An Efficient Strong Asymmetric PAKE Compiler Instantiable from Group Actions

Ian McQuoid and Jiayu Xu

Oregon State University, {mcquoidi,xujiay}@oregonstate.edu

Abstract. Password-authenticated key exchange (PAKE) is a class of protocols enabling two parties to convert a shared (possibly low-entropy) password into a high-entropy joint session key. Strong asymmetric PAKE (saPAKE), an extension that models the client-server setting where servers may store a client's password for repeated authentication, was the subject of standardization efforts by the IETF in 2019–20. In this work, we present the most computationally efficient saPAKE protocol so far: a compiler from PAKE to saPAKE which costs only 2 messages and 7 group exponentiations in total (3 for client and 4 for server) when instantiated with suitable underlying PAKE protocols. In addition to being efficient, our saPAKE protocol is conceptually simple and achieves the strongest notion of universally composable (UC) security.
In addition to classical assumptions and classical PAKE, we may instantiate our PAKE-to-saPAKE compiler with cryptographic group actions, such as the isogeny-based CSIDH, and post-quantum PAKE. This yields the first saPAKE protocol from post-quantum assumptions as all previous constructions rely on cryptographic assumptions weak to Shor's algorithm.

1 Introduction

Password-authenticated key exchange (PAKE) [10] constitutes a class of protocols allowing two parties to compute a shared cryptographic key exactly when both parties hold the same (possibly low-entropy) input string, *i.e.*, a "password". Integrally, PAKE is in the "password-only" setting and does not rely on key distribution through *e.g.*, a PKI as trusted infrastructure may not always be available and additionally has a history of insecurities. Traditionally, passwords were assumed to come from some low-entropy distribution. This modeling represented the distribution of human-memorable inputs which have been estimated to only have 30 bits of entropy [27]. Because such passwords come from low-entropy or enumerable distributions, an adversary may always impersonate an honest party and perform a series of online invocations of the protocol to learn the password with non-negligible probability. A key property of PAKE is that this inevitable method should be the *only* efficient attack — with multiple guesses from a single interaction computationally infeasible. A shortcoming of PAKE is that it only models symmetric roles and precludes inherent asymmetries in the client-server setting. As passwords continue to be the most common

form of client-server authentication on the internet, it is incumbent on us to model password storage and, indeed, compromise of the server.

Asymmetric PAKE (aPAKE) [11,26] is a PAKE variant modeling the client-server setting where the client inputs its password pw in the clear and the server inputs a one-way function or *digest* of the password $F(pw)$. The parties then arrive at the same key if and only if the client supplies the preimage of the server's digest. Asymmetric PAKE is comparable to the model of a server storing *publicly* salted hashes of the user's password, and is vulnerable to pre-computation attacks where an adversary may pre-compute possible password files $(pw, F(pw))$ the server *could* hold and, on compromise, recover the password almost immediately by checking the server's storage against the pre-computed table to test each of its guesses.

To prevent pre-computation attacks, Jarecki, Krawczyk and Xu [31] introduced the concept of *strong* asymmetric PAKE (saPAKE), a variant of aPAKE where an adversary must spend time proportional to the number of password guesses made *after compromising the server* in order to recover the password. This is analogous to the server's password salt being *private*, and achieves the original intention of aPAKE — a PAKE that is resilient to (adaptive) server compromise. Note that as the server holds enough information to verify itself to a client, there exists an inevitable attack on the server's storage: the adversary may *locally* run the online protocol acting as both the client and as the server, testing the equality of the two output keys. Such an attack has runtime linear in the number of offline password guesses made. Strong asymmetric security guarantees that (asymptotically) this is the best possible complexity by showing a tight lower-bound on the complexity of a *post-compromise attack*.

Recent years have witnessed increasing interests in (sa)PAKE including the standardization efforts by the IETF in 2019–20. Despite this, saPAKE protocols have proven difficult to construct, and all existing protocols suffer from issues regarding either security or efficiency (see Section 1.2 for a detailed discussion). Furthermore, while resilience to quantum adversaries is a major concern of the PAKE community, all known constructions are based on classical assumptions and are easily broken by quantum adversaries. During the IETF's PAKE standardization process, the notion of "quantum-annoyingness" was proposed [41] and was subsequently formalized by Eaton and Stebila [23]. Roughly, a PAKE scheme is quantum-annoying if solving a discrete logarithm or integer factorization problem does not subsequently break the entire session; rather, each solution only allows the adversary an additional password guess. While quantum-annoyingness is a good stepping stone to protecting saPAKE protocols against quantum adversaries, it remains a major open problem in the area of PAKE to construct an saPAKE protocol under post-quantum assumptions.

1.1 Our Contributions

In this paper, we propose two compilers from PAKE to saPAKE. We prove the security of both compilers in the Universal Composability (UC) framework; concretely, both compilers realize the standard UC saPAKE functionality [31,15] if

the underlying PAKE protocol realizes the standard UC PAKE functionality [17]. UC-security for (sa)PAKE has superseded traditional game-based definitions due to multiple advantages, including security under arbitrary composition, modeling of adversarially-chosen password distribution, and therefore modeling password reuse across different accounts.

The first of our protocols (see Section 3) works in a cryptographic group. It is a compiler which adds only 3 exponentiations and 1 message — which can be sent in parallel with the server's PAKE message — on top of PAKE. For instantiations of UC PAKE such as certain variants of encrypted key exchange (EKE) [24,34] this results in the most efficient saPAKE protocols to date with only 2 messages and 7 exponentiations. In addition to being computationally efficient and conceptually simple, the resulting saPAKE inherits the underlying PAKE's quantum-annoying property. The only other quantum-annoying saPAKE protocol known is CRISP [21] which additionally relies on bilinear pairings. Our protocol is proven UC-secure in the generic group model (GGM) for offline security and the algebraic group model (AGM) plus the discrete logarithm (DL) assumption for online security, in addition to the random oracle model (ROM). We note that using the GGM for offline security but not online security is standard for saPAKE protocols whose server storage is group-based [15,21]; see Section 2.5 for a detailed explanation.

The second compiler (see Section 5.2) follows the same formula as the first, but can be instantiated with cryptographic group actions [6,20,36] such as isogeny-based assumptions like CSIDH [18]. Just as the first, the second compiler only costs 3 group actions and 1 message from the server on top of PAKE. When compiling from PAKE protocols based on lattices [24], we obtain the first efficient saPAKE protocol from post-quantum assumptions realizing the standard UC functionality. Additionally, even if the assumptions on the group action fail, our protocol still provides (symmetric) PAKE security as the server's message is independent of the password. This allows our protocol to use newer assumptions like CSIDH without future breakthroughs in isogeny cryptanalysis completely invalidating security. Our protocol is UC-secure in the generic group action model with twists ($\text{GGAM}^\top$) for offline security and the algebraic group action model with twists ($\text{AGAM}^\top$) plus the group-action discrete logarithm (GA-DL) assumption for online security, in addition to the ROM. One caveat is that our protocol is proven secure under *post-quantum assumptions*, but not against *quantum adversaries*, as that would require a security analysis in the quantum-accessible random oracle model (QROM) [13], which we do not consider. We leave constructing an saPAKE secure against quantum adversaries as future work.[1]

[1] Constructing a UC-secure saPAKE in the QROM seems out of reach, since there has been very little work that considers a quantum adversary in the UC framework [42], and we do not know of any protocol (PAKE or not) that is proven UC-secure in the QROM.

1.2 Comparison with Previous Results

To date, there are five known saPAKE protocols. The first two, due to Jarecki, Krawczyk, and Xu, come from the original saPAKE paper [31] and are constructed from a MitM-secure oblivious pseudorandom function (OPRF) and compile (respectively) an aPAKE or authenticated key exchange (AKE) into an saPAKE protocol. The second compiler, called OPAQUE, was standardized by the IETF in 2020 [14], and when based on classical assumptions, the resultant protocols are efficient (their costs can be found in Table 1). However, OPAQUE only realizes a weak UC saPAKE functionality (which we call $\mathcal{F}^{-}_{\text{saPAKE}}$), which includes two significant and contrived relaxations of the standard saPAKE functionality:

- $\mathcal{F}^{-}_{\text{saPAKE}}$ allows for *delayed extraction*, namely the ideal adversary's password guess can happen *even after the session completes*. The password guess interface is meant to model the real-world scenario where the adversary runs an honest party's algorithm on a candidate password and interacts with the other honest party in order to test if the candidate password is the correct one. Clearly, such attacks *cannot be carried out* after the session already ends.[2]
- In $\mathcal{F}^{-}_{\text{saPAKE}}$, compromising one open (*i.e.*, not completed) session automatically results in compromising *all other* open sessions without any additional command from the ideal adversary. This significantly weakens saPAKE security: consider a MitM adversary that attempts to attack two open sessions in parallel, the first of which fails and the second of which succeeds (*i.e.*, resulting in a compromised session). In the real world, it should not be the case that the first session — in which *the wrong password guess is already used* — can be compromised, let alone compromised without any additional work from the adversary. However, this is exactly what $\mathcal{F}^{-}_{\text{saPAKE}}$ allows!

We stress that neither weakening above is inherent or "natural". Few PAKE and aPAKE protocols that were proven UC-secure need the first relaxation, and *none of them* needs the second; for saPAKE, two of the other three existing protocols (see below) need neither of the two relaxations. In fact, the two relaxations appear to be the result of tailoring the UC saPAKE functionality to fit the OPAQUE protocol, as evident in [31, p.12]:

> *In our context, either requirement prevents proving security of the protocols obtained via our general compiler [...], including the OPAQUE protocol [...]. For this reason we relax [$\mathcal{F}_{saPAKE}$] to obtain our definition of UC Strong aPAKE functionality [$\mathcal{F}^{-}_{saPAKE}$].*

By contrast, our protocol realizes the standard UC saPAKE functionality without either of the two relaxations.[3]

[2] This change is similar in spirit to the $\mathcal{F}_{\text{rPAKE}}$ functionality (r for "relaxed") for symmetric PAKE in [1].

[3] The other saPAKE construction in [31], presented as a warm-up, seems to require these two relaxations as well, and is much less efficient. An accurate comparison is difficult since the security proof of that protocol is outdated (see [31, Section 4]).

Finally, we note that the assumptions in OPAQUE and in our protocol are incomparable: the security of OPAQUE relies on the very strong one-more gap Diffie-Hellman (OMGDH) assumption, which has been proven in the GGM but is not equivalent to DL in the AGM [9].[4] By contrast, the online part of our protocol is proven secure in the AGM+DL, without relying on any "one-more" type assumptions.

We now compare our saPAKE protocol with the other three saPAKEs in the literature. The third protocol called strong AuCPace, due to Haase and Labrique [29], is very similar in spirit to the both previous compilers and runs a modified version of OPRF before a (symmetric) PAKE protocol. Our compiler and their protocol follow the same intuition compiling a PAKE by way of a sub-session-specific dictionary map using an ephemeral salt. Where the strong AuCPace protocol communicates the server's long-term salt by way of an OPRF, we simply blind the server's long-term salt with a random exponent. This alteration allows our compiler to use fewer exponentiations. Furthermore, strong AuCPace only realizes a weaker UC saPAKE functionality (which requires the first relaxation of the full $\mathcal{F}_{\text{saPAKE}}$ functionality as in $\mathcal{F}^{-}_{\text{saPAKE}}$, but not the second).

The fourth protocol, due to Bradley, Jarecki, and Xu [15], follows the "commit-and-SPHF" paradigm [32] of PAKE design and realizes the full $\mathcal{F}_{\text{saPAKE}}$ functionality. However, their protocol requires roughly *three times* as many exponentiations as ours; furthermore, it is unclear if their protocol can be converted to use post-quantum assumptions. We also note that their analysis of offline security is similar to ours, but our analysis is much more accurate; see Section 2.4 for a detailed explanation.

The final protocol is CRISP due to Cremers *et al.* [21] which is conceptually similar to our protocol and compiles a PAKE protocol into a strong identity-binding PAKE (siPAKE) — a stronger primitive than saPAKE. However, the CRISP compiler critically relies on bilinear pairings which increases the computational and communication burden of the protocol while restricting the groups over which we can implement the compiler. Further, this reliance on pairings means the CRISP compiler has no post-quantum instantiation.

Regarding security assumptions, the offline security analyses of both [15] and [21] rely on the GGM as we do ([21] additionally requires the GGM for a bilinear group with a hash-to-group operation), but their online security is based on standard group assumptions. By contrast, our protocol needs the stronger online AGM (plus the DL assumption). To the best of our knowledge, this is the first instance of applying the AGM to the UC framework since the original work on UC-AGM [2]. The security of [29] relies on the ROM and a number of strong and non-standard group assumptions such as strong simultaneous CDH (sSDH) [5].

[4] [9, Section 10] shows that the one-more discrete logarithm (OMDL) assumption, which is weaker than OMGDH, cannot be proven equivalent to DL in the AGM.

	client	server	rounds	security	assumption	model
CKEM-saPAKE [15]	13E	8E	2	full	2-SDH, DDH	ROM+off GGM
OPAQUE HQMV [31]	5E, 1H	4E	3	relaxed	OMGDH	ROM
JKX18 Compiler [31]	2E, 1H, aPAKE	1E, aPAKE	3C	relaxed	OMGDH	ROM
CRISP [21]	6E, 3P, 3H, PAKE	3E, 3P, 1H, PAKE	3	full (siPAKE)	CDH	off GGM^{+}
AuCPace [29]	6E, 2H	5E, 1H	3	relaxed	sSDH, OMGDH	ROM
Ours Figure 6	E, PAKE	2E, PAKE	2	full	CDH	ROM + off GGM + on AGM
Ours Figure 7	A, PAKE	2A, PAKE	2	full	GACDH	ROM + off $\mathrm{GGAM}^{\top}$ + on $\mathrm{AGAM}^{\top}$

Table 1. A comparison of UC-secure saPAKE schemes. (1) E denotes exponentiations, H denotes hashing into the group, P denotes pairing evaluations, and A denotes group actions; (2) although both OPAQUE and AuCPace only achieves relaxed security, AuCPace realizes a stronger functionality than OPAQUE; (3) on/off denotes if we require the assumption in the online or offline phases respectively; (4) GGM^{+} denotes the extended generic group model [21] including hashing to the group, pairing evaluation, and isomorphism evaluation.

saPAKE under post-quantum assumptions. Very few password-based protocols under post-quantum assumptions have been proposed. The only such aPAKE protocol that we know of is the recent one by Freitas, Gu, and Jarecki [24], which can be instantiated under lattice assumptions. However, it is not a strong aPAKE, *i.e.*, it is subject to pre-computation attacks.[5]

The recent OPRF protocol due to Basso [8] is based on isogeny assumptions, and may provide some hope for constructing an saPAKE under post-quantum

[5] Another possible way to construct an aPAKE from post-quantum assumptions is to take a post-quantum UC oblivious transfer (OT), use the OT-to-PAKE compiler from [16] to obtain a UC PAKE, and then use the PAKE-to-aPAKE compiler from [26]. This also yields an aPAKE but not an saPAKE.

assumptions — by compiling the OPRF with some suitable aPAKE/AKE, as in the paradigm of the JKX compiler [31]. However, this OPRF protocol is only claimed to realize a UC OPRF functionality that does not take into account adaptive server compromise, which is crucial for saPAKE (and even aPAKE) security. As such, it is unclear whether this OPRF yields an saPAKE under post-quantum assumptions. Even if it does, the resulting saPAKE would only achieve the weak saPAKE functionality $\mathcal{F}_{\text{saPAKE}}^{-}$ with the two aforementioned shortcomings; furthermore, it would be less computationally efficient, take 2 additional rounds, and require significantly more bits of communication when compared with our protocol based on group actions.

