# Security Analysis of CPace

Michel Abdalla[1,2], Björn Haase[3], and Julia Hesse[4,⋆]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@gmail.com
[2] DFINITY, Zürich, Switzerland
[3] Endress+Hauser Liquid Analysis
bjoern.haase@endress.com
[4] IBM Research Europe, Zürich, Switzerland
jhs@zurich.ibm.com

**Abstract.** In response to standardization requests regarding password-authenticated key exchange (PAKE) protocols, the IRTF working group CFRG has setup a PAKE selection process in 2019, which led to the selection of the CPace protocol in the balanced setting, in which parties share a common password. In subsequent standardization efforts, the CPace protocol further developed, yielding a protocol family whose actual security guarantees in practical settings are not well understood. In this paper, we provide a comprehensive security analysis of CPace in the universal composability framework. Our analysis is *realistic* in the sense that it captures adaptive corruptions and refrains from modeling CPace's Map2Pt function that maps field elements to curve points as an idealized function. In order to extend our proofs to different CPace variants optimized for specific elliptic-curve ecosystems, we employ a new approach which represents the assumptions required by the proof as libraries accessed by a simulator. By allowing for the modular replacement of assumptions used in the proof, this new approach avoids a repeated analysis of unchanged protocol parts and lets us efficiently analyze the security guarantees of all the different CPace variants. As a result of our analysis, all of the investigated practical CPace variants enjoy adaptive UC security.

## 1 Introduction

Security analysis and efficient implementation of cryptographic protocols are often split into separate working groups. As a result, subtle differences between the actually implemented and analyzed protocols easily emerge, for example when implementors slightly tweak the protocol to improve efficiency. An example where particularly aggressive optimizations for efficiency are implemented on the protocol level is CPace as specified in current internet drafts [23, 24]. CPace is a password-authenticated key exchange protocol (PAKE) [8], which

⋆ Author supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 786725 OLYMPUS.

allows two parties to establish a shared cryptographic key from matching passwords of potentially low entropy. PAKEs are extremely useful for establishing secure and authenticated communication channels between peers sharing short common knowledge. The common knowledge could be a PIN typed into different wearables in order to pair them, sensor readings recorded by several cars in order to create an authenticated platoon or a security code manually entered by an admin to connect her maintenance laptop with a backbone router.

On a high level, CPace works as follows. Given a cyclic group $\mathcal{G}$, parties first locally and deterministically compute a generator $g \leftarrow \mathsf{Gen}(\mathsf{pw}), g \in \mathcal{G}$ from their passwords in a secure way, so that $g$ reveals as little information about the password as possible. Then, both parties perform a Diffie-Hellman key exchange by choosing secret exponents $x$ and $y$, respectively, exchanging $g^x$ and $g^y$ and locally compute $K = (g^x)^y = (g^y)^x$. The final key is then computed as the hash of $K$ together with session-identifying information such as transcript. The currently most efficient implementations of the above blueprint protocol use elliptic curve groups of either prime or composite order. To securely compute the generator, the password is first hashed to the finite field $\mathbb{F}_q$ over which the curve is constructed, and then mapped to the curve by a map called $\mathsf{Map2Pt}$. Depending on the choice of curve, efficiency tweaks such as simplified point verification on curves with twist security, or computation with only x-coordinates of points can be applied [22, 23]. Unfortunately, until today, it is not clear how these modifications impact security of CPace, and whether the protocol can be proven secure without assuming $(\mathsf{Map2Pt} \circ \mathsf{H})$ to be a truly random function.

*A short history of CPace.* In 1996, Jablon [30] introduced the SPEKE protocol, which performs a Diffie-Hellman key exchange with generators computed as $g \leftarrow \mathsf{H}_{\mathcal{G}}(\mathsf{pw})$, i.e. using a function $\mathsf{H}_{\mathcal{G}}$ hashing directly to the group. Many variants of SPEKE have emerged in the literature since then, including ones that fixed initial security issues of SPEKE. Among them, the PACE protocol [33, 9] aims at circumventing direct hashing onto the group with an interactive $\mathsf{Map2Pt}$ protocol to compute the password-dependent generators. From this, CPace [22] emerged by combining the best properties of PACE and SPEKE, namely computing the generator without interaction while avoiding the need to hash directly onto the group. More precisely, password-dependent generators are computed as $g \leftarrow \mathsf{Map2Pt}(\mathsf{H}(\mathsf{pw}))$. In 2020, the IRTF working group CFRG has chosen CPace as the recommended protocol for (symmetric) PAKE.

*Prior work on the security of CPace.* Bender et al. [9] conducted a game-based security analysis of the explicitly authenticated PACE protocol variants used in travel documents. Their work focusses on different variants of *interactive* $\mathsf{Map2Pt}$ constructions and hence does not allow for any conclusions about CPace which uses a (non-interactive) function $\mathsf{Map2Pt}$.

Static security of CPace, including function $\mathsf{Map2Pt}$ and some implementation artifacts such as cofactor clearing, was formally analyzed in [22]. Their work is the first to attempt a formalization of $\mathsf{Map2Pt}$ that allows for a security analysis. However, their proof was found to be insufficient by reviews done

during the CFRG selection process [37, 28], and indeed, the claimed security under the plain computational Diffie-Hellman assumption seems to be difficult to achieve. Besides these issues, their work does not consider adaptive corruptions and implementation artifacts such as twist security or single-coordinate representations.

Abdalla et al. [1] analyzed static security of several EKE [8] and SPEKE variants in the UC framework, including SPAKE2 [5] and TBPEKE [34]. They indicate that their proof for TBPEKE could be extended to CPace with generators computed as $\mathsf{H}_{\mathcal{G}}(pw)$ (i.e., without function $\mathsf{Map2Pt}$) if the protocol transcript and password-dependent generator is included in the final key derivation hash. However, in practice it is desirable to avoid unnecessary hash inputs for efficiency reasons and protection against side-channel attacks.

In a concurrent work, Abdalla et al. [2] formalized the algebraic group model within the UC framework and proved that the SPAKE2 and CPace protocols are universally composable in the new model with respect to the standard functionality for password-based authenticated key exchange in [15]. Stebila and Eaton [19] provided a game-based analysis of CPace in the generic group model. As in [1], these further studies do not deal with adaptive security and only consider a basic version of CPace without $\mathsf{Map2Pt}$ and without considering any implementation artifacts.

The above analyses demonstrate that a basic version of CPace, which essentially is a Diffie-Hellman key exchange computed on hashed passwords instead of a public generator, is UC-secure if the attacker is restricted to static corruptions. Unfortunately, this leaves many open questions. Does this basic protocol remain (UC-)secure if we use generator $\mathsf{Map2Pt}(\mathsf{H}(pw))$ instead, as it is done in practice to avoid direct hashing onto elliptic curves? Can the protocol handle adaptive corruptions? Which impact on security do implementation artifacts have, such as co-factor clearing on a composite-order curve group, or single-coordinate representation as used in, e.g., TLS1.3? Can we reduce hash inputs in order to make the protocol less prone to side-channel attacks? Altogether, it turns out the security of the *actually implemented* CPace protocol is not well understood.

*Our Contributions.* In this paper, we provide the first comprehensive security analysis of the CPace protocol that applies also to variants of CPace optimized for usage with state-of-the-art elliptic curves. We identify the core properties of the deterministic $\mathsf{Map2Pt}$ function that allow to prove strong security properties of CPace. Crucially, we restrict the use of random oracles to hash functions only and refrain from modeling $\mathsf{Map2Pt}$ as an idealized function, as it would not be clear how to instantiate it in practice. We show that, using some weak invertibility properties of $\mathsf{Map2Pt}$ that we demonstrate to hold for candidate implementations, CPace can be proven secure under standard Diffie-Hellman-type assumptions in the random-oracle model and with only minimal session-identifiying information included in the final key derivation hash. Our security

proof captures adaptive corruptions and weak forward secrecy[5] and is carried out in the Universal Composability (UC) framework, which is today's standard when analyzing security of password-based protocols. Our work provides the first evidence that SPEKE-type protocols can handle adaptive corruptions.

We then turn our attention to modifications of CPace and, for each modification individually, state under which assumptions the security properties are preserved. In more detail, our analysis captures the following modifications.

- Using groups of composite order $c \cdot p$, where $p$ is a large prime and $c$ is a small cofactor coprime to $p$.
- Realize $\mathsf{Gen}(\mathsf{pw})$ generator calculations using $\mathsf{Map2Pt}$ with either map-twice-and-add strategy or as single execution.
- Using single-coordinate-only representations of elliptic-curve points in order to speed up and facilitate implementation.
- Avoiding computationally costly point verification on curves with secure quadratic twists such as Curve25519 [10].

To demonstrate the security of these variants, we take a new approach that saves us from a repeated analysis of unchanged parts of CPace. Namely, we implement the CDH-type cryptographic assumptions required by CPace as libraries which a simulator can access. This allows for modular replacement of assumptions required in the security proof, and lets us efficiently analyze all the different CPace variants' security guarantees. We believe that this new proof technique might be of independent interest in particular for machine-assisted proving, since reductions are captured in code instead of textual descriptions only.

As a side contribution, we identify a common shortcoming in all UC PAKE security definitions in the literature [15, 31, 29, 1], which impacts the suitability of these definitions as building blocks in higher-level applications. Namely, all these definitions allow a malicious party to learn the shared key computed by an honest party *without knowing her password*. We strengthen the definition to prevent such attacks, and demonstrate with our analysis of CPace that our fix yields a security definition that is still met by PAKE protocols.

In conclusion, our results demonstrate that CPace enjoys strong provable security guarantees in a realistic setting, and this holds for all its variants that have been proposed in the different elliptic-curve ecosystems.

### 1.1 Technical overview of our results

*$\mathsf{Map2Pt}$'s impact on security.* At its core, the CPace protocol is a SPEKE-type protocol, meaning that it is simply a Diffie-Hellman key exchange (DHKE) computed with a generator that each party *individually* computes from her password. Intuitively, the most secure choice is to compute $g \leftarrow \mathsf{H}_{\mathcal{G}}(pw)$, and indeed this was proven secure [1, 2] conditioned on $\mathsf{H}$ being a perfect hash function (or, put

[5] In the case of PAKE, weak forward secrecy is implied by UC security and hence achieved also by prior work. If key confirmation is added, then this gives a protocol with perfect forward secrecy as noted in [1].

differently, a random oracle (RO)). However, DHKE-type protocols are most efficient when implemented on elliptic-curve groups, and it is not known how to efficiently hash directly onto such groups. Recent standardization efforts by the CFRG [20] show that, in practice, one would always first hash to the finite field $\mathbb{F}_q$ over which the curve is constructed, and then map the field element to the curve $\mathcal{G}$ using some curve-specific mapping $\mathsf{Map2Pt} : \mathbb{F}_q \to \mathcal{G}$. Hence, the generator in CPace can be assumed to be computed as $g \leftarrow \mathsf{Map2Pt}(\mathsf{H}(pw))$ for a $\mathsf{H}$ being a hash function such as SHA-3.

In order to analyze how the function $\mathsf{Map2Pt}$ impacts CPace's security, it is obviously not helpful to abstract $\mathsf{Map2Pt} \circ \mathsf{H}$ as a truly random function. In a first attempt to analyze under which properties of $\mathsf{Map2Pt}$ CPace remains secure, Haase et al. [22] assumed $\mathsf{Map2Pt}$ to be a bijection. Intuitively, a bijective $\mathsf{Map2Pt}$ function does not "disturb" the "nice" distribution of the prepended hash function, and in particular does not introduce any collisions. Besides the known shortcomings in their conducted analysis (the claimed security under CDH does not seem to hold, and their proof lacks an indistinguishability argument [37, 28]), it does not cover non-bijective mappings on widely used short-weierstrass curves such as NIST P-256. Hence, in our work we refrain from assuming $\mathsf{Map2Pt}$ to be a bijection. Instead, we introduce a property of *probabilistic invertibility*, which demands that, given an element $g$ in the group $\mathcal{G}$, we can efficiently compute all preimages $h \in \mathbb{F}_q$ such that $\mathsf{Map2Pt}(h) = g$. On a high level, this invertibility property will aid the simulation of CPace since it allows to "tightly" link a group element $g$ to a previously computed hash $h$ and thus recognize collisions efficiently. Here, tightly/efficiently means without iterating over all hash queries in the system. We demonstrate that all mappings used in practice [20] are probabilistically invertible. As a result, we conclude that CPace implemented with current mappings enjoys strong security guarantees.

