. 3, 6, 11, 26

[SOAA15] Stanislav V. Smyshlyaev, Igor B. Oshkin, Evgeniy K. Alekseev, and Liliya R. Ahmetzyanova. On the security of one password authenticated key exchange protocol. Cryptology ePrint Archive, Report 2015/1237, 2015. `https://eprint.iacr.org/2015/1237`. 6

[W+98] Thomas D Wu et al. The secure remote password protocol. In *NDSS*, volume 98, pages 97–111, 1998. 3, 5

[Zha04] Muxiang Zhang. Analysis of the speke password-authenticated key exchange protocol. *IEEE Communications Letters*, 8(1):63–65, 2004. 7

A Notes regarding short Weierstrass curves

Our construction shares with [JKX18] the requirement that an efficient hashing to group elements must be available for the elliptic curve's point group. Unfortunately, this is not always the case for important established curves, namely regarding standards using the short Weierstrass form. In order to circumvent this problem, as an alternative in [PW17] a construction TBPEKE based on two base points and an additional scalar multiplication has been suggested by Pointcheval and Wang. Note that this construction is very similar to the balanced sub-protocol CPace presented in this paper.

In this appendix we deal with the question, whether the TBPEKE construction could also be used instead of CPace as a balanced sub-protocol component for AuCPace. I.e. the question is whether the TBPEKE construction could also be proven secure in the UC model. In our opinion, this answer could be given affirmatively. However, unfortunately, our UC security proof technique that allowed for *fully adaptive* adversaries could probably not be carried out for TBPEKE because of a technical commitment problem within the Diffie-Hellman step. However, we come to the conclusion, that the balanced sub-step of TBPEKE could be proven secure also in the UC framework, when considering a weaker *static* adversary model as used for most other security proofs of efficient constructions in the UC framework such as [GMR06].

I.e. for implementations forced to use older short Weierstrass curves, we suggest to replace our technique for the calculation of the ephemeral generator G as $G = \mathsf{Map2Point}(\mathsf{H}_1(PRS))$ by the TBPEKE equivalent of $G = A + C^{\mathsf{H}_1(PRS)}$. The essential property (as pointed out also in [BFK09]) is that the discrete logarithm of G must be unknown for both, honest parties and the adversaries.

Note that for any TBPEKE-based construction we see as important pre-requisite that the "nothing-upon-my-sleeve" problem related to the secrecy of the discrete log of the points C and A needs to be resolved in a trustworthy way.

Here we make the following suggestion. For any of the older short Weierstrass form elliptic curves we suggest to determine the curve points A and C by the following algorithm. For the point A (C) we suggest to first take the packed little-endian encoding of the standardised curve's base point x (y) coordinate and calculate $\tilde{x}_A = \mathrm{SHA512}(x)$ ($\tilde{x}_C = \mathrm{SHA512}(y)$). When doing so, there is a non-negligible probability that the x-coordinates $\tilde{x}_A$ and $\tilde{x}_C$ actually correspond to the x-coordinate of a point on the twist or possibly on a small subgroup. In this case we suggest to increment the coordinates step by step by one until a point on the cryptographic group is returned. We then suggest to choose the one out of two y-coordinate candidates y_A and y_C such that the least-significant bit 0 of the y-coordinate is zero.

Based on the assumption that no common mathematical structure is shared between the respective short Weierstrass curve and the Add-Rotate-XOR (ARX) algorithm SHA512, we conclude that it is justified to conjecture the secrecy of the discrete logarithms of A and C.

B Notes regarding UC security of the PACE protocol variant from [HL17]

The protocol in [HL17] is closely related to the protocol CPace presented in this paper. This protocol and the specific optimisation steps were yet not analysed within the UC framework. The main difference to CPace stems from the strategy used for circumventing the patents on SPEKE.

While we do not detail a full UC security proof for this protocol, we never the less would like to sketch the necessary steps for executing it. In the notation of [HL17] the password pw corresponds to a password-based key π generated, e.g. by hashing the password. The difference of [HL17] to CPace essentially is that the calculation of the password related string PRS involves an additional symmetric encryption, not actually needed for securely implementing the protocol. In order to cover the patent circumvention protocol, we suggest to proceed as follows:

- Firstly we use the encrypted version of the messages s together with the nonce value and the message t that are exchanged at the beginning of the protocol for deriving the session id needed for the UC framework.
- We then use the symmetric salsa20-20 primitive on the password in order to generate the XOR pad used in [HL17] and modify the definition of the password-related string PRS according to the patent circumvention construction $s||t$.
- In order to fend off relay attacks, it will be mandatory to incorporate identifiers for the parties (corresponding to the CI of CPace) into the input to the Map2Point function ($\mathsf{H}(s||t)$) such that not only the password is authenticated but also the client and server identities. This could e.g. be done by incorporating a channel identifier component into input parameter π (the password-derived key) used in [HL17].
- We then prove that the password dependent string $(s||t)$ generated this way matches iff the same password parameter π was used by both, server and client, ensuring that the password and the identities match. For this step, we essentially need the property that the entropy of π is preserved when extracting a random stream from π and nonce value by using salsa20-20.
- The rest of the proof could then be executed by the same procedures as used in this paper. (Note that the explicit authentication step involving generation of several authenticators by one single run of SHA512 is not mandatory for securely implementing $\mathcal{F}_{\mathrm{pwKE}}$.)