2 Preliminaries

2.1 Notation

We use κ to denote the security parameter. For an integer n, $[n]$ denotes the set $\{1, \ldots, n\}$. For a probability distribution $\mathcal{D}$ over some set, we denote sampling an element d according to the distribution by $d \leftarrow \mathcal{D}$; we extend this notation naturally to probabilistic algorithms $a \leftarrow \mathcal{A}(x_1, x_2, \ldots)$ where the implicit distribution is defined by $\mathcal{A}$'s random coins. For a set S with no obvious accompanying distribution, we overload this notation to denote sampling from S according to the uniform distribution $s \leftarrow S$. For deterministic processes f, we denote assignment of $f(x_1, x_2, \ldots)$ to y by $y := f(x_1, x_2, \ldots)$. Finally, we use "PPT" as a shorthand for "probabilistic polynomial-time".

2.2 Computational Assumptions

Throughout this work, we use a cyclic group $\mathbb{G}$ with generator g and of prime order p, where $2^\kappa \leq p < 2^{\kappa+1}$. We assume $(\mathbb{G}, g, p)$ is public information and is omitted from all parties' inputs. We use the multiplicative notation for the group operation.

Definition 1 (The Discrete Logarithm (DL) Problem). *Let $a \leftarrow \mathbb{Z}_p^*$. Given g^a, the* Discrete Logarithm Problem *asks one to compute a.*

Definition 2 (The Computational Diffie-Hellman (CDH) Problem). *Let $(a, b) \leftarrow (\mathbb{Z}_p^*)^2$. Given a tuple (g^a, g^b), the* Computational Diffie-Hellman Problem *asks one to compute g^{ab}.*

The *advantage* of an adversary $\mathcal{A}$, denoted $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{DL}}$ (resp. $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{CDH}}$), is the probability that $\mathcal{A}$ solves the DL (resp. CDH) problem. The corresponding hardness assumptions state that there is no PPT adversary $\mathcal{A}$ whose advantage is non-negligible.[6] In Section 5.1, we use the natural extensions of these problems to group actions.

[6] Note that we sample exponents from $\mathbb{Z}_p^*$ rather than $\mathbb{Z}_p$, *i.e.*, 0 is excluded. This makes the protocol description and proof cleaner. It is obvious that our versions of DL and CDH assumptions are equivalent to the standard versions where the exponents are sampled from $\mathbb{Z}_p$.

2.3 UC saPAKE Security Model

We recall the UC functionalities for PAKE [17] (Figure 1) and saPAKE [31] (Figure 2 and Figure 3). Note that both functionalities only have implicit authentication, which is standard in the PAKE literature; explicit authentication can be achieved by adding a single key confirmation flow [28].

The (symmetric) PAKE functionality $\mathcal{F}_{\mathbf{PAKE}}$. In a PAKE protocol, two parties run a session on their (respective) passwords in order to generate a shared key k, modeled by the NewSession interface. If the MitM adversary does not interfere with the session, the two honest parties arrive at the same key k exactly when their passwords match. The only possible attack is the inevitable *online guessing attack*, in which the MitM adversary guesses a password pw^* and interacts with an honest party by running the counterparty's algorithm on pw^*. This is modeled by the TestPwd interface, through which the ideal adversary can control an honest party's key using the NewKey interface if the password guess is correct, *i.e.*, pw^* is equal to this party's password.

Functionality $\mathcal{F}_{\text{PAKE}}$

Storage:

- two maps, sessionStatus and session

Upon receiving $(\mathsf{NewSession}, \text{sid}, \mathsf{P}, \mathsf{P}', \text{role}, pw)$ from P:

1. Send $(\mathsf{NewSession}, \text{sid}, \mathsf{P}, \mathsf{P}', \text{role})$ to $\mathcal{A}^*$.
2. If there is no record $\mathsf{session}[(\text{sid}, \cdot, \cdot)]$ or exactly one record $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}', \mathsf{P})]$, set $\mathsf{session}[(\text{sid}, \mathsf{P}, \mathsf{P}')] := pw$ and $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}, \mathsf{P}')] := \mathsf{fresh}$.

Upon receiving $(\mathsf{TestPwd}, \text{sid}, \mathsf{P}, pw^*)$ from $\mathcal{A}^*$:

1. If $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}, \mathsf{P}')]$ is not fresh, ignore this query.
2. Otherwise:
   1. Retrieve $pw := \mathsf{session}[(\text{sid}, \mathsf{P}, \mathsf{P}')]$.
   2. If $pw = pw^*$, set $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}, \mathsf{P}')] := \mathsf{compromised}$ and return "correct guess" to $\mathcal{A}^*$.
   3. Otherwise, set $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}, \mathsf{P}')] := \mathsf{interrupted}$ and return "wrong guess" to $\mathcal{A}^*$.

Upon receiving $(\mathsf{NewKey}, \text{sid}, \mathsf{P}, k^*)$ from $\mathcal{A}^*$ where $|k^*| = \kappa$:

1. If $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}, \mathsf{P}')]$ is defined, but is not completed:
   1. If the record is compromised, set $k := k^*$.
   2. Else, if the record is fresh, (sid, k') was sent to P', $\mathsf{session}[(\text{sid}, \mathsf{P}, \mathsf{P}')] = \mathsf{session}[(\text{sid}, \mathsf{P}', \mathsf{P})]$, and at the time $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}', \mathsf{P})]$ was fresh, set $k := k'$.
   3. Otherwise, sample $k \leftarrow \{0,1\}^\kappa$.
2. Finally, set $\mathsf{sessionStatus}[(\text{sid}, \mathsf{P}, \mathsf{P}')] := \mathsf{completed}$ and send (sid, k) to P.

Fig. 1. Ideal functionality $\mathcal{F}_{\text{PAKE}}$

The saPAKE functionality $\mathcal{F}_{\mathbf{saPAKE}}$. As (s)aPAKE is meant to model the extension of PAKE to the client-server setting, we follow convention by calling one of the parties the client C, and calling corresponding counterparty the server S. C runs an saPAKE session on a (plain) password pw' through the ClientSession interface, while S runs the session on a password file $\mathsf{file}[(\mathsf{sid}, \mathsf{C}, \mathsf{S})]$ — representing the password underlying server's stored password digest — which is created through the StorePwdFile interface. Similar to PAKE, if the MitM adversary does not interfere with the session, the two honest parties arrive at the same key k exactly when $\mathsf{file}[(\mathsf{sid}, \mathsf{C}, \mathsf{S})] = pw'$. The StorePwdFile interface represents client registration, but is traditionally non-interactive with the server presumably receiving the password out-of-band *e.g.*, over an authenticated and secure channel and then securely erasing the password after storing the file.[7]

In addition to the online attack interface TestPwd, $\mathcal{F}_{\text{saPAKE}}$ also models adaptive server compromise through the StealPwdFile interface. After sending StealPwdFile, the ideal adversary gains access to two additional interfaces: OfflineTestPwd and Impersonate which allow the adversary to perform an *offline dictionary attack* — *i.e.*, make a password guess without invoking an online session — and authenticate with an honest client, respectively. To exclude pre-computation attacks, *the OfflineTestPwd interface ignores all messages until StealPwdFile is sent.*

2.4 Simulation Rate

As repeatedly pointed out in prior works [26,31,30,15], the (s)aPAKE functionality alone is not enough to model the offline security guarantees expected from (s)aPAKE protocols. Roughly speaking, offline security is concerned with the runtime of an offline dictionary attack after server compromise, or equivalently, how many passwords a real adversary can test per idealized model query (in our context, generic group operation). Although not explicit in the functionality, (s)aPAKE security requires that the server's storage be a *tight one-way function* of the password: namely, there should be a linear relationship between the number of password tests and the number of idealized model queries an adversary makes. For example, if the server storage is a traditional salted hash $(s, H(pw, s))$ where H is a random oracle and s is the salt, then each post-compromise query to H tests *at most one password.*

In the saPAKE ideal functionality, the adversary's post-compromise password tests are modeled by the OfflineTestPwd interface. From the description of the functionality (Figure 2), it is clear that each OfflineTestPwd command tests one password. However, there is an important caveat: as previously observed [26,31], the UC-modeling of (s)aPAKE requires a restriction on the simulator limiting the simulator's access to the OfflineTestPwd interface. Indeed, given unmediated access to the interface,

[7] In some sense this is counter to the one of the goals of saPAKE which is to prevent the server from ever seeing the client's password. Our informal description of the protocol has the client generate the password file themselves and send it over a secured channel, but formally the server will still generate the file.

Functionality $\mathcal{F}_{\text{saPAKE}}$

Storage:

– Four maps: fileStatus, file, sessionStatus, session

Password Registration

Upon receiving $(\mathsf{StorePwdFile}, \text{sid}, \mathsf{C}, pw)$ from S, if $\mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})]$ is undefined, set $\mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})] := pw$ and set $\mathsf{fileStatus}[(\text{sid}, \mathsf{C}, \mathsf{S})] := \mathsf{uncompromised}$.

Stealing Password Data

Upon receiving $(\mathsf{StealPwdFile}, \text{sid})$ from $\mathcal{A}^*$:

1. If $\mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})]$ is undefined, return "no password file" to $\mathcal{A}^*$.
2. Otherwise:
   1. If $\mathsf{fileStatus}[(\text{sid}, \mathsf{C}, \mathsf{S})] = \mathsf{uncompromised}$, set $\mathsf{fileStatus}[(\text{sid}, \mathsf{C}, \mathsf{S})] := \mathsf{stolen}$.
   2. Return "password file stolen" to $\mathcal{A}^*$.

Upon receiving $(\mathsf{OfflineTestPwd}, \text{sid}, pw^*)$ from $\mathcal{A}^*$:

1. If $\mathsf{fileStatus}[(\text{sid}, \mathsf{C}, \mathsf{S})] = \mathsf{stolen}$:
   1. Retrieve $pw := \mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})]$.
   2. If $pw = pw^*$, return "correct guess" to $\mathcal{A}^*$.
   3. Otherwise, return "wrong guess" to $\mathcal{A}^*$.

Password Authentication

Upon receiving $(\mathsf{ClientSession}, \text{sid}, \text{ssid}, \mathsf{S}, pw')$ from C:

1. Send $(\mathsf{ClientSession}, \text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$.
2. If $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})]$ is undefined, set $\mathsf{session}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})] := pw'$ and set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})] := \mathsf{fresh}$.

Upon receiving $(\mathsf{ServerSession}, \text{sid}, \text{ssid})$ from S:

1. If $\mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})]$ is undefined, ignore this query.
2. Otherwise, retrieve $pw := \mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})]$ and send $(\mathsf{ServerSession}, \text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$.
3. If $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{S}, \mathsf{C})]$ is undefined, set $\mathsf{session}[(\text{sid}, \text{ssid}, \mathsf{S}, \mathsf{C})] := pw$ and set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{S}, \mathsf{C})] := \mathsf{fresh}$.

Fig. 2. Ideal functionality $\mathcal{F}_{\text{saPAKE}}$ (part 1)

1. Protocols which realize $\mathcal{F}_{\text{aPAKE}}$ also realize $\mathcal{F}_{\text{saPAKE}}$ [31,15,21]: When the aPAKE simulator would send a pre-compromise OfflineTestPwd command, the saPAKE simulator instead catalogues the command and upon compromise of the server, it sends OfflineTestPwd for each catalogued command.
2. Assuming the password dictionary Dict has polynomial size, simply letting the server store the plain password is "secure" (and any PAKE is also an saPAKE) [30]: upon compromise of the server, the simulator iterates through Dict sending OfflineTestPwd for each possible password.

To rule out such degenerate protocols and to model the tight one-wayness of the password storage, we must restrict ourselves to simulators with limited access to the OfflineTestPwd interface. To these ends, we make explicit the ratio of the idealized model queries and the password guesses the adversary makes.

Active Session Attacks

Upon receiving $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{P}, pw^*)$ from $\mathcal{A}^*$:

1. If $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')]$ is undefined, ignore this query.
2. Otherwise, retrieve $pw' := \mathsf{session}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')]$.
3. If $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')] = \mathsf{fresh}$:
    1. If $pw' = pw^*$, return "correct guess" to $\mathcal{A}^*$ and set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')] := \mathsf{compromised}$.
    2. Otherwise, set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')] := \mathsf{interrupted}$ and return "wrong guess" to $\mathcal{A}^*$.

Upon receiving $(\mathsf{Impersonate}, \text{sid}, \text{ssid})$ from $\mathcal{A}^*$:

1. If $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})] = \mathsf{fresh}$:
    1. If $\mathsf{fileStatus}[(\mathsf{C}, \mathsf{S})] = \mathsf{stolen}$ and $\mathsf{file}[(\text{sid}, \mathsf{C}, \mathsf{S})] = \mathsf{session}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})]$, set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})] := \mathsf{compromised}$ and return "correct guess" to $\mathcal{A}^*$.
    2. Otherwise set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{C}, \mathsf{S})] := \mathsf{interrupted}$ and return "wrong guess" to $\mathcal{A}^*$.