*Adaptive security.* Just like any other PAKE protocol, CPace comes with a large likelihood for idling. Indeed, in practice it will most likely be the same person who jumps between the two devices running the PAKE, to manually enter the same password, PIN or code. This gives room for attackers to corrupt devices *during the run of the protocol*, and hence calls for analyzing security of CPace in the presence of *adaptive corruptions*. To our knowledge, there is no proof of adaptive security for any SPEKE-type protocol in the literature. In this work, we closely investigate CPace's guarantees under adaptive corruptions and come to an indeed surprising conclusion:

*CPace enjoys adaptive UC security under the same DH-type assumptions that seem required for static security.*

The challenge of proving adaptive security lies in the need to reveal suitable secret values computed by a previously honest party during the run of the protocol. For CPace, these are the secret Diffie-Hellman exponents $x, y$ randomly chosen by parties. A bit simplified, our idea is to start the simulation of an honest party with $g^z$ for a generator $g$ of group $\mathcal{G}$ and randomly chosen exponent $z$, and hence independent of the actual (unknown) password used by that party. Upon

corruption, the simulator learns $pw$ and looks up the corresponding hash value $g^r = \mathsf{H}(pw)$ for which it knows $r^{-1}$ thanks to $\mathsf{H}$ being modeled as a random oracle. This allows the simulator to compute the "actual" secret exponent $y \leftarrow zr^{-1}$ that the simulated party would have used if started with actual password $pw$. Crucially, no additional assumptions or secure erasures are required and, as we demonstrate in the body of our paper, this simplified strategy still works when generators are computed using $\mathsf{Map2Pt} \circ \mathsf{H}$. Altogether, our analysis shows that CPace enjoys UC-security under adaptive corruptions.

*Falsifiable assumptions and a new approach to simulation-based proofs.* A falsifiable assumption can be modeled as an interactive game between an *effcient* challenger and an adversary, at the conclusion of which the challenger can efficiently decide whether the adversary won the game [21]. Most standard cryptographic assumptions such as CDH, DDH, RSA, and LWE are falsifiable. An example of a non-falsifiable assumption is the gap simultaneous Diffie-Hellman assumption, which was used in prior CPace security analyses [1, 2] and features a full DDH oracle that cannot be efficiently implemented by the challenger. Intuitively, the DDH oracle seems inherent for proving UC security of CPace since the attacker (more detailed, the distinguishing environment) determines passwords $pw$ used by honest parties and also receives their outputs, which is the final key $K$. More detailed, the attacker can deterministically compute the generator $G$ used by an honest party from only $pw$, and it also receives the honest party's message $g^x$. The attacker can now enforce the final key to be a DDH tuple $K = g^{xy}$ by simply sending $g^y$ to the honest party (we omit the final key derivation hash in this explanation for simplicity). Hence, to correctly simulate the final key output by an honest party under attack, the simulator relies on a DDH oracle. However, we observe that this oracle can be *limited to specific inputs* $g, g^x$ that the attacker cannot influence. This turns out to be an important limitation, because the *restricted* DDH oracle $DDH(g, g^x, \cdot, \cdot)$ can actually be implemented efficiently using knowledge of trapdoor exponents $r, r^{-1}$ of $g$. Thus, our conclusion is that CPace's security holds under falsifiable DH-type assumptions.

As another contribution, we define falsifiable assumptions as efficiently implementable libraries that a simulator can call. The advantage of this approach is that reductions to the underlying assumptions are *integrated* in the simulator's code, which will hence abort and detect itself whenever a query to the library solves the underlying hard problem. This makes reduction strategies readable from simulator codes and hence opens a new path for automatic verification of simulation-based proofs. While we demonstrate this only to work for proofs conducted in the UC framework and when using variants of strong CDH, we conjecture that our approach can be used for simulation-based proofs in arbitrary frameworks whenever only falsifiable assumptions are used.

*Minimal protocol design.* For optimal protection against side-channel attacks, we would like to have parties touch their passwords as little as possible. Optimally, passwords are only used to compute the generator of the DHKE. Unfortunately, in simulation-based frameworks a security proof often crucially relies on hashing

of secrets, and indeed previous CPace security analysis has relied on the password being included in the final key derivation hash [1]. In this work we ask what the minimal set of protocol-related values is that needs to be included in both hash functions used in CPace. Perhaps surprisingly, we find that CPace's security can be proven when (1) the password hash does not get any additional inputs and (2) the final key derivation hash is over session-specific values and the Diffie-Hellman key. Regarding (1), we observe that the simulation strategy (described above for adaptive corruptions) works even if the generator $g$ chosen by the simulator is used to simulate multiple instances of CPace, and where different parties use the same password: Choosing fresh secret exponents $z_A$ for each such simulated party $A$ ensures that all the revealed exponents $z_A r^{-}1$ are still uniformly distributed. Regarding (2), our simulation simply does not need to learn the password from an adversarial key derivation hash query: The simulator simply reads the simulated parts $g^z$ and adversarial part $Y$ of the transcript from the hash query and checks consistency of the query's format by checking whether it is a DDH tuple with respect to each trapdoor generated upon password hashing. Since there can be only a polynomial number of such queries, this simulation strategy is tight and efficient and saves us from hashing the password another time.

*Implementation artifacts.* Depending on the type of curve CPace is deployed in, the implementation will vary in certain aspects for which it is not clear how they will impact CPace's security. By adopting the security analysis to capture actual Map2Pt mappings used in practice we already demonstrated how to deal with the probably most important such artifact above. Closely related to this, we also analyze security of CPace when implemented on curves of composite order $p \cdot c$ with a small co-factor $c$, which needs to be "cleared" in order to ensure that parties use generators of the large subgroup. We can integrate this modification by chaining Map2Pt with a co-factor clearing function and by demonstrating that the resulting mapping is still probabilistically invertible. Technically, we "lift" our proof of security w.r.t simple Map2Pt described above by letting the simulator call a co-factor clearing class that ensures that simulated values will remain in the large subgroup.

A typical implementation pitfall is incorrectly implemented group-membership verification. As such a failure easily remains unnoticed, optimized resilient protocols such as X25519 and X448 [32] have been suggested for the conventional Diffie-Hellman use-case. We believe that we are the first to formalize the exact hardness assumption, the twist CDH problem sTCDH, under which the claimed resilience regarding group membership omission is actually justified. We show that under the sTCDH assumption, resilience with respect to incorrectly implemented point verification can also be achieved for CPace, when instantiated using single-coordinate Montgomery ladders on so-called "twist-secure" [12] elliptic curves. For details on how to deal with other implementation artifacts we refer the reader to Section 6 in the main body of the paper.

*Roadmap.* We introduce the PAKE security model in Section 2 and hardness assumptions and requirements for Map2Pt in Section 3. Details of the CPace

protocol are in Section 4. Then we analyse CPace, first using a simplified CPace in Section 5 (modeling the map as random-oracle) and then extending the analysis to real-world instantiations using actually deployed mapping constructions, composite-order groups, details on twist security and single-coordinate representations in Section 6. We defer the reader to the full version of this paper [4] for proofs, a description of issues with previous UC PAKE functionalities and implementation recommendations.

## 2 PAKE Security Model

We use the Universal Composability (UC) framework of Canetti [14] to formulate security properties of CPace. For PAKE, usage of the simulation-based UC framework comes with several advantages over the game-based model for PAKE introduced by Bellare et al. [7]. Most importantly, UC secure PAKE protocols preserve their security properties in the presence of adversarially-chosen passwords and when composed with arbitrary other protocols. Originally introduced by Canetti et al. [15], the ideal functionality $\mathcal{F}_{\mathsf{pwKE}}$ for PAKE (depicted in Fig. 1) is accessed by two parties, $\mathcal{P}$ and $\mathcal{P}'$, who both provide their passwords. $\mathcal{F}_{\mathsf{pwKE}}$ then provides both parties with a uniformly random session key if passwords match, and with individual random keys if passwords mismatch. Since an adversary can always engage in a session and guess the counterpart's password with non-negligible probability, $\mathcal{F}_{\mathsf{pwKE}}$ must include an adversarial interface $\mathsf{TestPwd}$ for such guesses. Crucially, only one guess against every honest party is allowed, modeling the fact that password guessing is an online attack and cannot be used to brute-force the password from a protocol's transcript. We refer the reader to [15] for a more comprehensive introduction to the PAKE functionality.

*An ideal functionality for the SPEKE protocol family.* Unfortunately, $\mathcal{F}_{\mathsf{pwKE}}$ is not suitable to analyze SPEKE-like PAKE protocols such as CPace, where session keys are computed as hashes of Diffie-Hellman keys (and possibly parts of the transcript). The reason is that $\mathcal{F}_{\mathsf{pwKE}}$'s $\mathsf{TestPwd}$ interface allows password guesses only *during* a protocol run, which requires a simulator to extract password guesses from the protocol's transcript. When the final output is a hash, the adversary might postpone its computation, keeping information from the simulator that is required for password extraction. To circumvent these issues, recently a "lazy-extraction PAKE" functionality $\mathcal{F}_{\mathsf{lePAKE}}$ was proposed and shown useful in the analysis of SPEKE-like protocols by Abdalla et al. [1]. $\mathcal{F}_{\mathsf{lePAKE}}$, which we also depict in Fig. 1, allows *either* one online *or* one offline password guess after the key exchange was finished. One might argue that usage of keys obtained from $\mathcal{F}_{\mathsf{lePAKE}}$ is never safe, since the adversary might eventually extract the key from it at any later point in time. This however can be easily prevented by adding a key confirmation round, which keeps an adversary from postponing the final hash query and guarantees perfect forward secrecy [1]. We refer the reader to [1] for a thorough discussion of $\mathcal{F}_{\mathsf{lePAKE}}$.

*Our adjustments to* $\mathcal{F}_{\mathsf{lePAKE}}$. The main difference between our $\mathcal{F}_{\mathsf{lePAKE}}$ and all PAKE functionalities from the literature [15, 31, 29, 1] is that we remove a shortcoming that rendered these functionalities essentially useless as building blocks for higher-level applications. More detailed, we remove the ability of the adversary to determine an honest party's output key in a corrupted session. The change can be seen in Fig. 1, where the dashed box shows the weakening that we simply omit in our version of $\mathcal{F}_{\mathsf{lePAKE}}$. In reality, nobody would want to use a PAKE where an adversary can learn (even set) the key of an honest party *without knowing the honest party's password*. This is not what one would expect from an authenticated key exchange protocol. In the full version of this work [4] we explain why existing PAKE protocols can still be considered secure, but also provide an illustrating example how this shortcoming hinders usage of PAKE functionalities in modular protocol analysis. In this paper, we demonstrate that CPace can be proven to protect against such attacks.

We also make two minor adjustments, which are merely to ease presentation in this paper. Namely, we add an explicit interface for adaptive corruptions, and we omit roles since we analyze a protocol where there is no dedicated initiator.

*How many keys can a PAKE functionality exchange?* All PAKE functionalities in Figure 1 produce only a single key for a single pair of parties $\mathcal{P}, \mathcal{P}'$. This can be seen from the NewSession interface, which takes action only upon the first such query (from any party $\mathcal{P}$) and the corresponding second query by the indicated counterparty $\mathcal{P}'$. The motivation behind this design choice is simplicity in the security analysis: one can prove security of a PAKE protocol for only a single session, and then run arbitrary many copies of the PAKE functionality to exchange arbitrarily many keys (between arbitrary parties). Consequently, by the UC composition theorem, replacing all those copies with the PAKE protocol that provably realizes the single-session $\mathcal{F}_{\mathsf{pwKE}}$ is at least as secure.

## 3 Preliminaries

### 3.1 Notation

With $\leftarrow_{\mathrm{R}}$ we denote uniformly random sampling from a set. With $\mathsf{oc}(X, Y)$ we denote ordered concatenation, i.e., $\mathsf{oc}(X, Y) = X||Y$ if $X \leq Y$ and $\mathsf{oc}(X, Y) = Y||X$ otherwise. We use multiplicative notation for the group operation in a group $\mathcal{G}$ and hence write, e.g., $g \cdot g = g^2$ for an element $g \in \mathcal{G}$. $I_{\mathcal{G}}$ denotes the neutral element in $\mathcal{G}$. To enhance readability, we sometimes break with the convention of denoting group elements with small letters and write $X := g^x$. We denote by $\mathcal{G}_m$ a subgroup of $\mathcal{G}$ of order $m$, and with $\bar{\mathcal{G}}$ we denote the quadratic twist of elliptic curve group $\mathcal{G}$. Throughout the paper, we use $\lambda$ as security parameter[6].

[6] For the hardness assumptions on elliptic curve groups, e.g. for the sCDH and sSDH problems, where security depends on the group type and the group order $p$, the bit size of $p$ implicitly serves also as a further security parameter.

Session initiation
On $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}', \mathsf{pw})$ from $\mathcal{P}$, send $(\mathsf{NewSession}, sid, \mathcal{P}, \mathcal{P}')$ to $\mathcal{A}$. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record $(sid, \mathcal{P}', \mathcal{P}, \mathsf{pw}')$, then record $(sid, \mathcal{P}, \mathcal{P}', \mathsf{pw})$ and mark this record fresh.

Active attack
- On $(\mathsf{TestPwd}, sid, \mathcal{P}, \mathsf{pw}^*)$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{pw}, \cdot\rangle$ then:
  - If $\mathsf{pw}^* = \mathsf{pw}$ then mark it compromised and return "correct guess";
  - If $\mathsf{pw}^* \neq \mathsf{pw}$ then mark it interrupted and return "wrong guess".
- On $(\mathsf{RegisterTest}, sid, \mathcal{P})$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', \cdot\rangle$ then mark it interrupted and flag it tested.
- On $(\mathsf{LateTestPwd}, sid, \mathcal{P}, \mathsf{pw}^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{pw}, K\rangle$ marked completed with flag tested then remove this flag and do:
  - If $\mathsf{pw}^* = \mathsf{pw}$ then return $\boxed{K}$ to $\mathcal{A}$;
  - If $\mathsf{pw}^* \neq \mathsf{pw}$ then return $\boxed{K^{\$} \leftarrow_{\mathrm{R}} \{0,1\}^{\lambda}}$ to $\mathcal{A}$.