Key Generation

Upon receiving $(\mathsf{NewKey}, \text{sid}, \text{ssid}, \mathsf{P}, k^*)$ from $\mathcal{A}^*$ where $|k^*| = \kappa$:

1. If $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')]$ is defined, but is not $\mathsf{completed}$:
    1. If the record is $\mathsf{compromised}$, set $k := k^*$.
    2. Else, if the record is fresh, $(\text{sid}, \text{ssid}, k')$ was sent to P', $\mathsf{session}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')] = \mathsf{session}[(\text{sid}, \text{ssid}, \mathsf{P}', \mathsf{P})]$, and at the time $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}', \mathsf{P})]$ was fresh, set $k := k'$.
    3. Otherwise, sample $k \leftarrow \{0,1\}^\kappa$.
2. Finally, set $\mathsf{sessionStatus}[(\text{sid}, \text{ssid}, \mathsf{P}, \mathsf{P}')] := \mathsf{completed}$ and send $(\text{sid}, \text{ssid}, k)$ to P.

Fig. 3. Ideal functionality $\mathcal{F}_{\text{saPAKE}}$ (part 2)

We define the *simulation rate*

$$r = \frac{\#\ \mathsf{OfflineTestPwd} \text{ commands sent by the simulator}}{\#\ \text{ideal model queries made by the adversary}}$$

of a protocol as the number of passwords that can be tested per a real adversary's ideal model query, and the "tight one-wayness" property can be expressed by requiring the simulation rate to be constant.[8] In the example above, the simulation rate of server storage $(s, H(pw, s))$ is 1. We then restrict the saPAKE simulator to only send at most r $\mathsf{OfflineTestPwd}$ commands *when the real adversary makes an idealized model query*, and disallow $\mathsf{OfflineTestPwd}$ from the simulator otherwise.

Hesse [30] proposes a way to formalize this intuitive change by restricting the simulator so that it may access $\mathsf{OfflineTestPwd}$ as long as its runtime remains *locally* T*-bounded* [30, Definition 3]. In other words, given any real-world adversary which runs in time $T(n)$, this change restricts the simulator to run in time $T(n)$ as well, where n is the number of input bits provided by the en-

[8] The term "simulation rate" is borrowed from [35].

vironment and functionality minus the adversary's output bits.[9] For the sake of simplicity, in this work we instead use the equivalent intuition of a "ticketing" mechanism as is common for limiting a simulator's actions. Our simulators will (conceptually) receive r "test tickets" whenever the real world adversary would make a specific oracle query and consume one of these tickets when the simulator sends OfflineTestPwd to $\mathcal{F}_{\text{saPAKE}}$. We then restrict our proofs to only consider simulators which do not send OfflineTestPwd when they have no tickets to consume.

Bradley, Jarecki, and Xu [15] prove the offline security of their saPAKE protocol in the GGM as we do; using our terminology, [15, Theorem 4] states that the simulation rate of their protocol is $O(1)$. While the offline security proof of our protocol is similar to theirs, we present a *concrete* analysis and show that our protocol has simulation rate 2. To the best of our knowledge, this is the first concrete offline security analysis of saPAKE in the GGM. We additionally show that our protocol achieves a simulation rate of 1 in the generic group action (with twists) model.

2.5 Idealized Models

The random oracle and generic group models. Our UC random oracle and generic group functionalities can be found in Figure 4 and Figure 5, respectively. For simplicity, we use a variant of the GGM where the adversary is allowed to compute $A^c B^d$ for group elements A, B and integers c, d of its choice *in a single query*; such a step corresponds to at least $\log \max\{c, d\}$ steps in the standard GGM where the adversary can only perform one multiplication or division per query.[10]

The algebraic group model. The algebraic group model, proposed by Fuchsbauer, Kiltz, and Loss [25], is an idealized model between the GGM and the standard model intended to analyze the security of group-based protocols.[11] Roughly speaking, the AGM requires the adversary to be *algebraic*, namely when it outputs a group element X, it must also output its *algebraic representation* $[X]_{\boldsymbol{x}} = (\lambda_1, \ldots, \lambda_n) \in \mathbb{Z}_p^n$ such that

$$X = X_1^{\lambda_1} \cdots X_n^{\lambda_n},$$

where $X_1, \ldots, X_n$ are group elements in the adversary's view so far.

The following lemma was proven in [25]:

[9] [15] uses a simpler formalization that is problematic; see [30, Appendix D] for a discussion.

[10] When we say in Section 2.4 that the simulation rate of our protocol is 2 in the GGM, we refer to this GGM variant. It is likely that the simulation rate is smaller than 1 in the standard GGM, although we do not perform a detailed analysis in this case.

[11] While the GGM is widely used to prove lower bounds for cryptographic assumptions, it is considered problematic to use it on the protocol level; see, e.g., [40] for a discussion.

Functionality $\mathcal{F}_{\mathrm{RO}}$

Parameters:

- range $\mathcal{H}$

Storage:

- map $\mathsf{H} : \{0,1\}^* \to \mathcal{H}$.

Upon receiving $(\mathsf{Eval}, \mathrm{sid}, x)$ from P:

1. If $\mathsf{H}[x]$ is undefined, sample $\mathsf{H}[x] \leftarrow \mathcal{H}$.
2. Send $(\mathsf{Eval}, \mathrm{sid}, \mathsf{H}[x])$ to P.

Fig. 4. Ideal functionality $\mathcal{F}_{\mathrm{RO}}$

Functionality $\mathcal{F}_{\mathrm{GG}}$

Parameters:

- handle set $\mathcal{G} \subseteq \{0,1\}^\kappa$
- prime p

Storage:

- map $\mathsf{DL} : \mathcal{G} \to \mathbb{Z}_p$.

Upon receiving $(\mathsf{Multi}, \mathrm{sid}, A, B, c, d)$ from P:

1. Retrieve $a := \mathsf{DL}[A]$ and $b := \mathsf{DL}[B]$; if either are undefined, sample them and set DL accordingly.
2. If there does not exist a $g' \in \mathcal{G}$ such that $\mathsf{DL}[g'] := ca + db$, sample $g' \leftarrow \mathcal{G}$, and set $\mathsf{DL}[g'] := ca + db$.
3. In all cases, send $(\mathsf{Multi}, \mathrm{sid}, g')$ to P.

Fig. 5. Ideal functionality $\mathcal{F}_{\mathrm{GG}}$

Lemma 3. *The DL and CDH assumptions are equivalent in the AGM. Concretely, for any CDH solver $\mathcal{A}$, there is a DL solver $\mathcal{B}$ whose runtime is approximately equal to that of $\mathcal{A}$ such that* $\mathbf{Adv}^{\mathsf{DL}}_{\mathcal{B}} = \mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{A}}$.

Given this lemma, we can claim that a protocol is secure in the AGM+DL while constructing a reduction to CDH, with no additional security loss. This is the approach we take in the security proof of our protocol.

Abdalla *et al.* [2] considered algebraic adversaries in the UC framework; in particular, they showed that the composition theorem still holds if we restrict the adversary (and the environment) to be algebraic, and as in the standard UC framework, we can still assume w.l.o.g. that the adversary is "dummy".

Offline security in the GGM and online security in the AGM. As in prior works [15,21], the offline security analysis of our saPAKE protocol is done in the GGM. This seems necessary as the offline security — the server's storage is

a tight one-way function of the password — is essentially a lower-bound result. However, as mentioned above, using the GGM on the protocol level (online security) might be viewed as problematic. Cremers *et al.* [21] state that their entire security result is in the GGM (see [21, Theorem 2]), while noting that the GGM is only used in the offline security analysis. Bradley et al. [15] state "we do not rely on GGM in the security analysis of the saPAKE protocol that uses [a tight one-way function] as the password file" (see [15, p.14]), and the authors take a more modular (yet less intuitive) approach: they abstract out the server storage as a separate primitive called salted tight one-way function (STOWF), prove that the server storage in their protocol is a UC STOWF in the GGM, and then show their protocol is a UC saPAKE without the GGM given a UC STOWF (plus some additional game-based properties). For readability, we follow the approach of Cremers *et al.* [21] and assume that the adversary must perform group operations via generic group queries *only while doing an offline attack*, while in online attacks the only constraint is that it must behave algebraically. However, we note that a more formal separation is straightforward by modeling the server storage as a STOWF similar to Bradley et al. [15].

3 Our saPAKE Protocol

Overview. Our starting point is the following naïve Diffie-Hellman-like protocol: The server S stores g^h where $h = H(pw)$, picks a random integer r and sends $R = g^r$ to the client C, and the two parties output g^{hr} as the session key (R^h for C and $(g^h)^r$ for S). The problem with this protocol is that g^{hr} has low entropy in the view of an eavesdropper that sees $R = g^r$. But since the two parties agree upon a low-entropy value, we can boost it into a high-entropy value by running a PAKE on top of it with g^{hr} as the input.

This yields an aPAKE but not a strong aPAKE: since the server storage is $g^{H(pw)}$, an attacker can pre-compute the table of $(x, g^{H(x)})$ for all candidate passwords x before compromising the server. To make it an saPAKE, we simply replace the fixed group base g with a variable base $S = g^s$ for a random integer s; that is, the server stores $(S, sw = S^{H(pw)})$ instead of $g^{H(pw)}$ (sw for "salted password"). This prevents the aforementioned attack as the adversary does not know S pre-compromise.

We note that using $(S, sw = S^{H(pw)})$ as server storage in saPAKE was originally suggested in [15, Section 3]. However, [15] dismisses this idea because the server storage is *malleable*: an adversary that compromises the server (but without performing an offline dictionary attack) can impersonate the server using an alternative server storage (S^{r^*}, sw^{r^*}) for a random integer r^*, and the UC simulator cannot tell this is an impersonation attack if DDH is hard in the group. One of our critical observations is that *such an impersonation attack can be detected by the UC simulator in the AGM*, since the adversary must output r^* as part of the algebraic representation — so the simulator can tell that $(S, sw, S^{r^*}, sw^{r^*})$ forms a DH tuple. See the proof overview in Section 4 for more details.

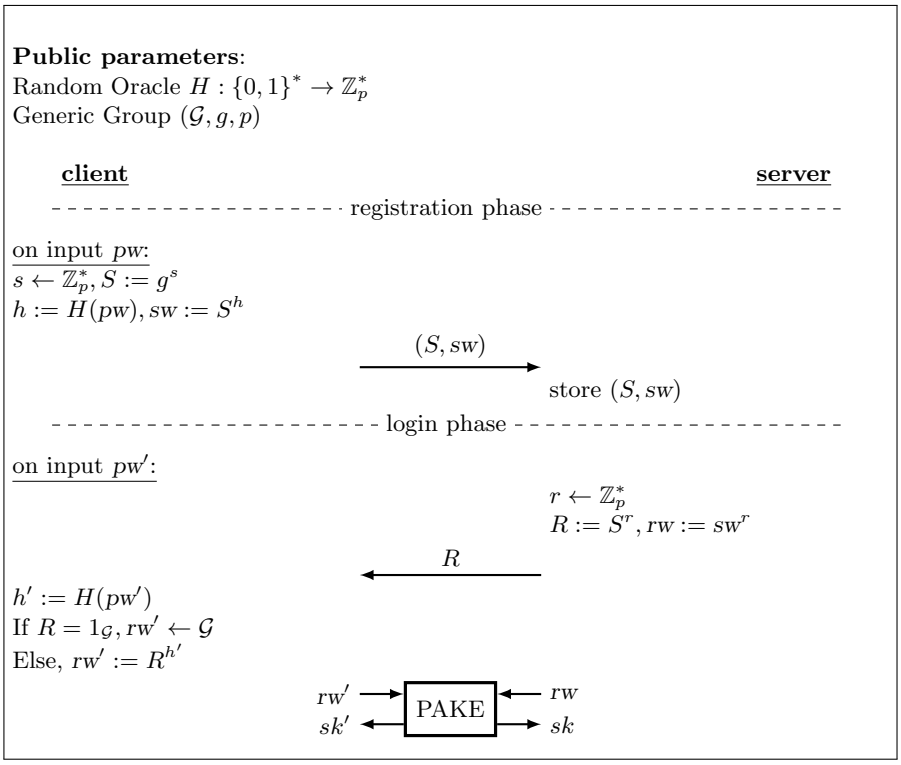

Fig. 6. Graphical representation of our protocol. See text for omitted details.

Below we formally present our saPAKE protocol, together with a graphic illustration.

Registration Phase

On input $(\mathsf{StorePwdFile}, \mathrm{sid}, \mathsf{C}, pw)$, S

1. Samples $s \leftarrow \mathbb{Z}_p^*$.
2. Sends $(\mathsf{Eval}, \mathrm{sid}, pw)$ to $\mathcal{F}_{\mathrm{RO}}$ receiving $(\mathsf{Eval}, \mathrm{sid}, h)$.
3. Sends $(\mathsf{Multi}, \mathrm{sid}, g, s)$ and $(\mathsf{Multi}, \mathrm{sid}, g, sh)$ to $\mathcal{F}_{\mathrm{GG}}$ receiving $(\mathsf{Multi}, \mathrm{sid}, S)$ and $(\mathsf{Multi}, \mathrm{sid}, sw)$.
4. Stores $\mathsf{file}[\mathrm{sid}] := (S, sw)$.

Server Compromise

Upon receiving $(\mathsf{StealPwdFile}, \mathrm{sid})$ from $\mathcal{A}$, S retrieves $\mathsf{file}[\mathrm{sid}]$ and sends it to $\mathcal{A}$. If there is no such record, S responds with "no password file".

Login Phase

1. On input $(\mathsf{ServerSession}, \text{sid}, \text{ssid})$, S
   1. Retrieves $(S, sw) := \mathsf{file}[\text{sid}]$.
   2. Samples $r \leftarrow \mathbb{Z}_p^*$.
   3. Sends $(\mathsf{Multi}, \text{sid}, S, r)$ and $(\mathsf{Multi}, \text{sid}, sw, r)$ to $\mathcal{F}_{\text{GG}}$ receiving $(\mathsf{Multi}, \text{sid}, R)$ and $(\mathsf{Multi}, \text{sid}, rw)$.
   4. Sends $(\text{sid}, \text{ssid}, R)$ to C and $(\mathsf{NewSession}, \text{sid}||\text{ssid}||R, \mathsf{S}, \mathsf{C}, \mathsf{S}, rw)$ to $\mathcal{F}_{\text{PAKE}}$.
2. On input $(\mathsf{ClientSession}, \text{sid}, \text{ssid}, \mathsf{S}, pw')$ and upon receiving $(\text{sid}, \text{ssid}, R)$ from S, C
   1. Sends $(\mathsf{Eval}, \text{sid}, pw')$ to $\mathcal{F}_{\text{RO}}$ receiving $(\mathsf{Eval}, \text{sid}, h')$.
   2. If $R = 1_{\mathcal{G}}$, samples $rw' \leftarrow \mathcal{G}$. Else, sends $(\mathsf{Multi}, \text{sid}, R, h')$ to $\mathcal{F}_{\text{GG}}$ receiving $(\mathsf{Multi}, \text{sid}, rw')$.
   3. Sends $(\mathsf{NewSession}, \text{sid}||\text{ssid}||R, \mathsf{C}, \mathsf{S}, \mathsf{C}, rw')$ to $\mathcal{F}_{\text{PAKE}}$.
3. Either party, upon receiving $(\text{sid}||\text{ssid}||R, k)$ from $\mathcal{F}_{\text{PAKE}}$, outputs $(\text{sid}, \text{ssid}, k)$.