Key generation
On $(\mathsf{NewKey}, sid, \mathcal{P}, K^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{pw}\rangle$ not marked completed then do:
- If the record is compromised, or either $\mathcal{P}$ or $\mathcal{P}'$ is corrupted, then $K := K^*$.
- If the record is fresh and $\exists$ a completed record $\langle sid, \mathcal{P}', \mathcal{P}, \mathsf{pw}, K'\rangle$ that was fresh when $\mathcal{P}'$ output $(sid, K')$, then set $K := K'$.
- In all other cases pick $K \leftarrow_{\mathrm{R}} \{0,1\}^{\lambda}$.

Finally, append $K$ to record $\langle sid, \mathcal{P}, \mathcal{P}', \mathsf{pw}\rangle$, mark it completed, and output $(sid, K)$ to $\mathcal{P}$.

Adaptive corruption
On $(\mathsf{AdaptiveCorruption}, sid, \mathcal{P})$ from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \cdot, \mathsf{pw}\rangle$ not marked completed then mark it completed and output $(sid, \mathsf{pw})$.

**Fig. 1.** UC PAKE variants: The original PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ of Canetti et al. [15] is the version with all gray text omitted. The lazy-extraction PAKE functionality $\mathcal{F}_{\mathsf{lePAKE}}$ [1] includes everything, and the variant of $\mathcal{F}_{\mathsf{lePAKE}}$ used in this work includes everything but the dashed box.

### 3.2 Cryptographic assumptions

The security of CPace is based on the hardness of a combination of strong and simultaneous Diffie-Hellman problems. To ease access to the assumptions, we state them with increasing complexity.

**Definition 1 (Strong CDH problem (sCDH) [3]).** *Let $\mathcal{G}$ be a cyclic group with a generator $g$ and $(X = g^x, Y = g^y)$ sampled uniformly from $(\mathcal{G} \setminus \{I_{\mathcal{G}}\})^2$. Given access to oracles $\mathsf{DDH}(g, X, \cdot, \cdot)$ and $\mathsf{DDH}(g, Y, \cdot, \cdot)$, provide $K$ such that $K = g^{xy}$.*

We note that sCDH is a weaker variant of the so-called gap-CDH assumption, where the adversary has access to "full" DDH oracles with no fixed inputs. Next we provide a stronger variant of sCDH where two CDH instances need to be solved that involve a common, adversarially chosen element.

**Definition 2 (Strong simultaneous CDH problem (sSDH)).** *Let $\mathcal{G}$ be a cyclic group and $(X, g_1, g_2)$ sampled uniformly from $(\mathcal{G} \setminus \{I_\mathcal{G}\})^3$. Given access to oracles $\mathsf{DDH}(g_1, X, \cdot, \cdot)$ and $\mathsf{DDH}(g_2, X, \cdot, \cdot)$, provide $(Y, K_1, K_2) \in (\mathcal{G} \setminus \{I_\mathcal{G}\}) \times \mathcal{G} \times \mathcal{G}$ s. th. $\mathsf{DDH}(g_1, X, Y, K_1) = \mathsf{DDH}(g_2, X, Y, K_2) = 1$*

As a cryptographic assumption sSDH above is justified since sSDH is implied by the gap simultaneous Diffie-Hellman assumption [34, 1], which allows for unlimited (i.e., with no fixed input) access to a DDH oracle. Lastly, we state a variant of the sSDH assumption where generators are sampled according to some probability distribution. Looking ahead, we require this variant since in CPace parties derive generators by applying a map which does not implement uniform sampling from the group. We state the non-uniform variant of sSDH for arbitrary probability distributions and investigate its relation to "uniform" sSDH afterwards.

With $\mathbf{Adv}^{\mathsf{sCDH}}_{\mathcal{B}_{\mathsf{sCDH}}}(\mathcal{G})$ and $\mathbf{Adv}^{\mathsf{sSDH}}_{\mathcal{B}_{\mathsf{sSDH}}}(\mathcal{G})$, we denote the probabilities that adversarial algorithms $\mathcal{B}_{\mathsf{sSDH}}$ and $\mathcal{B}_{\mathsf{sSDH}}$ having access to the restricted DDH oracles provide a solution for the sCDH and sSDH problems respectively in $\mathcal{G}$ when given a single randomly drawn challenge.

**Definition 3 (Strong simultaneous non-uniform CDH problem ($\mathcal{D}_\mathcal{G}$-sSDH)).** *Let $\mathcal{G}$ be a group and $\mathcal{D}_\mathcal{G}$ be a probability distribution on $\mathcal{G}$. The strong simultaneous non-uniform CDH problem $\mathcal{D}_\mathcal{G}$-sSDH is defined as the sSDH problem but with $(X, g_1, g_2)$ sampled using $\mathcal{U}_\mathcal{G} \times \mathcal{D}_\mathcal{G} \times \mathcal{D}_\mathcal{G}$, where $\mathcal{U}_\mathcal{G}$ denotes the uniform distribution on $\mathcal{G}$.*

Clearly, $\mathcal{U}_{\mathcal{G}\setminus\{I_\mathcal{G}\}}$-sSDH is equivalent to sSDH. We show that hardness of uniform and non-uniform sSDH are equivalent given that the distribution allows for probabilistic polynomial time (PPT) rejection sampling, which we now formalize.

**Definition 4 (Rejection sampling algorithm for $(\mathcal{G}, \mathcal{D}_\mathcal{G})$).** *Let $\mathcal{G}$ be a group and $\mathcal{D}_\mathcal{G}$ be a probability distribution on $\mathcal{G}$. With $\mathcal{D}_\mathcal{G}(g)$ we denote the probability for point $g$. Let $\mathsf{RS}$ be a probabilistic algorithm taking as input elements $g \in \mathcal{G}$ and outputting $\perp$ or a value $\neq\perp$. Then $\mathsf{RS}$ is called a* rejection sampling algorithm for $(\mathcal{G}, \mathcal{D}_\mathcal{G})$ *if there is a scaling factor $k$ such that $Pr[\mathsf{RS}(g) \neq \perp] = k \cdot \mathcal{D}_\mathcal{G}(g)$ for $g \in \mathcal{G}$.*

Informally RS is a probabilistic algorithm which accepts (output different from $\perp$) or rejects (output $\perp$) a candidate point. When queried multiple times on the same input $g \in \mathcal{G}$, the probability that $g$ will be accepted or rejected models a scaled distribution that is proportional to $\mathcal{D}_\mathcal{G}$. In this paper, we are interested in rejection samplers with "good" acceptance rate, such that they can be efficiently used to sample elements from the scaled distribution. We formalize the acceptance rate as follows.

**Definition 5 (Acceptance rate of a rejection sampler for $(\mathcal{G}, \mathcal{D}_\mathcal{G})$).** *Let $\mathcal{G}$ be a group and $\mathcal{D}_\mathcal{G}$ be a probability distribution on $\mathcal{G}$. Let $\mathsf{RS}$ be a rejection sampling algorithm for $(\mathcal{G}, \mathcal{D}_\mathcal{G})$. Let $g_i \in \mathcal{G}$ be a sequence of $m$ uniformly drawn points and $r_i = \mathsf{RS}(g_i)$. Then $\mathsf{RS}$ is said to have an acceptance rate of $(1/n)$ if the number of accepted points with $r_i \neq \perp$ converges to $m/n$ when $m \to \infty$.*

Using these definitions, we are able to prove that given some assumptions on the distribution $\mathcal{D}_\mathcal{G}$ hardness of $\mathsf{sSDH}$ and $\mathcal{D}_\mathcal{G}$-$\mathsf{sSDH}$ are equivalent up to the additional PPT computational effort for the rejection sampling algorithm.

**Theorem 1 ($\mathsf{sSDH} \iff \mathcal{D}_\mathcal{G}$-$\mathsf{sSDH}$).** *Let $\mathcal{G}$ be a cyclic group of order $p$ and $\mathcal{D}_\mathcal{G}$ a probability distribution on $\mathcal{G}$. If there exists a PPT rejection sampler $\mathsf{RS}$ for $(\mathcal{G}, \mathcal{D}_\mathcal{G})$ with acceptance rate $(1/n)$ then the probability of PPT adversaries against $\mathcal{D}_\mathcal{G}$-$\mathsf{sSDH}$ and $\mathsf{sSDH}$ of solving the respectively other problem differs by at most $(2\mathcal{D}(I_\mathcal{G}) + (1/p))$ and solving $\mathsf{sSDH}$ with the help of a $\mathcal{D}_\mathcal{G}$-$\mathsf{sSDH}$ adversary requires at most $2n$ executions of $\mathsf{RS}$ on average.*

*Proof.* **$\mathsf{sSDH}$ hard $\Rightarrow \mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ hard:** Given an adversary $\mathcal{B}_{\mathcal{D}_\mathcal{G}-\mathsf{sSDH}}$ against $\mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ with non-negligible success probability $\nu$, we show how to construct an adversary $\mathcal{A}_{\mathsf{sSDH}}$. On receiving an $\mathsf{sSDH}$-challenge $(X, g_1, g_2)$, first note that $X$ is uniformly sampled from $\mathcal{G} \setminus \{I_\mathcal{G}\}$. $\mathcal{A}_{\mathsf{sSDH}}$ uniformly samples $r, s \in \mathbb{Z}_p$ until $\mathsf{RS}(g_1^r) \neq \perp$ and $\mathsf{RS}(g_2^s) \neq \perp$, which requires $2n$ calls to $\mathsf{RS}$ on average. $\mathcal{A}_{\mathsf{sSDH}}$ runs $\mathcal{B}_{\mathcal{D}_\mathcal{G}-\mathsf{sSDH}}$ on input $(X, g_1^r, g_2^s)$. If $\mathcal{B}$ queries $\mathsf{DDH}(g_1^r, X, Z, L)$, $\mathcal{A}$ queries his own oracle with $\mathsf{DDH}(g_1, X, Z, L^{1/r})$ and relays the answer to $\mathcal{B}$ (queries $g_2^s$ are handled analogously). On receiving $(Y, K_1, K_2)$ from $\mathcal{B}_{\mathcal{D}_\mathcal{G}-\mathsf{sSDH}}$, $\mathcal{A}_{\mathsf{sSDH}}$ provides $(Y, K_1^{1/r}, K_2^{1/s})$ as solution in his $\mathsf{sSDH}$ experiment.

As $\mathsf{RS}$ is a rejection sampler for $\mathcal{D}_\mathcal{G}$, $(X, g_1^r, g_2^s)$ is a random $\mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ challenge, and thus $\mathcal{B}$ solves it with probability $\nu$. If $\mathcal{B}$ provides a solution, then $\mathcal{A}_{\mathsf{sSDH}}$ succeeds in solving his own challenge unless $g_1^r$ or $g_2^s = I_\mathcal{G}$ or $g_1^r = g_2^s$ which occurs at most with probability $(2\mathcal{D}_\mathcal{G}(I_\mathcal{G}) + 1/p)$. As $\mathsf{RS}$ executes in PPT, $\mathcal{A}_{\mathsf{sSDH}}$ is PPT, uses $(2n)$ calls to $\mathsf{RS}$ on average and succeeds with probability $\nu(1 - 2\mathcal{D}_\mathcal{G}(I_\mathcal{G}) - 1/p)$, which is non-negligible since $\nu$ is.

**$\mathsf{sSDH}$ hard $\Rightarrow \mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ hard:** Given an adversary $\mathcal{A}_{\mathsf{sSDH}}$ against $\mathsf{sSDH}$ with non-negligible probability $\mu$ we show how to construct a $\mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ adversary $\mathcal{B}_{\mathcal{D}_\mathcal{G}-\mathsf{sSDH}}$. On receiving a $\mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ challenge $(X, g_1, g_2)$, $\mathcal{B}$ samples $r, s \in \mathbb{Z}_p \setminus 0$ and starts $\mathcal{A}_{\mathsf{sSDH}}$ on input $(X, g_1^r, g_2^s)$. DDH oracle queries are handled the same as above. On receiving $(Y, K_1, K_2)$ from $\mathcal{A}_{\mathsf{sSDH}}$, $\mathcal{B}$ provides $(Y, K_1^{1/r}, K_2^{1/s})$ as solution to his own challenge.

If $\mathcal{A}$ is successful, then $\mathcal{B}$ succeeds unless either $g_1$ or $g_2 = I_\mathcal{G}$ or $g_1^r = g_2^s$ which occurs at most with probability $(2\mathcal{D}_\mathcal{G}(I_\mathcal{G}) + 1/p)$. Thus, $\mathcal{B}$ is a PPT adversary against $\mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ succeeding with non-negligible probability $\mu(1 - 2\mathcal{D}_\mathcal{G}(I_\mathcal{G}) - 1/p)$.

Informally, the assumptions $\mathsf{sSDH}$ and $\mathcal{D}_\mathcal{G} - \mathsf{sSDH}$ become equivalent if stepping over an element that gets accepted in the sampling process becomes sufficiently likely for a randomly drawn sequence of candidates. Secondly, the probability of accidentally drawing the neutral element from $\mathcal{D}_\mathcal{G}$ needs to be negligible.

### 3.3 Transforming passwords to points on an elliptic curve

The generators of the Diffie-Hellman exchange in CPace are computed using a deterministic mapping function $\mathsf{Gen}(pw)$. For a given curve group $\mathcal{G}$ over a field $\mathbb{F}_q$, $\mathsf{Gen}(pw)$ is calculated with the help of either one ($\mathsf{Gen}_{\mathsf{1MAP}}$) or two ($\mathsf{Gen}_{\mathsf{2MAP}}$) invocations of a function $\mathsf{Map2Pt}_{\mathcal{G}} : \mathbb{F}_q \to \mathcal{G}$ and a hash function $H_1$ hashing to $\mathbb{F}_q$. For the sake of shortened notation, we will drop the $\mathcal{G}$ subscript where the group is obvious from the context. In both cases, security of CPace relies on $\mathsf{Map2Pt}$ meeting the requirements from this section. Informally, we first require $\mathsf{Map2Pt}$ to be "invertible". That is, for any point on the image of the map, there must be an efficient algorithm that outputs all preimages in $\mathbb{F}_q$ of $\mathsf{Map2Pt}_{\mathcal{G}}$ for a given group element $g$. We use the notation $\mathsf{Map2Pt}_{\mathcal{G}}.\mathsf{PreImages}(g)$. Details on how such an inversion algorithm can be efficiently implemented for various elliptic curve groups are given in [20, 11, 13, 27] and references therein. Secondly, a bound for the maximum number of preimages $n_{\max}$ that $\mathsf{Map2Pt}_{\mathcal{G}}$ maps to the same element must be known and this $n_{\max}$ bound needs to be small (we use the notation $\mathsf{Map2Pt}_{\mathcal{G}}.n_{\max}$ for the bound that applies for a given $\mathsf{Map2Pt}_{\mathcal{G}}$ function and group $\mathcal{G}$). This is needed in order to construct a rejection sampling algorithm whose acceptance rate must depend on $n_{\max}$.

**Definition 6.** *Let $\mathcal{G}$ be a group of points on an elliptic curve over a field $\mathbb{F}_q$. Let $\mathsf{Map2Pt} : \mathbb{F}_q \to \mathcal{G}$ be a deterministic function. Then $\mathsf{Map2Pt}(\cdot)$ is called* probabilistically invertible with at most $n_{\max}$ preimages *if there exists a probabilistic polynomial-time algorithm $(r_1, \ldots, r_{n_g}) \leftarrow \mathsf{Map2Pt.PreImages}(g)$ that outputs all $n_g$ values $r_i \in \mathbb{F}_q$ such that $g = \mathsf{Map2Pt}(r_i)$ for any $g \in \mathcal{G}$; and $\forall g \in \mathcal{G}$, $n_{max} \geq n_g \geq 0$.*

For a map $\mathsf{Map2Pt}$ that fulfills the previous definition with a bound for the numbers of preimages $\mathsf{Map2Pt}.n_{\max}$, we define an "inversion algorithm" $\mathsf{Map2Pt}^{-1} : \mathcal{G} \to \mathbb{F}_q$ that, on input $g \in \mathcal{G}$, returns one of potentially many preimages of $g$ under $\mathsf{Map2Pt}$ if a biased coin comes up heads. If the coin comes up tails, the algorithm outputs failure. The "inversion algorithm" also serves as rejection sampling algorithm for the distribution $\mathcal{D}_{\mathcal{G}}$ that is produced by $\mathsf{Map2Pt}(r)$ for uniformly distributed inputs $r \in \mathbb{F}_q$:

**Algorithm 1** $\mathsf{Map2Pt}^{-1} : \mathcal{G} \longrightarrow \mathbb{F}_q \cup \{\perp\}$

On input $g \in \mathcal{G}$: Sample $i$ uniformly from $\{1, \ldots, \mathsf{Map2Pt}.n_{\max}\}$; Then obtain $n_g \in \{0, \ldots, \mathsf{Map2Pt}.n_{\max}\}$ pre-images $(r_1, \ldots, r_m) \leftarrow \mathsf{Map2Pt.PreImages}(g)$; If $n_g < i$ return $\perp$, else return $r_i$.

**Lemma 1.** *Let $\mathsf{Map2Pt} : \mathcal{G} \to \mathbb{F}_q$ be probabilistically invertible with at most $\mathsf{Map2Pt}.n_{max}$ preimages and let $\mathcal{D}_{\mathcal{G}}$ denote the distribution it induces on $\mathcal{G}$. Then Algorithm 1 is a PPT rejection sampler for $(\mathcal{G}, \mathcal{D}_{\mathcal{G}})$ with average acceptance rate $(|\mathbb{F}_q|/|\mathcal{G}|)/\mathsf{Map2Pt}.n_{max}$.*

*Proof.* We first define the average number of preimages $n_{\max} \geq \bar{n} \geq 1$ as the quotient of the order of the field $\mathbb{F}_q$ and the number of points on the image of the map, i.e., $\bar{n} = |\mathbb{F}_q|/|\text{support}(\mathcal{D}_\mathcal{G})|$. When drawing an element $g$ uniformly from $\mathcal{G}$, the probability that the number of preimages $n_g$ for $g$ is nonzero is given by the quotient of the order of the support of $\mathcal{D}_\mathcal{G}$ and the order of the group. By the definition of $\bar{n}$ above this is $|\mathbb{F}_q|/(\bar{n}|\mathcal{G}|)$.

For any point on the map with a nonzero number $n_g$ of preimages, Algorithm 1 returns a result $\neq\perp$ with probability $n_g/n_{\max}$. As the average value for the number of preimages for any point on the image of the map is $\bar{n}$, the average acceptance rate is $(|\mathbb{F}_q|/(\bar{n}|\mathcal{G}|)) \cdot \bar{n}/n_{\max} = (|\mathbb{F}_q|/|\mathcal{G}|)/n_{\max}$.

*Use of $\mathsf{Map2Pt}^{-1}$ for uniformly sampling field elements from $\mathbb{F}_q$.* As $\mathsf{Map2Pt}$ is deterministic, each point $g$ from $\mathcal{G}$ is characterized by the number of preimages $n_g$ for $\mathsf{Map2Pt}$ in $\mathbb{F}_q$ with $n_{\max} \geq n_g \geq 0$. When generating points $\mathsf{Map2Pt}(s) \in \mathcal{G}$ for uniformly sampled field elements $s \leftarrow_\text{R} \mathbb{F}_q$, the probability of obtaining a given point $g$ is $(n_g/q)$ and can only take the values of zero or integer multiples of $1/q$ up to $n_{\max}/q$. In order to compensate for this, $\mathsf{Map2Pt}^{-1}$ is constructed such that the probability of returning $r \neq\perp$ for a point $g$ increases proportionally with $n_g$ making any actually produced field element $r \neq\perp$ equally likely in $\mathbb{F}_q$. As a result, we can use $\mathsf{Map2Pt}^{-1}$ for transforming a sequence of uniformly sampled group elements $g_l \in \mathcal{G}$ to a sequence of uniformly sampled field elements $r_l \in \mathbb{F}_q$

**Corollary 1.** *Let $\mathsf{Map2Pt}$ be a probabilistically invertible map with at most $\mathsf{Map2Pt}.n_{max}$ preimages and let $g_l \leftarrow_R \mathcal{G}$. Then $r_l \leftarrow \mathsf{Map2Pt}^{-1}(g_l)$ outputs results $r_l \neq\perp$ with probability $p \geq (|\mathbb{F}_q|/|\mathcal{G}|)/\mathsf{Map2Pt}.n_{max}$ and the distribution of outputs $r_l \neq\perp$ is uniform in $\mathbb{F}_q$.*

Moreover as the collision probability when drawing two elements $r_a, r_b$ from $\mathbb{F}_q$ is $1/q$ and as there are at most $n_{\max}$ values $s_l$ generating the same group element $g = \mathsf{Map2Pt}(s_l)$ the collision probability for $g_a = \mathsf{Map2Pt}(r_a)$ and $g_b = \mathsf{Map2Pt}(r_b)$ is increased at most by $n_{\max}^2$.

**Corollary 2.** *When sampling two field elements $r_a, r_b \leftarrow_R \mathbb{F}_q$ uniformly, we have $\mathsf{Map2Pt}(r_a) = \mathsf{Map2Pt}(r_b)$ with a probability of at most $n_{max}^2/q$.*

## 4 The CPace protocol

The CPace protocol [22] is a SPEKE-type protocol [30] allowing parties to compute a common key via a Diffie-Hellman key exchange with password-dependent generators. The blueprint of the protocol is depicted in Fig. 2. Informally, a party $\mathcal{P}$ willing to establish a key with party $\mathcal{P}'$ first computes a generator $g$ from a password $\mathsf{pw}$. Next, $\mathcal{P}$ generates an element $Y_a = g^{y_a}$ from a secret value $y_a$ sampled at random and sends it to $\mathcal{P}'$. Upon receiving a value $Y_b$ from $\mathcal{P}'$, $\mathcal{P}$ then computes a Diffie-Hellman key $K = (Y_b)^{y_a} = g^{y_a y_b}$ and aborts if $K$ equals the identity element. Finally, it computes the session key as the hash of $K$ and the exchanged values $Y_a$ and $Y_b$.

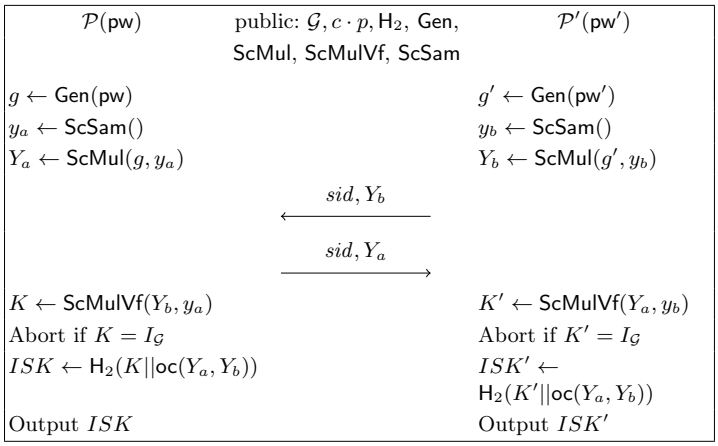


**Fig. 2. Above:** Blueprint protocol CPace[Gen, ScMul, ScMulVf, ScSam] requiring group $\mathcal{G}$ of order $c \cdot p$ with prime $p$ and algorithms for DH generator computation (Gen), exponentiation (ScMul, ScMulVf) and scalar sampling (ScSam). $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^\lambda$ denotes a hash function.
**Below:** "Basic" CPace $\mathsf{CPace}_{\mathsf{base}}$ with $c = 1$, generators computed from hash function $\mathsf{H}_{\mathcal{G}} : \{0,1\}^* \to \mathcal{G}$ and canonical exponentation, point verification and sampling.

| $\mathsf{CPace}_{\mathsf{base}} := \mathsf{CPace}[\mathsf{Gen}_{\mathsf{RO}}, \mathsf{ScMul}_{\mathsf{base}}, \mathsf{ScMulVf}_{\mathsf{base}}, \mathsf{ScSam}_p]$ | | | |
|---|---|---|---|
| $\mathsf{Gen}_{\mathsf{RO}}(\mathsf{pw})$ : | $\mathsf{ScMul}_{\mathsf{base}}(g, y)$ : | $\mathsf{ScMulVf}_{\mathsf{base}}(g, y)$ : | $\mathsf{ScSam}_p()$ : |
| return $\mathsf{H}_{\mathcal{G}}(\mathsf{pw})$ | return $g^y$ | if $g \notin \mathcal{G}$: return $I_{\mathcal{G}}$ | $y \leftarrow_{\mathrm{R}} \{1, \ldots, p\}$ |
| | | else: return $g^y$ | return $y$ |

In order to allow for efficient instantiations over different types of groups, most of which are elliptic curves, we present the CPace protocol in form of a blueprint CPace[Gen, ScMul, ScMulVf, ScSam] in Fig. 2 that provides the following generalizations: (1) The blueprint uses a generic algorithm $\mathsf{Gen}(\mathcal{D}) \to \mathcal{G}$ that turns a password from dictionary $\mathcal{D}$ into a group element; (2) The computation of the $y_i$ and $Y_i$ values is done with generic algorithms for sampling ($\mathsf{SamSc} : 0, 1^* \to 0, 1^*$) and scalar multiplication ($\mathsf{ScMul} : \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|} \to \mathcal{G}$); (3) The Diffie-Hellman key is computed with another generic algorithm $\mathsf{ScMulVf} : \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|} \to \mathcal{G}$, in order to allow for additional point verification that is necessary on some curves (but not on all) to protect against trivial attacks; (4) the blueprint protocol uses an ordered concatenation function oc so that messages can be sent in any order and parties do not have to play a specific initiator or responder role. In the remainder of the paper, we will instantiate the CPace blueprint in various ways, by specifying a set of concrete algorithms Gen, ScMul, ScMulVf, ScSam.

*On the necessity of point verification.* Many elliptic curve scalar multiplication algorithms will work correctly independent whether the input operand encodes a point on the correct curve or not. As a consequence if group membership is not correctly verified by an implementation various attack scenarios become feasible. An active attacker may for instance provide a point on a curve of low

order on which the discrete-logarithm problem could be solved. The threat for real-world implementations is that this serious error might remain undetected as the corresponding verification event is never generated in communications of honest parties. In order to make CPace resilient to this type of attack and implementation pitfalls, [22] suggested to first restrict invalid curve attacks to the quadratic twist (by using a single-coordinate Montgomery ladder) and then choose a curve where also the twist has a large prime-order subgroup and invalid curve attacks become impossible. The CPace draft [23] highlights this aspect on the protocol specification level by introducing a $\mathsf{ScMulVf}$ function which is specified to include point verification.