See Figure 6 for a graphic illustration of our protocol. Note that in Figure 6 the registration phase is done interactively, where the client computes the password file and sends it to the server via a secure channel; whereas in the formal description the server computes the password file on its own using the password (and then erases the password). Figure 6 is more likely to match real-world applications, whereas the formal description matches the UC saPAKE functionality. An additional difference is the addition of R to the sid for the PAKE session. This is needed so $\mathcal{A}$ can't test password ratios in the honest-honest case.

Correctness. As pointed out in [37], correctness of (sa)PAKE — the two parties output the same key if their passwords match and there is no active attack — is not implied by UC-security and needs to be checked separately. It is trivial to see that our saPAKE protocol is correct assuming the underlying PAKE protocol is correct: if $pw' = pw$, then $h' = h$ and thus $rw = sw^r = (S^h)^r = S^{hr}$ and $rw' = R^{h'} = R^h = (S^r)^h = S^{hr}$, so the two parties' inputs to the PAKE protocol are equal. By the correctness of PAKE, their output keys are also equal.

4 Security Proof

Theorem 4. *The protocol in Section 3 UC-realizes $\mathcal{F}_{saPAKE}$ (Figure 2, Figure 3) with simulation rate $r = 2$ in the $(\mathcal{F}_{PAKE}, \mathcal{F}_{RO}, \mathcal{F}_{GG})$-hybrid model using the AGM for online analysis and the GGM for offline analysis, in the setting where both the client and the server can be statically corrupted and assuming the DL problem is hard in group $(\mathbb{G}, g, p)$.*

Proof overview. We provide a brief overview of the simulation strategy.

Offline security: At a high level, offline security requires that given $(S, sw = S^h)$ for h randomly chosen from a polynomial-size set $\mathbf{H}$, it takes time linear in

$|\mathbf{H}|$ to find h. This "discrete logarithm over a sparse set" problem has been studied (in the GGM) by Schnorr [38, Lemma 3], of which our argument is essentially a rewrite. The simulator creates a formal variable $\mathbb{P}$ representing $\log_S sw$, and since each generic group query by the adversary computes a linear function (in the exponent), the simulator records the corresponding linear function $u_i + v_i\mathbb{P}$, and tries to solve equations

$$u_i + v_i\mathbb{P} = u_j + v_j\mathbb{P},$$

where the solutions are candidate DL values (which are then tested via $\mathsf{OfflineTestPwd}$ queries). The reason we solve equations of this form is because the adversary can only learn information about the discrete logarithms of group elements by string comparison (equality checking) of their handles. The difficulty here is to show a lower bound of the number of solutions when the set $\mathbf{H}$ has polynomial size, which is the main technical contribution of [38] and which we repeat here.

Online security: The simulator must detect online saPAKE password tests; in particular, when the adversary sends a $\mathsf{TestPwd}$ message on a certain rw^* to $\mathcal{F}_{\text{PAKE}}$, the simulator must extract the corresponding password guess pw^* on the saPAKE level. Since the PAKE-level password rw is supposed to be $S^{H(pw)\cdot r} = R^{H(pw)}$ (where R is the server's message), pw^* can be easily extracted via looking at all H queries and checking which one satisfies $R^{H(pw^*)} = rw^*$.[12]

The simulator also needs to detect impersonation attacks, i.e., the adversary executes the server's algorithm after compromising it *without knowing the saPAKE password*. Since the server's storage is $(S, sw = S^{H(pw)})$, the adversary can choose an integer r^*, send $R^* := S^{r^*}$ to client, and then send a $\mathsf{TestPwd}$ message for C on sw^{r^*} — which will result in "correct guess" if C's password is pw. While this seems not simulatable without the AGM, in the AGM the simulator can extract r^* from R^*, so when the adversary uses rw^* in a $\mathsf{TestPwd}$ message, the simulator can check if $rw^* = sw^{r^*}$ (and send $\mathsf{Impersonate}$ to $\mathcal{F}_{\text{saPAKE}}$ if this is the case).

Note that in the two attacking scenarios above, only the second (the impersonation attack) needs the AGM.

The rest of the section is dedicated to the formal proof of Theorem 4. The simulator is described in Section 4.1, and we argue that this simulator generates an ideal-world view that is indistinguishable from the real-world view in Section 4.2.

4.1 Simulator

We construct the following simulator Sim for any PPT environment $\mathcal{Z}$. As standard in UC, we assume that the real adversary $\mathcal{A}$ is "dummy", *i.e.*, it merely

[12] Note that if $R = 1_{\mathbb{G}}$, the adversary can make a valid password guess without making any H query by setting $rw^* = 1_{\mathbb{G}}$. While this happens with negligible probability, simply excluding this case makes the proof cleaner.

passes messages to and from $\mathcal{Z}$. Without loss of generality, we also assume that all $\mathcal{F}_{\text{RO}}$ and $\mathcal{F}_{\text{GG}}$ queries are made via $\mathcal{A}$, *i.e.*, $\mathcal{Z}$ does not make these queries on its own. In the following, the session id is always included as part of a random oracle input and is omitted (*i.e.*, $H(\text{sid}, x)$ is simplified to $H(x)$).

Stealing the Password File and Offline Queries

1. Upon receiving $(\mathsf{StealPwdFile}, \text{sid})$ from $\mathcal{A}$ sent to S, send $(\mathsf{StealPwdFile}, \text{sid})$ to $\mathcal{F}_{\text{saPAKE}}$.
   A. If $\mathcal{F}_{\text{saPAKE}}$ returns "password file stolen"
      I. Mark S compromised.
      II. If $\mathsf{file}[\text{sid}]$ is undefined
         1. Sample a pair of group handles $(S, sw) \leftarrow \mathcal{G}^2$ and return (S, sw) to $\mathcal{A}$ from S.
         2. Create a formal variable $\mathbb{P}$ representing the discrete logarithm of sw relative to base S and sample $s \leftarrow \mathbb{Z}_p^*$.
         3. Store $\mathsf{DL}[S] := s$ and $\mathsf{DL}[sw] := s\mathbb{P}$.
   B. Otherwise, return "no password file" to $\mathcal{A}$.
2. Upon receiving $(\mathsf{Eval}, \text{sid}, x)$ from $\mathcal{A}$ sent to $\mathcal{F}_{\text{RO}}$:
   A. If $H(x)$ is undefined, sample $y \leftarrow \mathbb{Z}_p$ and record $H(x) := y$.
   B. If there exists $x' \neq x$ such that $H(x') = H(x)$, output $\mathsf{Collision}$ and abort.
   C. If S is marked compromised, send $(\mathsf{OfflineTestPwd}, \text{sid}, x)$ to $\mathcal{F}_{\text{saPAKE}}$.
      I. If $\mathcal{F}_{\text{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $H(x)$ in all future responses and store $\mathsf{serverPW}[\text{sid}] := x$.
   D. Return $(\mathsf{Eval}, \text{sid}, H(x))$ to $\mathcal{A}$.
3. Upon receiving $(\mathsf{Multi}, \text{sid}, A, B, c, d)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{GG}}$:
   A. If $\mathsf{DL}[g]$, for generator g associated with sid, is undefined, set $\mathsf{DL}[g] = 1$.
   B. If either $\mathsf{DL}[A]$ or $\mathsf{DL}[B]$ is undefined, sample the missing logarithm(s) from $\mathbb{Z}_p$.
   C. Interpret $a := \mathsf{DL}[A]$ and $b := \mathsf{DL}[B]$ as linear combinations over $\mathbb{Z}_p$ of $\{1, \mathbb{P}\}$, and record linear function $ca + db$ denoted γ:

$$\begin{aligned} a &= \alpha_1 + \alpha_2\mathbb{P} \\ b &= \beta_1 + \beta_2\mathbb{P} \\ \gamma &= (c\alpha_1 + d\beta_1) + (c\alpha_2 + d\beta_2)\mathbb{P} \end{aligned}$$

   D. If S is marked compromised:
      I. Suppose this is the t-th query $\mathcal{A}$ made to $\mathcal{F}_{\text{GG}}$ post-compromise. Then let $u_1 s + v_1 s\mathbb{P}, \ldots, u_{t+2} s + v_{t+2} s\mathbb{P}$ be the $t+2$ linear equations, recorded in chronological order, after compromise such that $(u_1, v_1) = (1, 0)$, $(u_2, v_2) = (0, 1)$, and $u_{t+2} s + v_{t+2} s\mathbb{P}$ is the linear function recorded during the current query.
      II. Compute all solutions to the $t+1$ equations

$$(v_i - v_{t+2}) X_{t+2,i} = u_{t+2} - u_i,$$

      where $i \in [t+1]$. Let the solutions be $h_{t+2,i}$.

III. For any $h_{t+2,i} = H(x_{t+2,i})$, send $(\mathsf{OfflineTestPwd}, \mathrm{sid}, x_{t+2,i})$ to $\mathcal{F}_{\mathrm{saPAKE}}$. If more than $2t$ $\mathsf{OfflineTestPwd}$ commands would be sent in total (*i.e.*, there is no "ticket" from $\mathcal{Z}$ to send an $\mathsf{OfflineTestPwd}$ command), output $\mathsf{OfflineFailure}$ and abort.

IV. Whenever $\mathcal{F}_{\mathrm{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $h_{t+2,i}$ in this and all future responses and store $\mathsf{serverPW}[\mathrm{sid}] := x_{t+2,i}$.

E. If γ is a fresh discrete logarithm, that is, for all previously generated handles C_i, $\mathsf{DL}[C_i] \neq \gamma$, then sample a new handle C from the set of handles $\mathcal{G}$ and set $\mathsf{DL}[C] := \gamma$. Otherwise there is an existing handle C_i such that $\mathsf{DL}[C_i] = \gamma$; in this case output $\mathsf{Collision}$ and abort.

F. Return $(\mathsf{Multi}, \mathrm{sid}, C)$ to $\mathcal{A}$.

Password Authentication

4. Upon receiving $(\mathsf{ServerSession}, \mathrm{sid}, \mathrm{ssid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathrm{saPAKE}}$:
   A. If $\mathsf{file}[\mathrm{sid}]$ is undefined
      I. Sample a pair of group handles $(S, sw) \leftarrow \mathcal{G}^2$ and store $\mathsf{file}[\mathrm{sid}] := (S, sw)$.
      II. Create a formal variable $\mathbb{P}$ representing the discrete logarithm of sw relative to base S and sample $s \leftarrow \mathbb{Z}_p^*$.
      III. Store $\mathsf{DL}[S] := s$ and $\mathsf{DL}[sw] := s\mathbb{P}$.
   B. If $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}]$ is undefined, then set $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, \perp)$.
   C. Sample $r \leftarrow \mathbb{Z}_p^*$, compute $R := g^r$, and send $(\mathrm{sid}, \mathrm{ssid}, R)$ to C from S.
   D. Send $(\mathsf{NewSession}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S}, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}$ from $\mathcal{F}_{\mathrm{PAKE}}$, set $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, R)$, and mark $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}]$ as "PAKE active".
5. Upon receiving $(\mathsf{ClientSession}, \mathrm{sid}, \mathrm{ssid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathrm{saPAKE}}$:
   A. If $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}]$ is undefined, then set $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, \perp)$.
   B. Wait to receive $(\mathrm{sid}, \mathrm{ssid}, [R^*]_{\boldsymbol{x}})$ from S sent to C.[13]
   C. Send $(\mathsf{NewSession}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, \mathsf{S}, \mathsf{C})$ to $\mathcal{A}$ from $\mathcal{F}_{\mathrm{PAKE}}$, set $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, R^*)$, and mark $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}]$ as "PAKE active".

Active Session Attacks

6. Upon receiving $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S}, [rw^*]_{\boldsymbol{x}})$ from $\mathcal{A}$ sent to $\mathcal{F}_{\mathrm{PAKE}}$, if there is a record $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}] = (\mathsf{C}, \mathsf{S}, R)$ marked "PAKE active":
   A. Check if there exists an x such that $rw^* = R^{H(x)}$. If so, x is uniquely defined (if there were two such x, the simulator would have output $\mathsf{Collision}$ and aborted). Otherwise set $x := \perp$.

[13] Formally $\mathcal{A}$ only sends $(\mathrm{sid}, \mathrm{ssid}, R^*)$ to C, and additionally outputs $[R^*]_{\boldsymbol{x}}$ as the algebraic representation of R^*. We use this compact form for brevity.

B. Send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{S}, x)$ to $\mathcal{F}_{\text{saPAKE}}$ and relay the response ("correct guess" or "wrong guess") to $\mathcal{A}$ from $\mathcal{F}_{\text{PAKE}}$.
C. If $\mathcal{F}_{\text{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $H(x)$ in all future responses and store $\mathsf{serverPW}[\text{sid}] := x$.

7. Upon receiving $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ from $\mathcal{A}$ sent to $\mathcal{F}_{\text{PAKE}}$, if there is a record $\mathsf{clientSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R^*)$ marked "PAKE active":
   A. If (1) S is marked $\mathsf{compromised}$ and (S, sw) was previously given to $\mathcal{A}$ upon server compromise, and (2) $[1]_{\boldsymbol{x}} \neq [R^*]_{\boldsymbol{x}} = r^*[S]_{\boldsymbol{x}}$ and $[rw^*]_{\boldsymbol{x}} = r^*[sw]_{\boldsymbol{x}}$, then send $(\mathsf{Impersonate}, \text{sid}, \text{ssid})$ to $\mathcal{F}_{\text{saPAKE}}$ and relay the response ("correct guess" or "wrong guess") to $\mathcal{A}$ from $\mathcal{F}_{\text{PAKE}}$.
   B. Otherwise (*i.e.*, no $\mathsf{Impersonate}$ command was sent):
      I. Check if there exists an x such that $rw^* = (R^*)^{H(x)}$. If so, x is uniquely defined. Otherwise set $x := \perp$.
      II. Send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{C}, x)$ to $\mathcal{F}_{\text{saPAKE}}$ and relay the response ("correct guess" or "wrong guess") to $\mathcal{A}$ from $\mathcal{F}_{\text{PAKE}}$.
      III. If $\mathcal{F}_{\text{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $H(x)$ in all future responses.