## 5 Security of Simplified CPace

In this Section, as a warm-up, we analyze security of a "basic" variant of CPace, which we call $\text{CPace}_{\mathsf{base}}$ and which is depicted in Fig. 2. We instantiate $\mathsf{Gen}$ with a hash function $\mathsf{H}_{\mathcal{G}}$ that hashes onto the group $\mathcal{G}$. This way, parties compute generators as $g \leftarrow \mathsf{H}_{\mathcal{G}}(\mathsf{pw})$. Further, we assume $\mathcal{G}$ to be a multiplicatively written group of prime order $p$ where group membership is efficiently decidable. We instantiate $\mathsf{ScMul}(g, y) := g^y$ as exponentiation, and $\mathsf{ScMulVf}(g, y)$ such that it returns the neutral element if $g$ is not in the group and $g^y$ otherwise, and $\mathsf{SamSc}$ with uniform sampling from $\{1 \dots p\}$. A formal description of the protocol is given in Fig. 2, where the blueprint protocol is instantiated with the algorithms at the bottom of the Figure.

**Theorem 2 (Security of $\text{CPace}_{\mathsf{base}}$).** *Let $\lambda, p \in \mathbb{N}$ with $p$ prime. Let $\mathcal{G}$ be a group of order $p$, and let $\mathsf{H}_1 : \{0,1\}^* \to \mathcal{G}, \mathsf{H}_2 : \{0,1\} \to \{0,1\}^\lambda$ be two hash functions. If the sCDH and sSDH problems are hard in $\mathcal{G}$, then protocol $\text{CPace}_{\mathsf{base}}$ depicted in Fig. 2 UC-emulates $\mathcal{F}_{\mathsf{lePAKE}}$ in the random-oracle model with respect to adaptive corruptions when both hash functions are modeled as random oracles. More precisely, for every adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_{\mathsf{sSDH}}$ and $\mathcal{B}_{\mathsf{sCDH}}$ against the strong CDH (sCDH) and strong simultaneous CDH (sSDH) problems such that*

$$\begin{aligned} &|Pr[Real_{\mathcal{Z}}(CPace_{\mathsf{base}}, \mathcal{A})] - Pr[Ideal_{\mathcal{Z}}(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})| \\ &\qquad \leq l_{\mathsf{H}_1}^2/p + 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{\mathsf{sSDH}}_{\mathcal{B}_{\mathsf{sSDH}}}(\mathcal{G}) + \mathbf{Adv}^{\mathsf{sCDH}}_{\mathcal{B}_{\mathsf{sCDH}}}(\mathcal{G}) \end{aligned}$$

*where $l_{\mathsf{H}_1}$ denotes the number of $\mathsf{H}_1$ queries made by the adversary $\mathcal{A}$ and the simulator $\mathcal{S}$ is depicted in Fig. 3.*

*Proof (Sketch).* The main idea of the simulation is to fix a secret generator $g \in \mathcal{G}$ and carry out the simulation with respect to $g$. Messages of honest parties are simulated as $g^z$ for a fresh exponent $z$. Queries $\mathsf{H}_1(\mathsf{pw})$ are answered with $g^r$ for a freshly chosen "trapdoor" $r$. The simulator might learn an honest party's password via adaptive corruption or via an adversarial password guess. The simulator can now adjust the simulation in retrospective to let the honest party use $g^r = \mathsf{H}_1(\mathsf{pw})$ by claiming the party's secret exponent to be $zr^{-1}$. This already

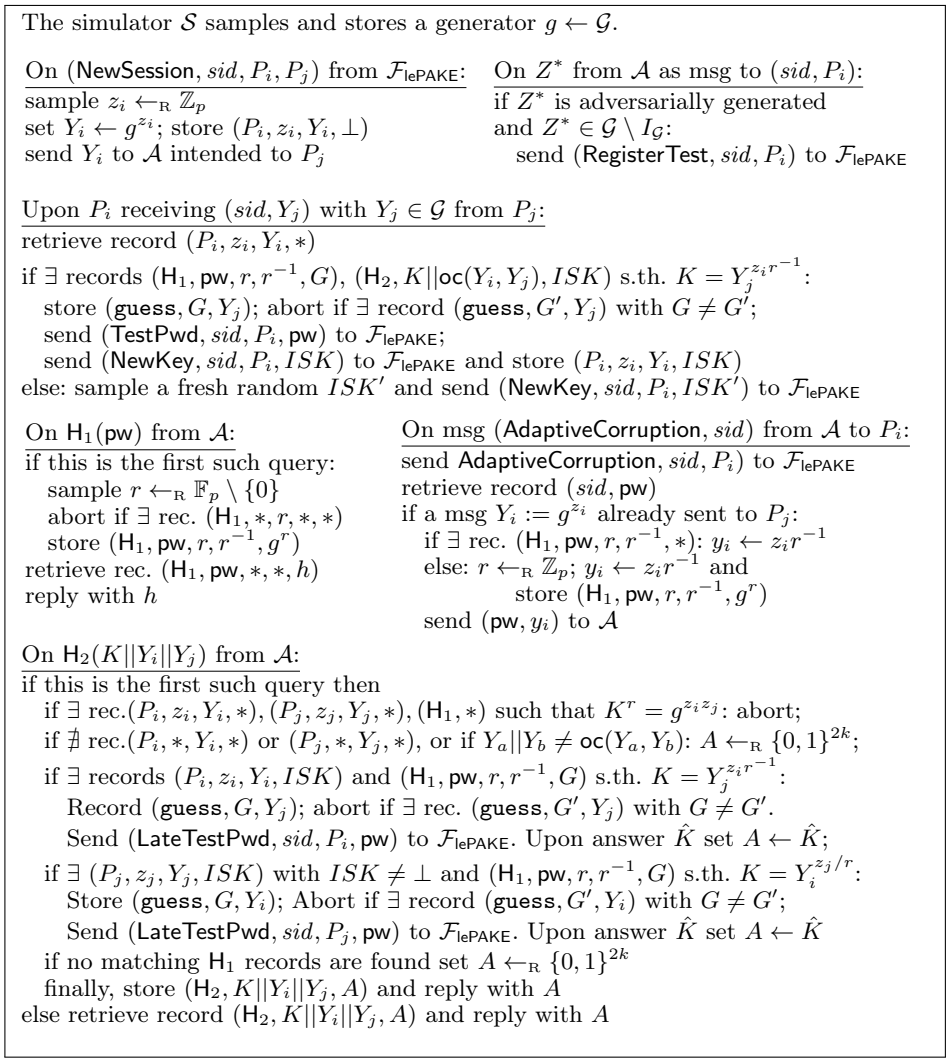


**Fig. 3.** Simulator for $\mathrm{CPace}_{\mathsf{base}}$. For brevity we omit the session identifier *sid* from all records stored by the simulator.

concludes simulation of honest parties without passwords. Adversarial password guesses can be read from $\mathcal{A}$ injecting $X$ (or, similarly, $Y$) and then querying $\mathsf{H}_2(K||X||Y)$ with $K$ being a correctly computed key w.r.t some generator $g^r$ provided by the simulation. $\mathcal{S}$ can now read the guessed password from the $\mathsf{H}_1$ list, and submit it as password guess to $\mathcal{F}_{\mathsf{lePAKE}}$. In case of success, the simulator sets the key of the honest party to $\mathsf{H}_2(K||X||Y)$.

The simulation is complicated by the order of honest parties' outputs (which are generated upon receipt of the single message) and the adversary's computa-

tion of final session keys via $\mathsf{H}_2$ queries. If the key is generated by $\mathcal{F}_{\mathsf{lePAKE}}$ before $\mathcal{A}$ computes it via $\mathsf{H}_2$ (which constitutes a password guess as detailed above), then $\mathcal{S}$ needs to invoke the $\mathsf{LateTestPwd}$ query of $\mathcal{F}_{\mathsf{lePAKE}}$ instead of $\mathsf{TestPwd}$. In case of a correct guess, this lets $\mathcal{S}$ learn the output key of the honest party, which $\mathcal{S}$ can then program into the corresponding $\mathsf{H}_2$ query.

Finally, the simulation can fail in some cases. Firstly, $\mathcal{S}$ might find more than one password guess against an honest party with simulated message $X$. In this case, the simulation cannot continue since $\mathcal{F}_{\mathsf{lePAKE}}$ allows for only one password guess per party. In this case, however, $\mathcal{A}$ would provide $(g^r, X, Y, K)$,$(g^{r'}, X, Y, K')$ which are two CDH tuples for passwords $\mathsf{pw}, \mathsf{pw}'$ with $g^r \leftarrow \mathsf{H}_1(\mathsf{pw}), g^{r'} \leftarrow \mathsf{H}_1(\mathsf{pw}')$. Provided that the simultaneous strong CDH assumption ($\mathsf{sSDH}$, cf. Definition 2) holds, this cannot happen. Here, the "strong" property, providing a type of DDH oracle, is required to help $\mathcal{S}$ identify CDH tuples among all queries to $\mathsf{H}_2$. A second case of simulation failure occurs when $\mathcal{A}$ wants to compute a key of an uncorrupted session via a $\mathsf{H}_2$ query. Since $\mathcal{S}$ does not know such keys, it would have to abort. Using a similar strategy as above, pseudorandomness of keys can be shown to hold under the strong CDH assumption, and thus the probability of $\mathcal{A}$ issuing such a $\mathsf{H}_2$ query is negligible. The full proof can be found in the full version of this work [4].

Our Theorem 2 demonstrates that adaptive security of CPace can be proven with only minimal information included in the hashes, i.e., the first hash requires only the password and the final key derivation hash requires the Diffie-Hellman key and the unique protocol transcript. We detail now under which circumstances additional data such as session identifiers needs to be included in the hashes. We further note that adding additional inputs to hashes, such as the name of a ciphersuite that an application wants parties to agree on, does not harm security.

*On multi-session security and hash domain separation* Theorem 2 demonstrates that $\mathsf{CPace}_{\mathsf{base}}$ allows to securely turn a joint password into one key. In practice, one would of course want to exchange more than one key, and many parties will end up using the same password. If session identifiers are globally unique, then the UC composition theorem (more detailed, the composition theorem of UC with Joint State [16]) allows to turn Theorem 2 into a proof of "multi-session CPace" by simply appending the unique session identifiers to all hash function inputs. This ensures that hash domains of individual sessions are separated and the programming activities of the individual simulators do not clash. To summarize, we obtain a secure multi-session version of CPace by ensuring uniqueness of session identifiers and including them in hashes. In practice, this can be ensured by, e.g., agreeing on a joint session identifier to which both users contributed randomness and in which party identifiers are incorporated (see, e.g., [6]). The agreement needs to happen before starting CPace, but does not require secrecy and can thus potentially be piggy-backed to messages sent by the application. As a last note, applications might choose to add more values to hashes, for example to authenticate addresses or to ensure agreement on a ciphersuite. We stress that

such additional values do not void our security analysis, but care still needs to be taken in order to protect against side-channel attacks.

### 5.1 Embedding CDH experiment libraries into the simulator

In this section, we discuss an alternative approach to carrying out reductions to cryptographic assumptions in the case of CPace/CDH. Both assumptions required by $\mathsf{CPace}_{\mathsf{base}}$, sCDH and sSDH, allow for an *efficient implementation* of experiments in the following sense: the secret exponents that are sampled by the experiment code (often also called the *challenger*) are sufficient for answering the restricted DDH queries allowed by both assumptions. An example for an assumption that does not allow for such efficient instantiation is, e.g., gap-CDH. In gap-CDH, the adversary is provided with a "full" DDH oracle that he can query on arbitrary elements, of which the experiment might not know an exponent for.

Due to this property, we can integrate implementations of the sCDH and sSDH experiments in the simulator's code. The simulator implements the DDH oracles on its own, and abort if at any time an oracle query solves the underlying assumption. We chose to integrate experiments as libraries (written as objects in python-style notation in Figure 4) into the simulator's code. This eases not only presentation but is also useful for analyzing variants of CPace that require slightly different assumptions.

The corresponding result for $\mathsf{CPace}_{\mathsf{base}}$ is shown in Fig. 5 when the challenge-generating experiment $\mathsf{exp} \leftarrow \mathsf{sSDH}(\mathsf{sCDH})$ is used (Fig. 4). The instance of the sSDH object first samples a random generator as member $s.g$ and creates a member instance $s.scdh \leftarrow \mathsf{sCDH}(g)$ of the experiment for the sCDH problem. The sCDH member object produces a challenge consisting of two uniformly drawn group elements $Y_1 \leftarrow g^{y_1}, Y_2 \leftarrow g^{y_2}$. The limited DDH oracle provided by the sCDH assumption can only receive inputs w.r.t one of these elements, and thus it can be implemented efficiently using secret exponents $y_1, y_2$. If a correct CDH solution $g, Y_1, Y_2, g^{y_1 \cdot y_2}$ is provided, the sCDH object aborts. In its implementation for $\mathsf{H}_1$, the sSDH object samples random new generators as $R \leftarrow (s.g)^r$ which will be used for simulating password-dependent base points and uses the $s.scdh$ member and the known exponent $r$ for answering DDH queries by use of the $s.scdh.\mathsf{DDH}$ function. The corrupt queries are implemented likewise and forwarded to the $s.scdh$ member object. The simulator from Fig. 3 is adapted to call the experiment. As an example, honest parties' messages are simulated by calling the challenge sampling procedure $\mathsf{exp.sampleY}()$ from sSDH which itself calls the corresponding function from its sCDH member.

*Proving indistinguishability.* With this simulation approach, a proof consists in demonstrating that ideal and real world executions are indistinguishable except for events in which the experiment libraries abort because a challenge was correctly answered. Compared to our proof of Theorem 2, the indistinguishability argument becomes simpler because the reduction strategies to both CDH-type assumptions are already embedded in the corresponding assumption experiment libraries. Losses such as the factor $2l_{\mathsf{H}_1}^2$ in the reduction to sSDH translate to libraries producing more than one challenge per simulation run, as is the case

```
# using python-style notation with self pointer s and _init_ constructor
def class sCDH:
  def _init_(s, g): s.g ← g; s.i ← 0; s.state← fresh;
  def sampleY(s):
    if s.i < 2: s.i+ = 1; sample s.y_i ←R F_p \ 0; return (s.g)^{s.y_i};
  def corrupt(s,X):
    for 1 ≤ m ≤ s.i:
      if (X = (s.g)^{s.y_m}): s.state← corrupt; return s.y_m;
  def DDH(s, g, Y, X, K):
    if (g ≠ s.g): return;
    if ({Y, X}={s.Y_1, s.Y_2}) and (s.state= fresh) and (K = (s.g)^{s.y_1·s.y_2}):
      abort("sCDH(g, Y_1, Y_2) solved")
    for 1 ≤ m ≤ s.i:
      if (Y = (s.g)^{s.y_m}): return (K = X^{s.y_m});
  def isValid(X): return (X ∈ G \ {I_G})

def class sSDH: # using python-style notation [ ] for list containers
  def _init_(s,sCdhExp): # Gets sCDH class; samples g; creates a sCDH instance
    s.g ←R G; s.scdh = sCdhExp(s.g); s.records =[ ]; s.guess = "yet no guess";
  def sampleY(s): return (s.scdh).sampleY();
  def isValid(s, X): return (s.scdh).isValid(X);
  def sampleH1(s):
    sample r ←R F_p \ {0};
    if (r, ∗) in s.records: abort("Hash to group collision");
    else: s.records.append((r, (s.g)^r)); return (s.g)^r;
  def corrupt(s, R, Y):
    if there is (r, R) in s.records: return (s.scdh).corrupt(Y^{1/r});
  def DDH(s,R, Y, X, K):
    if there is (r, R) in s.records:
      match ← (s.scdh).DDH(s.g, Y, X, K^{1/r});
      if match and (s.guess = "yet no guess"): (s.guess.g,s.guess.X)← (R, X);
      elif match and (s.guess.X = X) and (s.guess.g ≠ R):
        abort("sSDH problem (Y, R, s.guess.g) solved");
      return match;
```

**Fig. 4.** Libraries implementing sCDH and sSDH experiments.

for the sSDH experiment from Fig. 5. Altogether, the simulation with integrated CDH experiment libraries is an alternative approach of proving Theorem 2, as we formalize in the following.

**Theorem 3 (Alternative simulation for Theorem 2).** *The simulator depicted in Fig. 5 is a witness for the UC emulation statement in Theorem 2*

*Proof (Proof sketch.).* The output distribution of the simulator $\mathcal{S}$ from Fig. 5 is indistinguishable from the one of the simulator from Fig. 3 as it is obtained from internal restructuring. $\mathcal{S}$ aborts if either the sSDH or the sCDH experiment class aborts, which occurs iff a correct solution has been provided to the experiment

The simulator $\mathcal{S}(\mathsf{exp})$ is parametrized by an experiment class $\mathsf{exp}$.

On $(\mathsf{NewSession}, sid, P_i, P_j)$ from $\mathcal{F}_{\mathsf{lePAKE}}$:
set $Y_i \leftarrow \mathsf{exp.sampleY}()$;
store $(P_i, P_j, Y_i, \perp)$;
send $Y_i$ to $\mathcal{A}$ intended to $P_j$;

On $Z^*$ from $\mathcal{A}$ as msg to $(sid, P_i)$:
if $Z^*$ is adversarially generated
and $\mathsf{exp.isValid}(Z^*)$:
send $(\mathsf{RegisterTest}, sid, P_i)$ to $\mathcal{F}_{\mathsf{lePAKE}}$

Upon $P_i$ receiving $(sid, Y_j)$ from $P_j$: retrieve record $(P_i, *, z_i, Y_i, *)$
if not $\mathsf{exp.isValid}(Y_j)$: return;
if $\exists$ records $(\mathsf{H}_1, \mathsf{pw}, h)$, $(\mathsf{H}_2, K||(\mathsf{oc}(Y_i, Y_j), ISK)$
such that $\mathsf{exp.DDH}(h, Y_i, Y_j, K) = 1$:
send $(\mathsf{TestPwd}, sid, P_i, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{lePAKE}}$
send $(\mathsf{NewKey}, sid, P_i, ISK)$ to $\mathcal{F}_{\mathsf{lePAKE}}$ and store $(P_i, P_j, Y_i, ISK)$
else sample a fresh random $ISK'$ and send $(\mathsf{NewKey}, sid, P_i, ISK')$ to $\mathcal{F}_{\mathsf{lePAKE}}$
# $\mathcal{F}_{\mathsf{lePAKE}}$ will discard $ISK'$

On $\mathsf{H}_1(\mathsf{pw}))$ from $\mathcal{A}$:
if not $\exists$ record
$(\mathsf{H}_1, \mathsf{pw}, h)$:
$h \leftarrow \mathsf{exp.sampleH1}()$
store $(\mathsf{H}_1, \mathsf{pw}, h)$
lookup$(\mathsf{H}_1, \mathsf{pw}, h)$
reply with $h$

On $(\mathsf{AdaptiveCorruption}, sid)$ from $\mathcal{A}$ as msg to $P_i$:
Lookup $(P_i, P_j, Y_i, *)$; send $(\mathsf{AdaptiveCorruption}, sid, P_i)$
to $\mathcal{F}_{\mathsf{lePAKE}}$ and obtain $(sid, \mathsf{pw})$;
if a message $Y_i$ was already sent to $P_j$, then:
query $\mathsf{H}_1(\mathsf{pw})$ and
retrieve record $(\mathsf{H}_1, \mathsf{pw}, h)$
send $(\mathsf{pw}, \mathsf{exp.corrupt}(h, Y_i))$

On $\mathsf{H}_2(sid||K||Y_i||Y_j)$ from $\mathcal{A}$:
lookup $(\mathsf{H}_2, sid||K||Y_i||Y_j, h)$ and send $h$ if it exists;
else if this is the first such query:
if there are no records $(P_i, P_j, Y_i, *)$ or $(P_j, P_i, Y_j, *)$, or if $Y_a||Y_b \neq \mathsf{oc}(Y_a, Y_b)$:
sample $A \leftarrow \{0,1\}^{2k}$;
if $\exists$ records $(P_i, P_j, Y_i, ISK)$ with $ISK \neq \perp$ and $(\mathsf{H}_1, \mathsf{pw}, h)$
such that $\mathsf{exp.DDH}(h, Y_i, Y_j, K) = 1$:
send $(\mathsf{LateTestPwd}, sid, P_i, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{lePAKE}}$. Upon answer $\hat{K}$ set $A \leftarrow \hat{K}$
if $\exists$ records $(P_j, P_i, Y_j, ISK)$ with $ISK \neq \perp$ and $(\mathsf{H}_1, \mathsf{pw}, h)$
such that $\mathsf{exp.DDH}(h, Y_j, Y_i, K) = 1$:
send $(\mathsf{LateTestPwd}, sid, P_j, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{lePAKE}}$. Upon answer $\hat{K}$ set $A \leftarrow \hat{K}$
if no matching $\mathsf{H}_1$ records are found set $A \leftarrow \{0,1\}^{2k}$
finally, store $(\mathsf{H}_2, sid||K||Y_i||Y_j, A)$ and reply with $A$

**Fig. 5.** Generic simulator for different CPace variants, embedding challenges generated by the experiment object $\mathsf{exp}$. The simulator for $\mathrm{CPace}_{\mathsf{base}}$ is obtained when using $\mathsf{exp} \leftarrow \mathsf{sSDH}(\mathsf{sCDH})$ from Fig. 4.

implementation or a $\mathsf{H}_1$ collision is observed. These cases coincide with the abort cases in the proof of Theorem 2. As the $\mathsf{sSDH}$ object outputs $2l_{\mathsf{H}_1}^2$ different challenges and as it is sufficient for $\mathcal{Z}$ to provide a solution to one of these challenges for distinguishing both worlds, the advantage for solving the $\mathsf{sSDH}$ problem needs to be multiplied by this factor, thus reproducing the bounds from Theorem 2.

*Advantages of embedding libraries in the simulation.* To clarify, the approach presented in this section does *not* allow to prove stronger security statements. As demonstrated above, it is merely an alternative way of presenting security proofs in the UC framework or other simulation-based frameworks, and it works whenever the underlying cryptographic assumptions are efficiently implementable. However, we believe that the approach has its merits especially in the following dimensions.

- **Modular security analysis.** Slight modifications in the protocol might require to change the cryptographic assumption. As long as the public interface does not change, our approach allows to switch between assumptions by simply calling a different library. Cryptographers then need to only analyze this "local" change in the simulation, which prevents them from re-doing the whole indistinguishability argument.
- **Presentation of reduction strategies.** In normal game-based indistinguishability arguments [36], reductions to cryptographic assumptions are hidden within side-long proofs. With our approach, the reduction strategy is depicted in clear code as part of the simulator's code. This makes checking of proofs easier not only for readers but also might make simulation-based proofs more accessible to automated verification.

In this paper, our motivation is the first dimension. In the upcoming section, the library-based approach will turn out to be extremely useful to analyze the various variants of CPace that stem from (efficiency-wise) optimized implementations on different elliptic curves.

## 6 Analysis of Real-World CPace

The currently most efficient way to run CPace is over elliptic curves. Therefore, from this point onwards, we consider $\mathcal{G}$ to be an elliptic curve constructed over field $\mathbb{F}_q$. From a historical perspective, both CPace research and implementation first focused on prime order curves, such as the NIST-P-256 curve [18]. Subsequently significantly improved performance was shown on Montgomery- and (twisted-)Edwards curves, notably Curve25519 and Ed448 curves [10, 26], which both have a small cofactor $c$ in their group order $c \cdot p$. These approaches consider also implementation pitfalls, e.g., by designing the curve such that there are no incentives for implementers to use insecure speed-ups. Thirdly, recently ideal group abstractions have been presented in order to avoid the complexity of small cofactors in the group order [25, 17], while maintaining all advantages of curves with cofactor.

For smooth integration into each of these different curve ecosystems, CPace needs to be instantiated slightly differently regarding, e.g., computation of the DH generator, group size, multiplication and sampling algorithms. In this section, we analyze how such differences impact security. Using our modular approach with assumption libraries called by a simulator, we are able to present security in terms of differences from our basic CPace analysis in Section 5 in a concise way.

### 6.1 CPace without Hashing to the Group

| $\mathsf{CPace}_{\mathsf{1MAP}} := \mathsf{CPace}[\mathsf{Gen}_{\mathsf{1MAP}}, \mathsf{ScMul}_{\mathsf{base}}, \mathsf{ScMulVf}_{\mathsf{base}}, \mathsf{ScSam}_p]$ | | | |
|---|---|---|---|
| $\mathsf{Gen}_{\mathsf{1MAP}}(\mathsf{pw})$ : | $\mathsf{ScMul}_{\mathsf{base}}(g, y)$ : | $\mathsf{ScMulVf}_{\mathsf{base}}(g, y)$ : | $\mathsf{ScSam}_p()$ : |
| return $\mathsf{Map2Pt}(\mathsf{H}_1(\mathsf{pw}))$ | return $g^y$ | if $g \notin \mathcal{G}$: return $I_{\mathcal{G}}$<br>else: return $g^y$ | $y \leftarrow_{\mathrm{R}} 1 \dots p$<br>return $y$ |

**Fig. 6.** Protocol $\mathsf{CPace}_{\mathsf{1MAP}}$ for an elliptic curve group $\mathcal{G}$ of prime order $p$, over finite field $\mathbb{F}_q$. Generators are computed as $\mathsf{Map2Pt}(\mathsf{H}_1(\mathsf{pw}))$ with a hash function $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{F}_q$. Differences to $\mathsf{CPace}_{\mathsf{base}}$ are marked gray.