Key Generation

8. Upon receiving $(\mathsf{NewKey}, \text{sid}||\text{ssid}||R, \mathsf{C}, k^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$ such that there is a record $(\mathsf{C}, \mathsf{S}, R) := \mathsf{clientSession}[\text{sid}, \text{ssid}]$ marked "PAKE active":
   A. If there is a corresponding PAKE session for the server (*i.e.*, $\mathsf{serverSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R^*)$) and $R^* \neq R$ (*i.e.*, $\mathcal{A}$ modifies the message before PAKE), send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\text{saPAKE}}$.
   B. Regardless, send $(\mathsf{NewKey}, \text{sid}, \text{ssid}, \mathsf{C}, k^*)$ to $\mathcal{F}_{\text{saPAKE}}$ and mark $\mathsf{clientSession}[\text{sid}, \text{ssid}]$ as "PAKE completed".
9. Upon receiving $(\mathsf{NewKey}, \text{sid}||\text{ssid}||R, \mathsf{S}, k^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$ such that there is a record $(\mathsf{C}, \mathsf{S}, R) := \mathsf{serverSession}[\text{sid}, \text{ssid}]$ marked "PAKE active":
   A. If there is a corresponding PAKE session for the client (*i.e.*, $\mathsf{clientSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R^*)$) and $R^* \neq R$ (*i.e.*, $\mathcal{A}$ modifies the message before PAKE), send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\text{saPAKE}}$.
   B. Regardless, send $(\mathsf{NewKey}, \text{sid}, \text{ssid}, \mathsf{S}, k^*)$ to $\mathcal{F}_{\text{saPAKE}}$ and mark $\mathsf{serverSession}[\text{sid}, \text{ssid}]$ as "PAKE completed".

4.2 Proof of Indistinguishability

We now show that the simulator in Section 4.1 generates a view indistinguishable from the real world for any PPT environment $\mathcal{Z}$. We will proceed by a series of hybrids starting in the real world and ending in the ideal world. We use $\mathbf{Dist}_{\mathcal{Z}}^{i,i+1}$ to denote $\mathcal{Z}$'s distinguishing advantage between *Hybrids* i and $i+1$.

Hybrid 0: Real world
In this hybrid, the environment instructs the "dummy" adversary to play the role of a man-in-the-middle attacker between C and S. Recall that C's and S's passwords are denoted pw' and pw, respectively.

Hybrid 1: Ruling out random oracle and generic group collisions
In this hybrid, the challenger outputs $\mathsf{Collision}$ and aborts if there exist $x \neq x'$

such that $H(x) = H(x')$, or $A \neq A' \in \mathcal{G}$ such that their handles are equal. Assuming $\mathcal{A}$ makes q_{RO} Eval queries to $\mathcal{F}_{\text{RO}}$ and q_{GG} Multi queries to $\mathcal{F}_{\text{GG}}$, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{0,1} \leq \Pr[\mathsf{Collision}] \leq \frac{q_{\text{RO}}^2 + q_{\text{GG}}^2}{2p},$$

which is a negligible function of κ since $2^\kappa \leq p < 2^{\kappa+1}$.

Hybrid 2: Modifying R
In this hybrid, if $R^* \neq R$ (*i.e.*, $\mathcal{A}$ modifies the message from S to C before PAKE) and $\mathcal{A}$ does not send $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R, \mathsf{S}, \cdot)$ to $\mathcal{F}_{\text{PAKE}}$ (resp. $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R^*, \mathsf{C}, \cdot)$), then when $\mathcal{A}$ sends $(\mathsf{NewKey}, \text{sid}||\text{ssid}||R, \mathsf{S}, \cdot)$ to $\mathcal{F}_{\text{PAKE}}$ (resp. $(\mathsf{NewKey}, \text{sid}||\text{ssid}||R^*, \mathsf{C}, \cdot)$), S (resp. C) outputs a random key in $\{0,1\}^\kappa$ (independent of everything else).

In *Hybrid 1*, C's session id in $\mathcal{F}_{\text{PAKE}}$ is $\text{sid}||\text{ssid}||R^*$, and S's session id is $\text{sid}||\text{ssid}||R$. Therefore, if $R^* \neq R$ and there is no active attack on PAKE, $\mathcal{F}_{\text{PAKE}}$ will output independent random keys to C and S — exactly what *Hybrid 2* does. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{1,2} = 0.$$

Hybrid 3: Testing server's password
In this hybrid, when $\mathcal{A}$ sends $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R, \mathsf{S}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\text{PAKE}}$ and the server PAKE sub-session is active, $\mathcal{F}_{\text{PAKE}}$ returns "correct guess" and marks the sub-session compromised if $\mathcal{A}$ has queried $H(pw) = z$ and $rw^* = R^z$. Otherwise $\mathcal{F}_{\text{PAKE}}$ returns "wrong guess" and marks the sub-session interrupted.

In *Hybrid 2*, $\mathcal{F}_{\text{PAKE}}$ returns "correct guess" (and marks the sub-session compromised) if and only if $rw^* = R^{H(pw)}$. Therefore, *Hybrid 3* and *Hybrid 2* are identical unless $\mathcal{A}$ includes $rw^* = R^{H(pw)}$ in a TestPwd message without querying $z = H(pw)$. Call this event GuessServerrw. Note that $\mathcal{A}$ only learns $R = S^r$, and potentially S and $sw = S^{H(pw)}$ (if S is compromised); (S, sw, R, rw^*) forms a DH tuple. Therefore, an environment $\mathcal{Z}$ that causes GuessServerrw can be turned into a reduction $\mathcal{B}_1$ that solves the CDH problem in $(\mathbb{G}, g, p)$: Suppose there are at most ℓ sub-sessions. $\mathcal{B}_1(A, B)$ samples $i \leftarrow [\ell]$ as a guess that GuessServerrw happens in the i-th sub-session, runs the code of the *Hybrid 3* challenger with $S := g^s$, $sw := A^s$, and $R := B^s$ where R is the S-to-C message in the i-th sub-session (note that S and sw remain the same across all sub-sessions), and upon receiving rw^*, $\mathcal{B}_1$ outputs $(rw^*)^{1/s}$.[14] Clearly $\mathcal{B}_1$ wins if and only if GuessServerrw happens in the i-th sub-session. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{2,3} \leq \Pr[\mathsf{GuessServerrw}] \leq \ell \cdot \mathbf{Adv}_{\mathcal{B}_1}^{\mathsf{CDH}},$$

[14] Note that $\mathcal{A}$ never queries $H(pw)$ if GuessServerrw happens, so $\mathcal{B}_1$ can set S as g^s and $sw = S^{H(pw)}$ as A^s.

which is a negligible function of κ since the DL problem is hard in $(\mathbb{G}, g, p)$, and the CDH problem and the DL problem are equivalent in the AGM (see Lemma 3).

Hybrid 4: Impersonation attacks
In this hybrid, when $\mathcal{A}$ sends $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\mathrm{PAKE}}$ and the client PAKE sub-session is active, do the following if (1) S is compromised and (S, sw) was given to $\mathcal{A}$ upon server compromise, and (2) there exists $r \in \mathbb{Z}_p$ such that $[1]_{\boldsymbol{x}} \neq [R^*]_{\boldsymbol{x}} = r[S]_{\boldsymbol{x}}$ and $[rw^*]_{\boldsymbol{x}} = r[sw]_{\boldsymbol{x}}$:

- If $pw' = pw$, then $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" and marks the sub-session compromised;
- Otherwise $\mathcal{F}_{\mathrm{PAKE}}$ returns "wrong guess" and marks the sub-session interrupted.

Note that the change from *Hybrid 3* to *Hybrid 4* is made only if both (1) and (2) hold; in other words, if either (1) or (2) does not hold, there is no change from *Hybrid 3* to *Hybrid 4*. Now assume (1) and (2) hold. Then we have:

- $R^* = S^r$ and $rw^* = sw^r$, so (S, R^*, sw, rw^*) forms a DH tuple;
- $sw = S^{H(pw)}$ and $rw' = (R^*)^{H(pw')}$, so (S, R^*, sw, rw') forms a DH tuple if and only if $pw' = pw$ (note that collisions in H have been ruled out).

Thus, $rw^* = rw'$ if and only if (S, R^*, sw, rw') forms a DH tuple, which in turn happens if and only if $pw' = pw$. In *Hybrid 3*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" if and only if $rw^* = rw'$, whereas in *Hybrid 4*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" if and only if $pw' = pw$. This means that the conditions on which $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" in *Hybrid 3* and in *Hybrid 4* are equivalent. Thus, *Hybrid 3* and *Hybrid 4* are identical in $\mathcal{Z}$'s view, and

$$\mathbf{Dist}_{\mathcal{Z}}^{3,4} = 0.$$

Hybrid 5: Testing client's password
In this hybrid, when $\mathcal{A}$ sends $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\mathrm{PAKE}}$ and the client PAKE sub-session is active, if either (1) or (2) defined in *Hybrid 4* does not hold, do the following: $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" and marks the sub-session compromised if $\mathcal{A}$ has queried $H(pw') = z$ and $rw^* = (R^*)^z$. Otherwise $\mathcal{F}_{\mathrm{PAKE}}$ returns "wrong guess" and marks the sub-session interrupted.

In *Hybrid 4*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" (and marks the sub-session compromised) if and only if $rw^* = (R^*)^{H(pw')}$. Therefore, *Hybrid 5* and *Hybrid 4* are identical unless $\mathcal{A}$ includes $rw^* = (R^*)^{H(pw')}$ in a $\mathsf{TestPwd}$ message without querying $z = H(pw')$. Call this event $\mathsf{GuessClientrw}$. If $pw' \neq pw$, then $H(pw')$ is independent of the rest of the experiment, so $\mathsf{GuessClientrw}$ happens with probability $1/p$ over the choice of random oracle outputs.

If instead $pw' = pw$, an environment $\mathcal{Z}$ that causes $\mathsf{GuessClientrw}$ can be turned into a reduction $\mathcal{B}_2$ that solves the DL problem in $(\mathbb{G}, g, p)$. $\mathcal{B}_2(Q)$ samples $i \leftarrow [\ell]$ as before, and runs the code of the *Hybrid 5* challenger with

$S := g^s$ and $sw := Q^s$ in the i-th sub-session (so $q = \log Q$ is embedded as $H(pw)$). When $\mathcal{B}_2$ receives R^* and rw^* along with their algebraic representations $(a, b, c, t_1, \ldots, t_i)$ and $(\alpha, \beta, \gamma, \tau_1, \ldots, \tau_i)$ based on $g, S, sw, R_1, \ldots, R_i$ (where R_j is the S-to-C message in the j-th sub-session), $\mathcal{B}_2$ can obtain the expressions $R^* = g^d Q^e$ by condensing $g^a S^b \prod_j R_i^{t_j} = g^{a+sb+\sum_j sr_j t_j}$ and $sw^c = Q^{sc}$; similarly it can obtain $rw^* = g^\delta Q^\epsilon$. Combining these two equations with $rw^* = (R^*)^{H(pw')} = (R^*)^{H(pw)} = (R^*)^q$ we have

$$q^2 e + (d - \epsilon)q - \delta = 0,$$

from which $\mathcal{B}_2$ may solve for q when either $e \neq 0$ or $d - \epsilon \neq 0$. (Such equations are not generally solvable, but assuming GuessClientrw happens, there exists a solution. If there are two solutions, $\mathcal{B}_2$ may verify which one is correct by checking if $g^q = Q$ for each candidate solution.) If both are 0, we have $e = \delta = 0$ and $d = \epsilon$, so $R^* = g^d$ and $rw^* = Q^d$ which we covered in *Hybrid 4*.

We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{4,5} \leq \Pr[\mathsf{GuessClientrw}] \leq \max\left\{\ell \cdot \mathbf{Adv}_{\mathcal{B}_2}^{\mathsf{DL}}, \frac{1}{p}\right\},$$

which is a negligible function of κ since the DL problem is hard in $(\mathbb{G}, g, p)$.

Hybrid 6: Offline attacks
In this hybrid, S defines its password file file[sid] as $(S, sw) \leftarrow \mathcal{G}^2$, rather than $S \leftarrow \mathcal{G}$ and $sw := S^{H(pw)}$. Furthermore, when $\mathcal{A}$ computes $S^{H(pw)}$ via generic group queries, program the result as sw.

The difference between *Hybrid 6* and *Hybrid 5* is that in *Hybrid 5* sw is defined as $S^{H(pw)}$, while in *Hybrid 6* it is chosen at random from $\mathcal{G}$ and when $\mathcal{A}$ computes $S^{H(pw)}$, the result is programmed to be $S^{H(pw)}$. We can see that $\mathcal{Z}$'s views in these two hybrids are identical, so

$$\mathbf{Dist}_{\mathcal{Z}}^{5,6} = 0.$$

Combining all results above, we get

$$\mathbf{Dist}_{\mathcal{Z}}^{0,6} \leq \frac{q_{\mathrm{RO}}^2 + q_{\mathrm{GG}}^2 + 4}{2p} + \ell(\mathbf{Adv}_{\mathcal{B}_1}^{\mathsf{CDH}} + \mathbf{Adv}_{\mathcal{B}_2}^{\mathsf{CDH}}),$$

which is a negligible function of κ.

Comparison between *Hybrid 6* and the ideal world. We now compare $\mathcal{Z}$'s views in *Hybrid 6* and in the ideal world. *Hybrid 6* is a modified real world whose challenger, among other things, includes $\mathcal{F}_{\mathrm{PAKE}}$ with modified behavior (in particular, the rules on when sessions are marked compromised or interrupted are changed); we argue that in $\mathcal{Z}$'s view this challenger is identical to the combination of $\mathcal{F}_{\mathrm{saPAKE}}$ and the simulator Sim in the ideal world. First note that both games output Collision and abort if there is a collision in either H or the generic group. Below we assume that Collision does not happen.

We first analyze $\mathcal{F}_{\mathrm{PAKE}}$'s response to $\mathcal{A}$ ("correct guess" or "wrong guess") upon a TestPwd command. In both *Hybrid 6* and the ideal world, we have:

– When $\mathcal{A}$ sends $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R, \mathsf{S}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\text{PAKE}}$ and the server PAKE sub-session is active:
  - If $\mathcal{A}$ has queried $H(pw) = z$ and $rw^* = R^z$, then $\mathcal{F}_{\text{PAKE}}$ returns "correct guess";
  - Otherwise $\mathcal{F}_{\text{PAKE}}$ returns "wrong guess".