We now analyze a variant of the CPace protocol case-tailored for elliptic curve groups $\mathcal{G}$ over finite field $\mathbb{F}_q$. The protocol is depicted in Fig. 6. The only difference to $\mathsf{CPace}_{\mathsf{base}}$ analyzed in the previous section is how parties compute the generators: now the function $\mathsf{H}_1$ hashes onto the field $\mathbb{F}_q$, and generators are computed as $g \leftarrow \mathsf{Map2Pt}(\mathsf{H}_1(\mathsf{pw}))$ for a function $\mathsf{Map2Pt} : \mathbb{F}_q \to \mathcal{G}$. This way, the $\mathsf{H}_1$ outputs can be considered to form an alternative encoding of group elements, where $\mathsf{Map2Pt}$ decodes to the group. $\mathsf{ScMul}, \mathsf{ScMulVf}$ and $\mathsf{SamSc}$ are as in Section 5.

*Security analysis.* Compared to the analysis of $\mathsf{CPace}_{\mathsf{base}}$, the security analysis is complicated by the different computation of the generators in essentially two ways: first, the possibly non-uniform distribution of $\mathsf{Map2Pt}$ induces non-uniformity of DH generators computed by the parties. Second, embedding of trapdoors no longer works by simply programming elements with known exponents into $\mathsf{H}_1$. Instead, the proof will exploit that $\mathsf{Map2Pt}$ is probabilistically invertible, such that preimages of generators with known exponents can be programmed into $\mathsf{H}_1$ instead. Consequently, security of CPace will be based on the $\mathcal{D}_{\mathcal{G}} - \mathsf{sSDH}$ problem Definition 3 instead of the $\mathsf{sSDH}$ problem, where the distribution $\mathcal{D}_{\mathcal{G}}$ corresponds to the distribution of group elements $\mathsf{Map2Pt}(h_i)$ obtained for uniformly sampled field elements $h_i \leftarrow_{\mathrm{R}} \mathbb{F}_q$. All these changes can be captured by replacing library $\mathsf{sSDH}$ with a new library for $\mathcal{D}_{\mathcal{G}} - \mathsf{sSDH}$, as we demonstrate below.

**Theorem 4 (Security of $\mathsf{CPace}_{\mathsf{1MAP}}$).** *Let $\lambda, p, q \in \mathbb{N}$ with $p$ prime. Let $\mathcal{G}$ an elliptic curve of order $p$ over field $\mathbb{F}_q$. Let $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{F}_q, \mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^\lambda$ be two hash functions and $\mathsf{Map2Pt} : \mathbb{F}_q \to \mathcal{G}$ probabilistically invertible with bound $\mathsf{Map2Pt}.n_{max}$. If the $\mathsf{sCDH}$ and $\mathsf{sSDH}$ problems are hard in $\mathcal{G}$, then the CPace protocol depicted in Fig. 2 UC-emulates $\mathcal{F}_{\mathsf{lePAKE}}$ in the random-oracle model with respect to adaptive corruptions and both hash functions modeled as random oracles. More precisely, for every adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_{\mathsf{sSDH}}$ and $\mathcal{B}_{\mathsf{sCDH}}$ against the $\mathsf{sSDH}$ and $\mathsf{sCDH}$ problems such that*

$$
\begin{aligned}
&|Pr[Real(\mathcal{Z}, \mathcal{A}, CPace_{\mathsf{1MAP}})] - Pr[Ideal(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})]| \\
&\leq (\mathit{Map2Pt}.n_{max}) l_{\mathsf{H}_1}/q + (l_{\mathsf{H}_1})^2/p + (\mathit{Map2Pt}.n_{max} \cdot l_{\mathsf{H}_1})^2/q \\
&\quad + 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{\mathsf{sSDH}}_{\mathcal{B}_{\mathsf{sSDH}}}(\mathcal{G}) + \mathbf{Adv}^{\mathsf{sCDH}}_{\mathcal{B}_{\mathsf{sCDH}}}(\mathcal{G})
\end{aligned}
$$

*where $l_{\mathsf{H}_1}$ denotes the number of $\mathsf{H}_1$ queries made by the adversary $\mathcal{A}$ and the simulator $\mathcal{S}$ is as in Fig. 5 but using the object distExp (cf. Fig. 7) instead of the object sSdhExp.*

```
def class D_G _sSDH:
  def _init_(s, Map2Pt, sSDHExp):
    s.sSDH = sSDHExp; s.records = [];
    s.nmax = n_max; s.preim = Map2Pt.PreImages;
  def sampleY(s): return (s.sSDH).sampleY();
  def isValid(X): return (s.sSDH).isValid(X);
  def sampleH1(s):
    g ← (s.sSDH).sampleH1();
    while (1):
      r ←_R F_p; preimageList = (s.preim)(g^r); m ←_R {0 ... (s.Map2Pt.n_max − 1)};
      if len(preimageList) > m:
        if r = 0: abort("Sampled neutral element.");
        h ←preimageList[m];if h in s.records: abort("H1 collision");
        s.records.append(r, g^r, h); return h;
  def corrupt(s, h, Y):
    if there is (r, g, h) in s.records: return (s.sSDH).corrupt(g, Y^{1/r});
  def DDH(s,h, Y, X, K):
    if there is (r, g, h) in s.records: return (s.sSDH).DDH(g, Y, X, K^{1/r});
# Chaining the experiments for CPace on prime order curve, full (x,y) coordinates
sSdhExp = sSDH(sCDH);
distExp = D_G _sSDH(Map2Pt,sSdhExp);
```

**Fig. 7.** Experiment class definition $\mathcal{D}_{\mathcal{G}}$-sSDH using single executions of Map2Pt, where $\mathsf{H}_1$ hashes to $\mathbb{F}_q$.

*Proof (Proof Sketch.).* Let $\mathcal{D}_{\mathcal{G}}$ denote the distribution on $\mathcal{G}$ induced by Map2Pt. First note, that if the sSDH is hard in $\mathcal{G}$ then the corresponding $\mathcal{D}_{\mathcal{G}}$-sSDH problem is hard by Theorem 1 as $\mathsf{Map2Pt}^{-1}$ (implemented in the body of the sampleH1 method by the distExp object) is a rejection sampler for $\mathcal{D}_{\mathcal{G}}$.

We adjust the simulator for "basic" CPace from Fig. 5 as follows. First, we embed the reduction strategy from Theorem 1 into an experiment library that converts sSDH challenges into $\mathcal{D}_{\mathcal{G}}$ − sSDH challenges and obtain the class $\mathcal{D}_{\mathcal{G}}$_sSDH depicted in Fig. 7. The class $\mathcal{D}_{\mathcal{G}}$_sSDH uses the Map2Pt.PreImages function (passed as a constructor parameter) for implementing the $\mathsf{Map2Pt}^{-1}$ as defined in Algorithm 1 and an instance of the sSDH class implementing a sSDH experiment that is assigned to a member variable.

Each time the main body of the simulator from Fig. 5 makes calls to its exp object, the corresponding method of the new $\mathcal{D}_{\mathcal{G}}$_sSDH object will be executed, which itself translates the queries into calls to the sSDH object that was passed as constructor parameter.

Importantly, $\mathcal{D}_{\mathcal{G}}$_sSDH provides the same public API as the sSDH class with the distinction that sampling for $\mathsf{H}_1$ returns results from $\mathbb{F}_q$ instead of $\mathcal{G}$. Moreover $\mathcal{D}_{\mathcal{G}}$_sSDH aborts if the code of its sSDH object aborts and, now additionally, also upon $\mathsf{H}_1$ collisions.

We explain now how the indistinguishability argument of Theorem 2 needs to be adjusted in order to work for Theorem 4 and this new simulator. First, we ensure that the distribution of points provided by the $\mathcal{D}_{\mathcal{G}}$_sSDH object is uniform in $\mathbb{F}_q$ using Corollary 1. Second, we adjust the collision probability following the derivation from Corollary 2 which is now bound by $(n_{\max} \cdot l_{\mathsf{H}_1})^2/q$ in addition to the previous $l_{\mathsf{H}_1}^2/p$ probability. The probability that sampleH1 aborts because it samples the identity element from the distribution is bounded by $(\mathsf{Map2Pt}.n_{\max})l_{\mathsf{H}_1}/q$. Apart of these modification the proof applies without further changes.

*Instantiating Map2Pt.* Various constructions have been presented for mapping field elements to elliptic curve points such as Elligator2 [11], simplified SWU [20] and the Shallue-van de Woestijne method (SvdW) [35] (see also [20] and references therein). When considering short-Weierstrass representations of a curve, the general approach is to first derive a set of candidate values $x_l$ for the $x$ coordinate of a point such that for at least one of these candidates $x_l$ there is a coordinate $y_l$ such that $(x_l, y_l)$ is a point on the curve. Subsequently one point $(x_l, y_l)$ is chosen among the candidates. The property of *probabilistic invertibility* is fulfilled for all of the algorithms mentioned above and those currently suggested in [20]. The most generic of these algorithm, SvdW, works for all elliptic curves, while the simplified SWU and Elligator2 algorithms allow for more efficient implementations given that the curve fulfills some constraints.

All these mappings have a fixed and small bound $n_{\max}$ regarding the number of pre-images and come with a PPT algorithm for calculating all preimages. For instance, Elligator2 [11] comes with a maximum $n_{\max} = 2$ of two pre-images per point and $n_{\max} \leq 4$ for the simplified SWU and SvdW algorithms [20]. For all these algorithms, the most complex substep for determining all preimages is the calculation of a small pre-determined number of square roots and inversions in $\mathbb{F}_q$ which can easily be implemented in polynomial time with less computational complexity than one exponentiation operation.

### 6.2 Considering curves with small co-factor

In this subsection, we now additionally consider that the elliptic curve group $\mathcal{G}$ can be of order $c \cdot p$ with $c \neq 1$, but where Diffie-Hellman-type assumptions can only assumed to be computationally infeasible in the subgroup of order $p$, denoted $\mathcal{G}_p$. Consequently, $\mathsf{CPace}_{\mathsf{co}}$ on curves with co-factor $c \neq 1$ requires all

| $\mathsf{CPace_{co}}$ :=$\mathsf{CPace}[\mathsf{Gen_{1MAP}}, \mathsf{ScMul_{co}}, \mathsf{ScMulVf_{co}}, \mathsf{ScSam}_p]$ | | | |
|---|---|---|---|
| $\mathsf{Gen_{1MAP}}(\mathsf{pw})$ : | $\mathsf{ScMul_{co}}(g, y)$ : | $\mathsf{ScMulVf_{co}}(g, y)$ : | $\mathsf{ScSam}_p()$ : |
| return $\mathsf{Map2Pt}(\mathsf{H}_1(\mathsf{pw}))$ | return $g^{c \cdot y}$ | if $g \notin \mathcal{G}$: return $I_{\mathcal{G}}$ | $y \leftarrow_{\mathrm{R}} 1 \ldots p$ |
| | | else: return $g^{c \cdot y}$ | return $y$ |

**Fig. 8.** Definition of $\mathsf{CPace_{co}}$ for curves of order $p \cdot c$. The only difference (marked gray) to $\mathsf{CPace_{1MAP}}$ is that exponents are always multiplied by the cofactor $c$.

secret exponents to be multiples of $c$. Hence, $\mathsf{CPace_{co}}$ depicted in Fig. 8 deploys modified algorithms $\mathsf{ScMul}, \mathsf{ScMulVf}$.

```
# using python-style notation with self pointer s
def class cofactorClearer:
  "interfaces S to a prime-order experiment class"
  def _init_(s, c, p, primeOrderExpInstance ,p̄):
    s.c=c; s.i= s.c · integer(1/(s.c^2) mod p);s.it= s.c · integer(1/(s.c^2) mod p̄);
    s.exp = primeOrderExpInstance;
  def sampleY(s): return ((s.exp).sampleY())^{s.c};
  def isValid(X): return (s.exp).isValid(X^{s.i})
  def sampleH1(s): return (s.exp).sampleH1();
  def corrupt(s, h, Y): { return (s.exp).corrupt(h, Y^{s.i}); }
  def DDH(s,g, Y, X, K):
    if X ∈ G:  return (s.exp).DDH(g, Y^{s.i}, X^{s.i}, K^{s.i·s.i})
    if X on twist: return (s.exp).DDH(g, Y^{s.i}, X^{s.it}, K^{s.it·s.it})

sSdhExp = sSDH(sCDH);  ccExp = cofactorClearer(sSdhExp);
ccDistExp = D_G _sSDH(Map2Pt,ccExp);
```

**Fig. 9.** Cofactor-clearer class definition use for elliptic curves of order $p \cdot c$ with a quadratic twist having a subgroup of order $\bar{p}$. Note that the inverses $s.i$ and $s.it$ are constructed such that they are multiples of $c$.

**Theorem 5 (Security of $\mathsf{CPace_{co}}$).** *Let $\lambda, p, q, c \in \mathbb{N}$, $p, c$ coprime with $p$ prime. Let $\mathcal{G}$ be an elliptic curve of order $p \cdot c$ over field $\mathbb{F}_q$ and $\mathcal{G}_p \subset \mathcal{G}$ a subgroup of order $p$. Let $CC_c : (g) \mapsto ((g^c)^{1/c \bmod p})$ be a cofactor clearing function for $c$, $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{F}_q, \mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^\lambda$ be two hash functions and $\mathsf{Map2Pt} : \mathbb{F}_q \to \mathcal{G}$ probabilistically invertible with bound $\mathsf{Map2Pt}.n_{max}$. Let be the chained function $\mathsf{Map2Pt}_{\mathcal{G}_p} := (CC_c \circ \mathsf{Map2Pt})$. Let $\mathcal{D}_{\mathcal{G}_p}$ denote the distribution on $\mathcal{G}_p$ induced by $\mathsf{Map2Pt}_{\mathcal{G}_p}$. If the sCDH and sSDH problems are hard in $\mathcal{G}_p$, then the $\mathcal{D}_{\mathcal{G}_p}$-sSDH problem is hard in $\mathcal{G}_p$ and $\mathsf{CPace_{co}}$ UC-emulates $\mathcal{F}_{\mathsf{lePAKE}}$ in the random-oracle model with respect to adaptive corruptions when both hash functions are modeled as random oracles. More precisely, for every adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}_{\mathsf{sCDH}}$ and $\mathcal{B}_{\mathsf{sSDH}}$ against the sCDH and sSDH problems such that*

$$
\begin{aligned}
&|Pr[Real(\mathcal{Z}, \mathcal{A}, CPace_{\mathsf{co}})] - Pr[Ideal(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})]| \\
&\leq (\mathit{Map2Pt}.n_{max} \cdot c) l_{\mathsf{H}_1}/q + 2l_{\mathsf{H}_1}^2/p + (\mathit{Map2Pt}.n_{max} \cdot c \cdot l_{\mathsf{H}_1})^2/q \\
&\quad + 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{\mathit{sSDH}}_{\mathcal{B}_{\mathit{sSDH}}}(\mathcal{G}_p) + \mathbf{Adv}^{\mathit{sCDH}}_{\mathcal{B}_{\mathit{sCDH}}}(\mathcal{G}_p)
\end{aligned}
$$

*where $l_{\mathsf{H}_1}$ denotes the number of $\mathsf{H}_1$ queries made by the adversary $\mathcal{A}$ and the simulator $\mathcal{S}$ is as in Fig. 5 but using class ccDistExp (cf. Fig. 9) instead of object sSdhExp.*

*Proof (Proof Sketch.).* The full group $\mathcal{G}$ has a point $g_1$ of order $c$ with $g_1^c = I_{\mathcal{G}}$ where $I_{\mathcal{G}}$ denotes the identity element in $\mathcal{G}$, i.e., there are $c$ low-order points $g_1^i, i \in \{1 \dots c\}$. For any point $Y \in \mathcal{G}$ we can consider the points $Y_i = Y \cdot g_i$ as alternative ambiguous representations of the point $CC_c(Y) \in \mathcal{G}_p$. For any input point $Y \in \mathcal{G}_p$, all these $c$ alternative representations can be easily calculated using group operations and $g_i$. For any of these $c$ alternative representations of $Y$ at most $\mathsf{Map2Pt}.n_{\max}$ preimages will be returned by $\mathsf{Map2Pt.PreImages}$ since $\mathsf{Map2Pt}$ is probabilistically invertible on $\mathcal{G}$. Correspondingly, the probability of accidently drawing a representation of the identity element needs to be multiplied by $c$ and is now bounded by $(\mathsf{Map2Pt}.n_{\max} \cdot c) l_{\mathsf{H}_1}/q$. If up to $\mathsf{Map2Pt}.n_{\max}$ preimages exist per point on the full curve, the chained function $\mathsf{Map2Pt}_{\mathcal{G}_p}$ is probabilistically invertible also on $\mathcal{G}_p$. Its preimage function $\mathsf{Map2Pt}_{\mathcal{G}_p}.\mathsf{PreImages}$ for $\mathcal{G}_p$ can be defined such that it returns all of the preimages of the $c$ ambiguous representations of an input and the maximum number of preimages $\mathsf{Map2Pt}_{\mathcal{G}_p}.n_{\max}$ is, thus, bounded by $\mathsf{Map2Pt}_{\mathcal{G}_p}.n_{\max} = c \cdot \mathsf{Map2Pt}.n_{\max}$. Since we are able to provide all preimages for $\mathsf{Map2Pt}_{\mathcal{G}_p}$ and a bound for their number is known $\mathsf{Map2Pt}_{\mathcal{G}_p}$ is probabistically invertible. We thus can employ Theorem 1 and show that if the $\mathsf{sSDH}$ is hard in $\mathcal{G}_p$ then the corresponding $\mathcal{D}_{\mathcal{G}_p}$-$\mathsf{sSDH}$ problem is also hard.

As $\mathsf{ScMulVf}_{co}$ and $\mathsf{ScMul}_{co}$ use exponents that are a multiples of $c$ they are guaranteed to produce a unique result on $\mathcal{G}_p$ for all of the $c$ ambiguous representations of an input point. The additional factor of $c$ in the exponents is compensated by the simulation by calling an experiment library using the ccExp class from Fig. 9. [7] The ccExp object forwards queries to a $\mathcal{D}_{\mathcal{G}_p}$_$\mathsf{sSDH}$ object such that all inputs to the $\mathsf{DDH}$ oracle will be in $\mathcal{G}_p$.

### 6.3 CPace using single-coordinate Diffie-Hellman

Some Diffie-Hellman-based protocols, including CPace, can be implemented also on a group modulo negation, i.a. a group where a group element $Y$ and its inverse $Y^{-1}$ (i.e. the point with $I = Y \cdot Y^{-1}$) are not distinguished and share the same binary representation [8].

[7] Note that this class also accepts points on the quadratic twist, a feature that will become relevant only when considering simplified point verification on twist-secure curves as discussed in the full version of this paper [4].

[8] Counter-examples for protocols that cannot be instantiated on a group modulo negation and require full group structure are, e.g., TBPEKE [34] and SPAKE2 [5]. The reason is that these protocols require addition of arbitrary points on the group.

```
def class moduloNegationAdapter:
  "uses the strip- and reconstruct functions SC and RC."
  def _init_(s,, baseExperiment):
    s.exp ←baseExperiment;s.records← [];
  def sampleY(s): Y ← ((s.exp).sampleŶ())^{s.c}; s.records.append(Y); return SC(Y);
  def isValid(X̂):(X_0, X_1)←RC(X̂); return (s.exp).isValid(X_0);
  def sampleH1(s): return (s.exp).sampleH1();
  def corrupt(s, h, Ŷ):
    (Y, Y*) ← RC(Ŷ); if Y* in s.records: Y ← Y*; return (s.exp).corrupt(h, Y);
  def DDH(s,g, Ŷ, X̂, K̂):
    (Y, Y*) ← RC(Ŷ); if Y* in s.records: Y ← Y*;
    (X, X*) ← RC(X̂); (K, K*) ← RC(K̂);
    return (s.exp.DDH(g, Y, X, K)) or (s.exp.DDH(g, Y, X, K*))

# Chaining the experiments for prime order curve, single coordinate, single map
sSdhExp = sSDH(sCDH); distExp = D_G _sSDH(Map2Pt,sSdhExp);
singleCoorExp = moduloNegationAdapter(distExp)
```

**Fig. 10.** Single-coordinate experiment class definition for CPace instantiations on groups modulo negation.

An elliptic curve in Weierstrass representation becomes a group modulo negation when only using x-coordinates as representation. We use the notation $\hat{Y}$ for such ambiguous encodings and use $\hat{Y} \leftarrow \mathsf{SC}(Y)$ for a function returning the x-coordinate for a point $Y$ and $(Y^{-1}, Y) \leftarrow \mathsf{RC}(\hat{Y})$ for the inverse operation reconstructing $Y$ and $Y^{-1}$ in an undefined order.

The major advantage of using this type of ambiguous encoding is that it can be helpful in practice for all of the following: reducing code size, reducing public key sizes and network bandwidth, avoiding implementation pitfalls [10] and restricting invalid curve attacks to the curve's quadratic twist. Consequently, many real-world protocols such as TLS only use this single coordinate for deriving their session key, as to give implementers the flexibility to take benefit of the above advantages.

For the purpose of function definitions by chaining, we introduce a function $\mathsf{RSC}(\hat{Y}, x)$ that takes one ambigously encoded group element $\hat{Y}$ in addition to one scalar $x$, i.e. takes the same operands as $\mathsf{ScMul}$. We define $\mathsf{RSC}(\hat{Y}, x)$ such that it returns a tuple $(Y, x)$ such that $\mathsf{SC}(Y) = \hat{Y}$. With this definition, we can formalize CPace using single-coordinate scalar multiplications with the chained functions $\mathsf{ScMul}_{x-only} := (\mathsf{SC} \circ \mathsf{ScMul} \circ \mathsf{RSC})$, $\mathsf{ScMulVf}_{x-only} := (\mathsf{SC} \circ \mathsf{ScMulVf} \circ \mathsf{RSC})$ and $\mathsf{Gen}_{x-only} := \mathsf{SC} \circ \mathsf{Gen}$, such that the ambiguous encodings are used.[9]

**Theorem 6 (Security of $\mathsf{CPace}_{\mathsf{x-only}}$).** *Given a group $\mathcal{G}$, assume CPace[Gen, ScMul, ScMulVf, SamSc] on $\mathcal{G}$ can be distinguished from an ideal-world run of $\mathcal{F}_{\mathsf{lePAKE}}$ and $\mathcal{S}$ from Fig. 5 with negligible advantage, where $\mathcal{S}$ embeds*

[9] Note that this definition obtained from chaining with SC and RSC for the scalar multiplications corresponds exactly to the conventional so-called single-coordinate ladder algorithms.

*an experiment object* exp*. Then CPace*$[\mathsf{SC} \circ \mathsf{Gen}, \mathsf{SC} \circ \mathit{ScMul} \circ \mathsf{RSC}, \mathsf{SC} \circ \mathit{ScMulVf} \circ \mathsf{RSC}, \mathit{SamSc}]$ *on the corresponding group modulo negation* $\hat{\mathcal{G}}$ *cannot be distinguished from* $\mathcal{F}_{\mathsf{lePAKE}}$ *running with a simulator* $\hat{\mathcal{S}}$ *that is obtained by chaining* exp *with* moduloNegationAdapter*, the adapter class from Fig. 10, and the difference in the distinguishing advantage is bounded by a factor of 2.*

We defer the proof sketch to the full version of this paper [4]. With our library-based approach to simulation, it is also possible to argue security of CPace variants which combine several of the aspects above. In a nutshell, this works by chaining of the experiment classes. We refer the reader to the full version [4] for details and examples.

## References


1. Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020.
2. Michel Abdalla, Manuel Barbosa, Jonathan Katz, Julian Loss, and Jiayu Xu. Algebraic adversaries in the universal composability framework. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, LNCS. Springer, Heidelberg, December 2021.
3. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001.
4. Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. Cryptology ePrint Archive, Report 2021/114, 2021. https://eprint.iacr.org/2021/114.
5. Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, February 2005.
6. Boaz Barak, Yehuda Lindell, and Tal Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004. https://eprint.iacr.org/2004/006.
7. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.
8. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.
9. Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009*, volume 5735 of *LNCS*, pages 33–48. Springer, Heidelberg, September 2009.
10. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

11. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013.
12. Daniel J. Bernstein and Tanja Lange. SafeCurves: Choosing safe curves for elliptic-curve cryptography. Definition of Twist security. (accessed on 15 January 2019), 2019. https://safecurves.cr.yp.to/twist.html.
13. Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.
14. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
15. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.
16. Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, August 2003.
17. H. de Valence, J. Grigg, G. Tankersley, F. Valsorda, I. Lovecruft, and M. Hamburg. The ristretto255 and decaf448 groups. Rfc, IRTF, 10 2020.
18. Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST), FIPS PUB 186-4, U.S. Department of Commerce, July 2013. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.
19. Edward Eaton and Douglas Stebila. The "quantum annoying" property of password-authenticated key exchange protocols. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, volume 12841 of *LNCS*, pages 154–173. Springer, Heidelberg, July 2021.
20. A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves, 2019. https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/.
21. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.
22. Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.
23. Björn Haase. CPace, a balanced composable PAKE, 2020. https://datatracker.ietf.org/doc/draft-haase-cpace/.
24. Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286, 2018. https://eprint.iacr.org/2018/286.
25. Mike Hamburg. Decaf: Eliminating cofactors through point compression. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 705–723. Springer, Heidelberg, August 2015.
26. Mike Hamburg. Ed448-goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. https://eprint.iacr.org/2015/625.

27. Mike Hamburg. Indifferentiable hashing from elligator 2. Cryptology ePrint Archive, Report 2020/1513, 2020. https://eprint.iacr.org/2020/1513.
28. Julia Hesse. Review of (security of) remaining candidates. Posting to the CFRG mailing list, 2020. https://mailarchive.ietf.org/arch/msg/cfrg/47pnOSsrVS8uozXbAuM-alEk0-s/.
29. Julia Hesse. Separating symmetric and asymmetric password-authenticated key exchange. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 579–599. Springer, Heidelberg, September 2020.
30. David P. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996.
31. Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018.
32. A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748, IETF, January 2016.
33. Advanced security mechanism for machine readable travel documents (extended access control (EAC), password authenticated connection establishment (PACE), and restricted identification (RI)). Federal Office for Information Security (BSI), BSI-TR-03110, Version 2.0, 2008.
34. David Pointcheval and Guilin Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17*, pages 301–312. ACM Press, April 2017.
35. Andrew Shallue and Christiaan E. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2006.
36. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://eprint.iacr.org/2004/332.
37. Björn Tackmann. Updated review of PAKEs. Posting to the CFRG mailing list, 2020. https://mailarchive.ietf.org/arch/msg/cfrg/eo8O6JYPmWY6L9TlcIXStFy5gNQ/.