  This can be seen from *Hybrid 3* above and steps 6A and 6B of the simulator.[15]
– When $\mathcal{A}$ sends $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\text{PAKE}}$ and the client PAKE sub-session is active, if (1) S is compromised and (S, sw) was given to $\mathcal{A}$ upon server compromise, and (2) $\mathcal{A}$ has computed (R^*, rw^*) as (S^r, sw^r) for some $r \in \mathbb{Z}_p$:
  - If $pw' = pw$, then $\mathcal{F}_{\text{PAKE}}$ returns "correct guess";
  - Otherwise $\mathcal{F}_{\text{PAKE}}$ returns "wrong guess".

  This can be seen from *Hybrid 4* above and step 7A of the simulator.
– When $\mathcal{A}$ sends $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\text{PAKE}}$ and the client PAKE sub-session is active, if either (1) or (2) above does not hold:
  - If $\mathcal{A}$ has queried $H(pw') = z$ and $rw^* = (R^*)^z$, then $\mathcal{F}_{\text{PAKE}}$ returns "correct guess";
  - Otherwise $\mathcal{F}_{\text{PAKE}}$ returns "wrong guess".

  This can be seen from *Hybrid 5* above and steps 7B(I) and 7B(II) of the simulator.

Next, we analyze C and S's output keys when $\mathcal{A}$ sends NewKey to $\mathcal{F}_{\text{PAKE}}$. We first consider *Hybrid 6*. From *Hybrids 3–5*, we can see that whenever $\mathcal{A}$ sends a $\mathsf{TestPwd}$ command to $\mathcal{F}_{\text{PAKE}}$ resulting in "correct guess", $\mathcal{F}_{\text{PAKE}}$ marks the corresponding sub-session $\mathsf{compromised}$. Then when $\mathcal{A}$ sends NewKey, $\mathcal{F}_{\text{PAKE}}$ lets the corresponding party output the key that $\mathcal{A}$ specifies. On the other hand, if the $\mathsf{TestPwd}$ command results in "wrong guess", the sub-session is marked $\mathsf{interrupted}$, and when NewKey is sent, the corresponding party outputs an independent random key.

In the ideal world, when $\mathcal{A}$ sends a $\mathsf{TestPwd}$ command aimed at $\mathcal{F}_{\text{PAKE}}$, Sim always sends its own $\mathsf{TestPwd}$ command to $\mathcal{F}_{\text{saPAKE}}$ and relays $\mathcal{F}_{\text{saPAKE}}$'s answer to $\mathcal{A}$. This means that if $\mathcal{A}$ receives "correct guess", $\mathcal{F}_{\text{saPAKE}}$ marks the corresponding session $\mathsf{compromised}$; after that, when $\mathcal{A}$ sends NewKey, $\mathcal{F}_{\text{saPAKE}}$ lets the corresponding party output the key that $\mathcal{A}$ specifies. Similarly, if $\mathcal{A}$ receives "wrong guess", $\mathcal{F}_{\text{saPAKE}}$ marks the corresponding session $\mathsf{interrupted}$, and when $\mathcal{A}$ sends NewKey, $\mathcal{F}_{\text{saPAKE}}$ lets the corresponding party output an independent random key.

In other words, in both *Hybrid 6* and the ideal world, C or S outputs the key that $\mathcal{A}$ specifies if $\mathcal{A}$ has sent a $\mathsf{TestPwd}$ command aimed at $\mathcal{F}_{\text{PAKE}}$

[15] In the ideal world, Sim checks if there exists pw^* such that $\mathcal{A}$ has queried $H(pw^*) = z$ and $rw^* = R^z$; if not, Sim defines $pw^* := \perp$. Then Sim sends $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R, \mathsf{S}, pw^*)$ to $\mathcal{F}_{\text{saPAKE}}$. $\mathcal{F}_{\text{saPAKE}}$ sends "correct guess" to Sim if and only if $pw^* = pw'$, and Sim relays the answer to $\mathcal{A}$. Since we have ruled out collisions in H, $\mathcal{A}$ receives "correct guess" if and only if $\mathcal{A}$ has queried $H(pw) = z$ and $rw^* = R^z$. The cases below can be seen similarly.

resulting in "correct guess", and outputs an independent random key if the TestPwd command results in "wrong guess". The remaining case is that $\mathcal{A}$ does not send a TestPwd command. We argue that in this case, when $\mathcal{A}$ sends $(\mathsf{NewKey}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S}, k^*)$ and $(\mathsf{NewKey}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, k^*)$ aimed at $\mathcal{F}_{\mathrm{PAKE}}$, in both *Hybrid 6* and the ideal world,

- If $R^* = R$ (*i.e.*, $\mathcal{A}$ does not modify the message before PAKE), C and S output the same random key;
- Otherwise C and S output independent random keys.

In *Hybrid 6*, if $R^* = R$, $\mathcal{F}_{\mathrm{PAKE}}$ ensures that C and S output the same random key; otherwise they output independent random keys due to *Hybrid 2*. In the ideal world, if $R^* = R$, Sim does not send any TestPwd command to $\mathcal{F}_{\mathrm{saPAKE}}$, so $\mathcal{F}_{\mathrm{saPAKE}}$ ensures that C and S output the same random key; otherwise Sim sends $(\mathsf{TestPwd}, \mathrm{ssid}, \mathsf{C}, \perp)$ and $(\mathsf{TestPwd}, \mathrm{ssid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\mathrm{saPAKE}}$ (steps 8A and 9A), and $\mathcal{F}_{\mathrm{saPAKE}}$ marks both C sub-session and S sub-session interrupted, so C and S output independent random keys — which is exactly what happens in *Hybrid 6*.

We finally consider offline attacks. In *Hybrid 6*, S's password file is $(S, sw) \leftarrow \mathcal{G}^2$, and when $\mathcal{A}$ computes $S^{H(\mathrm{pw})}$ via random oracle and generic group queries, the result is programmed as sw. In the ideal world, this is exactly what Sim does in steps 1–3: whenever $\mathcal{A}$ makes a post-compromise generic group query, Sim solves for all x such that $\mathcal{A}$ tests if $sw = S^{H(x)}$, sends $(\mathsf{OfflineTestPwd}, \mathrm{sid}, x)$ to $\mathcal{F}_{\mathrm{saPAKE}}$, and if $\mathcal{F}_{\mathrm{saPAKE}}$ returns "correct guess" (*i.e.*, $x = pw$), then Sim programs $sw := S^{H(x)}$. The only difference is that in the ideal world, if at any point $\mathcal{A}$ makes t generic group queries but Sim needs to send more than $2t$ OfflineTestPwd commands (*i.e.*, Sim runs out of "tickets"), then Sim outputs OfflineFailure and aborts.

In sum, we have proven that $\mathcal{Z}$'s views in *Hybrid 6* and in the ideal world are identical, unless OfflineFailure happens. Since we have also proven that $\mathcal{Z}$'s views in the real world and in *Hybrid 6* are indistinguishable, this means that $\mathcal{Z}$'s views in the real world and in the ideal world are indistinguishable as long as OfflineFailure happens with negligible probability.

Lemma 5. $\Pr[\mathit{OfflineFailure}]$ *is a negligible function of* κ.

As mentioned in the proof overview, this is essentially rendering the proof of [38, Lemma 3] in the UC setting; for completeness, we include the proof of the lemma above in Appendix A.

5 An saPAKE from Group Actions

In this section we extend the analysis of the compiler in Section 3 to the generic group action model (GGAM).

5.1 Group Actions

Until this point, our compiler has relied on classical assumptions in cryptographic groups, specifically the hardness of the DL problem. However, Shor has shown [39] that discrete logarithms can be computed in polynomial time using a sufficiently large quantum computer. Our compiled protocols are not alone in this insecurity; indeed, previous UC-secure saPAKE protocols are built from Diffie-Hellman assumptions in groups [31,15,21] and thus are vulnerable to an adversary who can compute discrete logarithms.

As a competitor to the DL assumption, Couveignes [20] proposed replacing the group operations in traditional Diffie-Hellman with *cryptographic group actions* (therein referred to as hard homogenous spaces). For a group $\mathbb{G}$ and a set $\mathcal{X}$, a group action $\star$ is a map from $\mathbb{G} \times \mathcal{X}$ to $\mathcal{X}$ — analogous to exponentiation in classical groups — which respects group operations in $\mathbb{G}$; integrally, there is no group law on $\mathcal{X}$ which makes group actions resilient to Shor's algorithm. Following Couveignes' work, group actions have been used to construct various cryptographic schemes including symmetric PAKE [4].

We recall the definition of group actions:

Definition 6 (Group Action). *A group action of a group $(\mathbb{G}, e, \cdot)$ on a set $\mathcal{X}$ is a mapping $\star : \mathbb{G} \times \mathcal{X} \to \mathcal{X}$, usually written using infix notation as $g \star x$, which satisfies the following two properties:*

1. *Identity: $e \star x = x$ for all $x \in \mathcal{X}$.*
2. *Compatibility: $g \star g' \star x = (g \cdot g') \star x$ for all $g, g' \in \mathbb{G}$ and $x \in \mathcal{X}$.*

We additionally consider three properties of group actions:

1. *Freeness*: A group action $(\mathbb{G}, \mathcal{X}, \star)$ is said to be *free* when $g \star x = x \implies g = e$ for any $x \in \mathcal{X}$.
2. *Transitivity*: A group action $(\mathbb{G}, \mathcal{X}, \star)$ is said to be *transitive* when $\mathcal{X}$ is the only orbit under $\mathbb{G}$. In other words, $\forall x, y \in \mathcal{X},\ \exists g \in \mathbb{G} \mid x = g \star y$.
3. *Regularity*: A group action $(\mathbb{G}, \mathcal{X}, \star)$ is said to be *regular* when the action is both free and transitive.

For the rest of the paper, we will only consider actions which are regular and for which $\mathbb{G}$ is abelian. In the context of our protocol in Section 3, we can view the action of $\mathbb{Z}_p^*$ on $\mathcal{G} \setminus \{e\}$ in the natural way $a \star g = g^a$. Indeed, the only operation our protocol requires is exponentiation, so an honest party and simulator will only interact with $\mathcal{G}$ through this action. However, the additional structure $\mathcal{G}$ imposes disallows us from analyzing it as a generic group action.

As we wish to relate the security of our protocol to computational assumptions, we will further restrict our group actions to those with polynomial-time algorithms:

Definition 7 (Effective Group Action). *A group action $(\mathbb{G}, \mathcal{X}, \star)$ is said to be effective with respect to a computation security parameter κ if the following properties are satisfied:*

1. *$\mathbb{G}$ is finite and there exist polynomial-time algorithms (in κ) for the following:*
   (a) *Membership Testing: Decide if a given bitstring represents an element in $\mathbb{G}$.*
   (b) *Equality Testing: Decide if two given bitstrings represent the same element in $\mathbb{G}$.*
   (c) *Sampling: Sample an element g from $\mathbb{G}$ according to some distribution $\mathcal{D}_{\mathbb{G}}$. For the purpose of our protocol, we assume that $\mathcal{D}_{\mathbb{G}}$ is statistically close to the uniform distribution $\mathcal{U}_{\mathbb{G}}$ on $\mathbb{G}$.*
   (d) *Operation: Compute $g \cdot g'$ for any two elements $g, g' \in \mathbb{G}$.*
   (e) *Inversion: Compute g^{-1} for any element $g \in \mathbb{G}$.*
2. *$\mathcal{X}$ is finite (note that $|\mathbb{G}| = |\mathcal{X}|$ for regular actions) and there exist polynomial-time algorithms (in κ) for the following:*
   (a) *Membership Testing: Decide if a given bitstring represents an element in $\mathcal{X}$.*
   (b) *Unique Representation: Compute a unique bitstring $x^!$ canonically representing a given element $x \in \mathcal{X}$.*
3. *There exists a distinguished element $\widetilde{x} \in \mathcal{X}$ with known representation. We will refer to $\widetilde{x}$ as the* origin.
4. *There exists a polynomial-time algorithm (in κ) to evaluate the group action for any $g \in \mathbb{G}$ and $x \in \mathcal{X}$.*

An important category of post-quantum assumptions are those of isogeny-based cryptographic group actions, the formost of which is CSIDH [18]. Briefly, given a prime $p = 4 \cdot \ell_1 \cdots \ell_n - 1$ for ℓ_i small distinct odd primes, and elliptic curve $E_0 = y^2 = x^3 + x$ over $\mathbb{F}_p$ with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$, then

$$\begin{aligned} \star: &\quad \mathrm{cl}(\mathcal{O}) \times \mathcal{E}\ell\ell_p(\mathcal{O}) \to \mathcal{E}\ell\ell_p(\mathcal{O}) \\ \star: &\quad ([\mathfrak{a}], E) \mapsto E/\mathfrak{a} \end{aligned}$$

is a regular group action where $\mathrm{cl}(\mathcal{O})$ is the ideal-class group of $\mathcal{O}$ and $\mathcal{E}\ell\ell_p(\mathcal{O})$ is the set of all elliptic curves over $\mathbb{F}_p$ with $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$ [18].

To capture actions like CSIDH, we follow Duman *et al.* 's framework [22] and extend our definitions to include an additional operation called a *twist*

$$\begin{aligned} \tau: &\quad \mathcal{X} \to \mathcal{X} \\ \tau: &\quad (g \star x) \mapsto g^{-1} \star x \end{aligned}$$

which has a polynomial-time algorithm. As our results concern abelian groups, we will instead use additive notation and write $\tau : (g \star x) \mapsto (-g) \star x$. It is important to note that there is no corresponding operation for classical cryptographic groups assuming the inverse CDH Problem is hard (which is equivalent to the DL problem in the GGM [7]). Our protocol and simulator do not make use of the twist operation, and our proofs can readily be adapted to group actions without twists. However, to capture assumptions such as CSIDH, we provide the operation to the environment.

Finally, we assume that the structure of $\mathbb{G}$ is known including a minimal set of generators $\{g_1, \ldots, g_n\}$. Indeed, effective group actions over abelian groups are

quantum-equivalent to effective group actions over known-order groups through a generalization of Shor's algorithm [19] which computes an isomorphism $\mathbb{G} \simeq \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \cdots \times \mathbb{Z}_{m_n}$ along with a minimal set of generators. CSIDH-512, for example, is known to have a cyclic group of order

$$\begin{aligned} N =& 3 \cdot 37 \cdot 1407181 \cdot 51593604295295867744293584889 \\ & \cdot 31599414504681995853008278745587832204909 \end{aligned}$$

with generator $\langle 3, \pi - 1 \rangle$ *i.e.*, $\mathbb{G} \simeq \mathbb{Z}_N$ [12].

5.2 The Protocol

Our compiler in Figure 7 is the natural extension of our compiler in Figure 6 replacing the group operations with group actions. As the compiler runs independently of the PAKE protocol, we may instantiate the PAKE from classical assumptions [3,34], group actions (using the generic transform [16] from OT [33] to UC PAKE), or lattice assumptions [24] with instantiations using post-quantum assumptions resulting in the first UC-secure saPAKE protocols (realizing the full functionality) from post-quantum assumptions. Note that the recent group action PAKE protocol due to Abdalla *et al.* [4] is not known to be comptible with our compiler as their protocol has not been proven UC-secure.

5.3 Security Analysis

Theorem 8. *The protocol in Section 5.2 UC-realizes $\mathcal{F}_{saPAKE}$ (Figure 2, Figure 3) with simulation rate $r = 1$ in the $(\mathcal{F}_{PAKE}, \mathcal{F}_{RO}, \mathcal{F}_{GA^\top})$-hybrid model using the $AGAM^\top$ for online analysis and the $GGAM^\top$ for offline analysis, in the setting where both the client and the server can be statically corrupted and assuming the GA-DL problem is hard for known-order, abelian, effective group action $(\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}, \mathcal{X}, \widetilde{x}, \star)$, where ℓ_2, ℓ_3, the number of m_i divisible by 2 and 3, are $O(\log(\kappa))$. (See Appendix B for a formal description of the GGAM functionality $\mathcal{F}_{GA^\top}$.)*

The proof of this theorem is substantially similar to that of Section 4, so we only provide a sketch here and defer the full proof to Appendix C. The main change is that when the environment would produce server-to-client messages $R^* = g^a sw^b$ and PAKE inputs $rw^* = g^c sw^d$ in the online phase, it instead produces elements of the form $a \star \tilde{x}$, $b \star sw$, or $c \star -sw$. The non-trivial change we must make is in *Hybrid 5*, when $\mathcal{A}$ produces R^* and rw^*, $\mathcal{A}$ does not query $z = H(pw')$, and $pw = pw'$. We now consider the case where R^* is of the form $(a + b(s+q)) \star \tilde{x}$ where $b \in \{-1, 0, 1\}$ and similarly rw^* is of the form $(c + d(s+q)) \star \tilde{x}$ (for $d \in \{-1, 0, 1\}$) which combined with $rw^* = H(pw') \star R^* = H(pw) \star R^* = q \star R^*$ arrives at

$$q(d - 1 - b) = a + bs - c - ds.$$

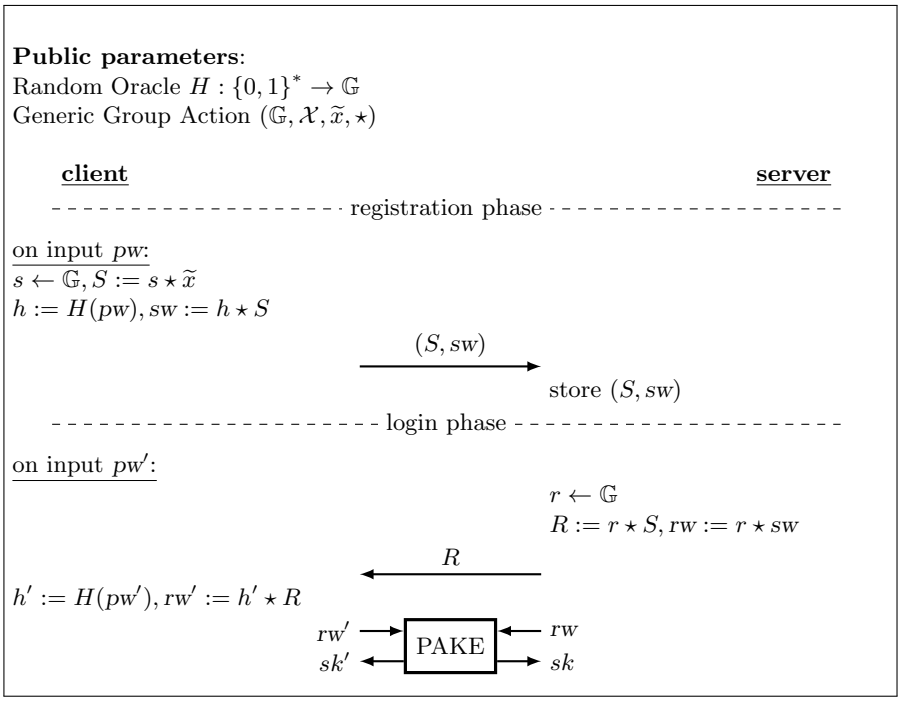

Fig. 7. Strong Asymmetric PAKE from Group Actions

Here, we have that $(d-b) \in \{-2,-1,0,1,2\}$ which means $(d-1-b) \in \{-3,-2,-1,0,1\}$. Just as before, this equivalence actually hides a system of modulo-equivalences. The i-th equivalence in the system has a single solution when $\gcd((d-1-b), m_i) = 1$ and at most $|d-1-b|$ solutions otherwise. As our reduction may verify possible solutions for q by computing $(q \star \tilde{x}) \stackrel{?}{=} Q$, we must show that the total number of solutions to this system is polynomial in κ. The total number of solutions is

$$\begin{aligned} |[\boldsymbol{q}]| &= \prod_{i \in [N]} \gcd((d-1-b), m_i) \\ &\leq 2^{\ell_2} \cdot 2^{\ell_3} \end{aligned}$$

where ℓ_2 is the number of m_i such that $\gcd(2, m_i) \neq 1$ and ℓ_3 is the number of m_i such that $\gcd(3, m_i) \neq 1$. If both 2^{ℓ_2} and 2^{ℓ_3} are polynomial in κ then there are a polynomial number of possible solutions and the reduction may extract the correct q.

CSIDH-512, for instance, has $\ell_2 = 0, \ell_3 = 1$ and we can achieve the same bound as Lemma 9 since $|\mathbf{H}| < q_{\mathrm{RO}}$ is excluded when we remove collisions. We note that our proof does not consider adversaries who can make queries in superposition. In addition to constructing more efficient PAKE building blocks,

constructing a proof for UC-secure saPAKE in the QROM and QGGAM is left as an interesting open problem.

References

1. M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In *CRYPTO 2020, Part I*, Aug. 2020.
2. M. Abdalla, M. Barbosa, J. Katz, J. Loss, and J. Xu. Algebraic adversaries in the universal composability framework. In *ASIACRYPT 2021, Part III*, Dec. 2021.
3. M. Abdalla, M. Barbosa, P. B. Rønne, P. Y. Ryan, and P. Šala. Security characterization of J-PAKE and its variants. Cryptology ePrint Archive, Report 2021/824, 2021. https://eprint.iacr.org/2021/824.
4. M. Abdalla, T. Eisenhofer, E. Kiltz, S. Kunzweiler, and D. Riepel. Password-authenticated key exchange from group actions. In *CRYPTO 2022, Part II*, Aug. 2022.
5. M. Abdalla, B. Haase, and J. Hesse. Security analysis of CPace. In *ASIACRYPT 2021, Part IV*, Dec. 2021.
6. N. Alamati, L. De Feo, H. Montgomery, and S. Patranabis. Cryptographic group actions and applications. In *ASIACRYPT 2020, Part II*, Dec. 2020.
7. F. Bao, R. H. Deng, and H. Zhu. Variations of Diffie-Hellman problem. In *ICICS 03*, Oct. 2003.
8. A. Basso. A post-quantum round-optimal oblivious prf from isogenies. Cryptology ePrint Archive, Paper 2023/225, 2023. https://eprint.iacr.org/2023/225.
9. B. Bauer, G. Fuchsbauer, and J. Loss. A classification of computational assumptions in the algebraic group model. In *CRYPTO 2020, Part II*, Aug. 2020.
10. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, May 1992.
11. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM CCS 93*, Nov. 1993.
12. W. Beullens, T. Kleinjung, and F. Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In *ASIACRYPT 2019, Part I*, Dec. 2019.
13. D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random oracles in a quantum world. In *ASIACRYPT 2011*, Dec. 2011.
14. D. Bourdrez, H. Krawczyk, K. Lewi, and C. Wood. The opaque asymmetric pake protocol, 2023. https://cfrg.github.io/draft-irtf-cfrg-opaque/draft-irtf-cfrg-opaque.html.
15. T. Bradley, S. Jarecki, and J. Xu. Strong asymmetric PAKE based on trapdoor CKEM. In *CRYPTO 2019, Part III*, Aug. 2019.
16. R. Canetti, D. Dachman-Soled, V. Vaikuntanathan, and H. Wee. Efficient password authenticated key exchange via oblivious transfer. In *PKC 2012*, May 2012.
17. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, May 2005.
18. W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: An efficient post-quantum commutative group action. In *ASIACRYPT 2018, Part III*, Dec. 2018.

19. K. K. Cheung and M. Mosca. Decomposing finite abelian groups. *Quantum Information & Computation*, 1(3):26–32, 2001.
20. J.-M. Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291.
21. C. Cremers, M. Naor, S. Paz, and E. Ronen. CHIP and CRISP: Compromise resilient identity-based symmetric PAKEs. In *CRYPTO 2022, Part II*, Aug. 2022.
22. J. Duman, D. Hartmann, E. Kiltz, S. Kunzweiler, J. Lehmann, and D. Riepel. Generic models for group actions. In *PKC 2023, Part I*, May 2023.
23. E. Eaton and D. Stebila. The "quantum annoying" property of password-authenticated key exchange protocols. In *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, 2021.
24. B. Freitas Dos Santos, Y. Gu, and S. Jarecki. Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In *EUROCRYPT 2023, Part V*, Apr. 2023.
25. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *CRYPTO 2018, Part II*, Aug. 2018.
26. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *CRYPTO 2006*, Aug. 2006.
27. P. Grassi, M. Garcia, J. Fenton, et al. NIST digital identity guidelines. 2020. https://csrc.nist.gov/publications/detail/sp/800-63/3/final.
28. A. Groce and J. Katz. A new framework for efficient password-based authenticated key exchange. In *ACM CCS 2010*, Oct. 2010.
29. B. Hasse and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. In *CHES 2019*, Aug. 2019.
30. J. Hesse. Separating symmetric and asymmetric password-authenticated key exchange. In *SCN 20*, Sept. 2020.
31. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *EUROCRYPT 2018, Part III*, Apr. / May 2018.
32. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, May 2001.
33. Y.-F. Lai, S. D. Galbraith, and C. de Saint Guilhem. Compact, efficient and UC-secure isogeny-based oblivious transfer. In *EUROCRYPT 2021, Part I*, Oct. 2021.
34. I. McQuoid, M. Rosulek, and L. Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In *ACM CCS 2020*, Nov. 2020.
35. I. McQuoid, M. Rosulek, and J. Xu. How to obfuscate MPC inputs. In *TCC 2022*, Nov. 2022.
36. A. Rostovtsev and A. Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145.
37. L. Roy and J. Xu. A universally composable PAKE with zero communication cost (And why it shouldn't be considered uc-secure). In *PKC 2023, Part I*, May 2023.
38. C. Schnorr. Small generic hardcore subsets for the discrete logarithm: Short secret DL-keys. *Information Processing Letters*, 79(2):93–98, 2001.
39. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.
40. J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart. Flaws in applying proof methodologies to signature schemes. In *CRYPTO 2002*, Aug. 2002.
41. S. Thomas. Re: [cfrg] proposed pake selection process. CFRG Mailing list, 2019. https://mailarchive.ietf.org/arch/msg/cfrg/dtf91cmavpzT47U3AVxrVGNB5UM.

42. D. Unruh. Universally composable quantum multi-party computation. In *EUROCRYPT 2010*, May / June 2010.

A Proof of Lemma 9

Lemma 9. $\Pr[\mathit{OfflineFailure}]$ *is a negligible function of* κ.

Proof. Since the server's storage is $(S, sw) = (g^s, g^{sh})$ (where $h = H(pw)$), every Multi query that $\mathcal{A}$ makes to $\mathcal{F}_{\mathrm{GG}}$ computes g^{us+vsh} for some $u, v \in \mathbb{Z}_p$; below we will often use (u, v) to denote the corresponding query or the group element g^{us+vsh}. Before compromise of the server, $\mathcal{Z}$'s view only consists of $g = (1/s, 0)$, so any query must be of form $(u, 0)$. After compromise of the server, $\mathcal{A}$ receives $f_1 := g^s = (1, 0)$ and $f_2 := g^{sh} = (0, 1)$ from Sim, which allows $\mathcal{A}$ to make queries (u, v) for $v \neq 0$; thus, we denote the i-th Multi query $\mathcal{A}$ makes to $\mathcal{F}_{\mathrm{GG}}$ after server compromise as $f_{i+2} := g^{u_{i+2}s+v_{i+2}sh} = (u_{i+2}, v_{i+2})$. (As we have just seen, $(u_1, v_1) = (1, 0)$ and $(u_2, v_2) = (0, 1)$.) Without loss of generality, we assume that $\mathcal{A}$ never repeats a query.

Suppose $\mathcal{A}$'s $\mathcal{F}_{\mathrm{RO}}$ queries result in a set of distinct integers $\mathbf{H} = \{h_1, \ldots, h_{q_{\mathrm{RO}}}\} \subseteq \mathbb{Z}_p$.[16] Consider $\mathcal{A}$'s first t $\mathcal{F}_{\mathrm{GG}}$ queries *after compromising the server*; Sim sends an OfflineTestPwd command to $\mathcal{F}_{\mathrm{saPAKE}}$ only for $1 \leq i < j \leq t + 2$ such that the solution to $f_i = f_j$

$$h_{i,j} := (u_j - u_i)(v_i - v_j)^{-1} \in \mathbf{H}$$

(note that we assume $(u_i, v_i) \neq (u_j, v_j)$, which implies $v_i \neq v_j$,[17] so $(v_i - v_j)^{-1}$ is well-defined — here we use the fact that p is prime). Therefore, for $\boldsymbol{u} = (u_1, \ldots, u_{t+2})$ and $\boldsymbol{v} = (v_1, \ldots, v_{t+2})$ defined by $\mathcal{A}$'s queries to $\mathcal{F}_{\mathrm{GG}}$, Sim makes $|\mathbf{H}_{\boldsymbol{u},\boldsymbol{v}}|$ OfflineTestPwd queries that correspond to $\mathcal{A}$'s $\mathcal{F}_{\mathrm{GG}}$ queries, where

$$\mathbf{H}_{\boldsymbol{u},\boldsymbol{v}} = \{h_{i,j}\}_{1 \leq i < j \leq t+2} \cap \mathbf{H} = \{h_{i,j} \in \mathbf{H} \mid 1 \leq i < j \leq t + 2\}.$$

Recall that OfflineFailure happens if for some t, Sim makes more than $2t$ queries that correspond to $\mathcal{A}$'s $\mathcal{F}_{\mathrm{GG}}$ queries. We now upper bound $\Pr[|\mathbf{H}_{\boldsymbol{u},\boldsymbol{v}}| > 2t]$, or equivalently, the probability that there exist more than $2t$ $h_{i,j} \in \mathbf{H}$. Any $\boldsymbol{u}, \boldsymbol{v}$ uniquely define all $h_{i,j}$; fixing $h_{i,j}$, consider the following linear equations of variables $U_3, \ldots, U_{t+2}, V_3, \ldots, V_{t+2}$

$$\begin{aligned}
&(u_1 - u_2) + h_{1,2}(v_1 - v_2) = 0 && \\
&(u_1 - U_j) + h_{1,j}(v_1 - V_j) = 0 && \text{for } 3 \leq j \leq t + 2 \\
&(u_2 - U_j) + h_{2,j}(v_2 - V_j) = 0 && \text{for } 3 \leq j \leq t + 2 \\
&(U_i - U_j) + h_{i,j}(V_i - V_j) = 0 && \text{for } 3 \leq i < j \leq t + 2
\end{aligned}$$

[16] If the password dictionary Dict is *a priori* fixed and has polynomial size, q_{RO} can be replaced by $\min\{q_{\mathrm{RO}}, |\mathsf{Dict}|\}$.

[17] If $v_i = v_j$, then $f_i = f_j$ implies $u_i = u_j$, but then $(u_i, v_i) = (u_j, v_j)$ — which has already been excluded.

(we stress that here $h_{i,j}$ is a fixed coefficient, (u_1, v_1) and (u_2, v_2) are fixed to be $(1, 0)$ and $(0, 1)$ respectively, and $U_3, \ldots, U_{t+2}, V_3, \ldots, V_{t+2}$ are variables over $\mathbb{Z}_p$). There are $\binom{t+2}{2}$ such linear equations but only $2t$ variables, so the dimension of the linear system is at most $2t$. Let $\mathbf{I}$ denote the largest subset of these $\binom{t+2}{2}$ linear equations such that all equations in $\mathbf{I}$ are linearly independent (if there are multiple, break ties arbitrarily), and $\mathbf{H}_I$ denote the $h_{i,j}$ values that appear in $\mathbf{I}$; then $|\mathbf{H}_I| = |\mathbf{I}| \leq 2t$. Now all $h_{i,j}$ are partitioned into two subsets: those in $\mathbf{H}_I$ (there are $|\mathbf{I}| \leq 2t$ of them) and those in $\mathbf{H} \setminus \mathbf{H}_I$ (there are $\binom{t+2}{2} - |\mathbf{I}| < \binom{t+2}{2}$ of them). The crucial observation is that *the solution space is fully determined by* $\mathbf{I}$*, so those* $h_{i,j} \in \mathbf{H} \setminus \mathbf{H}_I$ *are independent of* $\boldsymbol{u}, \boldsymbol{v}$. Furthermore, these $h_{i,j} \in \mathbf{H} \setminus \mathbf{H}_I$ are also independent of each other and all $h_{i,j} \in \mathbf{H}_I$. In other words, for each $h_{i,j} \in \mathbf{H} \setminus \mathbf{H}_I$, it is independent of everything else, so we have

$$\Pr[h_{i,j} \in \mathbf{H} \setminus \mathbf{H}_I] \leq \frac{q_{\mathrm{RO}}}{p},$$

so

$$\begin{aligned}
\Pr[\mathsf{OfflineFailure}] &= \Pr[\text{there exist more than } 2t \ h_{i,j} \in \mathbf{H}] \\
&\leq \Pr[\text{there exists at least one } h_{i,j} \in \mathbf{H} \setminus \mathbf{H}_I] \\
&< 1 - \left(1 - \frac{q_{\mathrm{RO}}}{p}\right)^{\binom{t+2}{2}} \\
&\approx 1 - \left(\frac{1}{e}\right)^{\frac{\binom{t+2}{2} q_{\mathrm{RO}}}{p}},
\end{aligned}$$

which is a negligible function of κ.

B Generic Group Action Functionality

See Figure 8 for the generic group action functionality $\mathcal{F}_{\mathrm{GA}^\top}$ we use in Section 5.

C Full Proof of Theorem 8

C.1 Simulator

Stealing the Password File and Offline Queries

1. Upon receiving $(\mathsf{StealPwdFile}, \mathrm{sid})$ from $\mathcal{A}$ sent to S, send $(\mathsf{StealPwdFile}, \mathrm{sid})$ to $\mathcal{F}_{\mathrm{saPAKE}}$.
   A. If $\mathcal{F}_{\mathrm{saPAKE}}$ returns "password file stolen"
      I. Mark S compromised.
      II. If $\mathsf{file}[\mathrm{sid}]$ is undefined
         1. Sample a pair of set handles $(S, sw) \leftarrow \mathcal{G}^2$ and return (S, sw) to $\mathcal{A}$ from S.

Functionality $\mathcal{F}_{\mathrm{GA}^\top}$

Parameters:

– group $(\mathbb{G}, +)$ and handle set $\mathcal{X} \subseteq \{0,1\}^\kappa$.

Storage:

– map $\mathsf{DL} : \mathbb{G} \to \mathcal{X}$.

Upon receiving $(\mathsf{op}, \mathrm{sid}, g', x)$ from P:

1. Retrieve $g := \mathsf{DL}[x]$ and if there does not exist an $x' \in \mathcal{X}$ such that $\mathsf{DL}[x'] := g' + g$, sample $x' \leftarrow \mathcal{X}$, and set $\mathsf{DL}[x'] := g' + g$.
2. In both cases, send $(\mathsf{op}, \mathrm{sid}, x')$ to P.

Upon receiving $(\mathsf{tw}, \mathrm{sid}, x)$ from P:

1. Retrieve $g := \mathsf{DL}[x]$ and if there does not exist an $x' \in \mathcal{X}$ such that $\mathsf{DL}[x'] := -g$, sample $x' \leftarrow \mathcal{X}$, and set $\mathsf{DL}[x'] := -g$.
2. In both cases, send $(\mathsf{op}, \mathrm{sid}, x')$ to P.

Fig. 8. Ideal functionality $\mathcal{F}_{\mathrm{GA}^\top}$

2. Create a formal variable $\mathbb{P}$ representing the "discrete logarithm" of sw relative to base S and sample $s \leftarrow \mathbb{G}$.
3. Store $\mathsf{DL}[S] := s$ and $\mathsf{DL}[sw] := s + \mathbb{P}$.

B. Otherwise, return "no password file" to $\mathcal{A}$.

2. Upon receiving $(\mathsf{Eval}, \mathrm{sid}, z)$ from $\mathcal{A}$ sent to $\mathcal{F}_{\mathrm{RO}}$:
   A. If $H(z)$ is undefined, sample $y \leftarrow \mathbb{G}$ and record $H(z) := y$.
   B. If there exists $z' \neq z$ such that $H(z') = H(z)$, output $\mathsf{Collision}$ and abort.
   C. If S is marked $\mathsf{compromised}$, send $(\mathsf{OfflineTestPwd}, \mathrm{sid}, z)$ to $\mathcal{F}_{\mathrm{saPAKE}}$.
      I. If $\mathcal{F}_{\mathrm{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $H(z)$ in all future responses and store $\mathsf{serverPW}[\mathrm{sid}] := z$.
   D. Return $(\mathsf{Eval}, \mathrm{sid}, H(z))$ to $\mathcal{A}$.
3. Upon receiving $(\mathsf{op}, \mathrm{sid}, \boldsymbol{g}, x)$ or $(\mathsf{tw}, \mathrm{sid}, x)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{GA}^\top}$:
   A. If $\mathsf{DL}[\tilde{x}]$, for origin $\tilde{x}$ associated with sid, is undefined, set $\mathsf{DL}[\tilde{x}] = 0$.
   B. If $\mathsf{DL}[x]$ is undefined, sample $\mathsf{DL}[x] \leftarrow \mathbb{G}$.
   C. Interpret $\mathsf{DL}[x]$ as an equation of the form $\boldsymbol{a} + b\mathbb{P} = (a_1 + b\mathbb{P}_1, \ldots, a_n + b\mathbb{P}_n)$ where $b \in \{-1, 0, 1\}$, and record $\boldsymbol{\gamma}$:

$$\boldsymbol{\gamma} := \begin{cases} (a_1 + g_1 + b\mathbb{P}_1, \ldots, a_n + g_n + b\mathbb{P}_n) & \text{if query was } \mathsf{op} \\ -\boldsymbol{a} - b\mathbb{P} & \text{if query was } \mathsf{tw} \end{cases}$$

   D. If S is marked $\mathsf{compromised}$:
      I. Suppose this is the t-th query $\mathcal{A}$ made to $\mathcal{F}_{\mathrm{GA}^\top}$ after the server's compromise. Then let $\boldsymbol{a}_1 + b_1\mathbb{P}, \ldots, \boldsymbol{a}_{t+2} + b_{t+2}\mathbb{P}$ be the $t+2$ discrete logarithms of the queries that $\mathcal{A}$ has made to $\mathcal{F}_{\mathrm{GA}^\top}$, recorded in chronological order, after compromise such that $(\boldsymbol{a}_1, b_1) = (s, 0)$, $(\boldsymbol{a}_2, b_2) = (s, 1)$, and $\boldsymbol{a}_{t+2} + b_{t+2}\mathbb{P}$ is the "discrete logarithm" $(\boldsymbol{\gamma})$ recorded during the current query.

II. Compute all solutions to the $t+1$ equations

$$\boldsymbol{a}_i + b_i \boldsymbol{h}_{t+2,i} = \boldsymbol{a}_{t+2} + b_{t+2} \boldsymbol{h}_{t+2,i}$$
$$(b_i - b_{t+2}) \boldsymbol{h}_{t+2,i} = \boldsymbol{a}_{t+2} - \boldsymbol{a}_i$$

where $i \in [t+1]$.

III. For any $\boldsymbol{h}_{t+2,i}$ such that there was a previous query $(\mathsf{Eval}, \mathrm{sid}, \boldsymbol{g}_{t+2,i})$ with $H(\boldsymbol{g}_{t+2,i}) = \boldsymbol{h}_{t+2,i}$, send $(\mathsf{OfflineTestPwd}, \mathrm{sid}, \boldsymbol{g}_{t+2,i})$ to $\mathcal{F}_{\mathrm{saPAKE}}$ unless an identical query has already been made. If this would cause the $t+1$th $\mathsf{OfflineTestPwd}$ command to be sent (*i.e.*, there is no "ticket" from $\mathcal{Z}$ to send an $\mathsf{OfflineTestPwd}$ command), output $\mathsf{OfflineFailure}$ and abort.

IV. Whenever $\mathcal{F}_{\mathrm{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $\boldsymbol{h}_{t+2,i}$ in this and all future responses and store $\mathsf{serverPW}[\mathrm{sid}] := \boldsymbol{g}_{t+2,i}$.

E. If $\boldsymbol{\gamma}$ is a fresh "discrete logarithm", that is, for all previously generated handles X_i, $\mathsf{DL}[X_i] \neq \boldsymbol{\gamma}$, then sample a new handle X from the set of handles $\mathcal{G}$ and set $\mathsf{DL}[X] := \boldsymbol{\gamma}$. If our new handle X happens to already appear as the handle of a previously computed set element (*e.g.*, there is already a "discrete logarithm" assigned to this handle which is *different from the currently computed "discrete logarithm"* $\boldsymbol{\gamma}$), output $\mathsf{Collision}$ and abort.

F. Return $(\mathsf{op}, \mathrm{sid}, X)$, respectively $(\mathsf{tw}, \mathrm{sid}, X)$, to $\mathcal{A}$.

Password Authentication

4. Upon receiving $(\mathsf{ServerSession}, \mathrm{sid}, \mathrm{ssid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathrm{saPAKE}}$:
   A. If $\mathsf{file}[\mathrm{sid}]$ is undefined
      I. Sample a pair of set handles $(S, sw) \leftarrow \mathcal{G}^2$ and store $\mathsf{file}[\mathrm{sid}] := (S, sw)$.
      II. Create a formal variable $\mathbb{P}$ representing the discrete logarithm of sw relative to base S and sample $s \leftarrow \mathbb{G}$.
      III. Store $\mathsf{DL}[S] := s$ and $\mathsf{DL}[sw] := s + \mathbb{P}$.
   B. If $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}]$ is undefined, then set $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, \perp)$.
   C. Sample group element $r \leftarrow \mathbb{G}$, compute set element $R := r \star \tilde{x}$, and send $(\mathrm{sid}, \mathrm{ssid}, R)$ to C from S.
   D. Send $(\mathsf{NewSession}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S})$ to $\mathcal{A}$ from $\mathcal{F}_{\mathrm{PAKE}}$, set $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, R)$, and mark $\mathsf{serverSession}[\mathrm{sid}, \mathrm{ssid}]$ as "PAKE active".
5. Upon receiving $(\mathsf{ClientSession}, \mathrm{sid}, \mathrm{ssid}, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathrm{saPAKE}}$:
   A. If $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}]$ is undefined, then set $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, \perp)$.
   B. Wait to receive $(\mathrm{sid}, \mathrm{ssid}, [R^*]_{\boldsymbol{x}})$ from S sent to C.
   C. Send $(\mathsf{NewSession}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C})$ to $\mathcal{A}$ from $\mathcal{F}_{\mathrm{PAKE}}$, set $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}] := (\mathsf{C}, \mathsf{S}, R^*)$, and mark $\mathsf{clientSession}[\mathrm{sid}, \mathrm{ssid}]$ as "PAKE active".

Active Session Attacks

6. Upon receiving $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R, \mathsf{S}, [rw^*]_{\boldsymbol{x}})$ from $\mathcal{A}$ sent to $\mathcal{F}_{\text{PAKE}}$, if there is a record $\mathsf{serverSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R)$ marked "PAKE active":
   A. Check if there exists an z such that $rw^* = H(z) \star R$. If so, z is uniquely defined. Otherwise set $z := \perp$.
   B. Send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{S}, z)$ to $\mathcal{F}_{\text{saPAKE}}$ and relay the response ("correct guess" or "wrong guess") to $\mathcal{A}$ from $\mathcal{F}_{\text{PAKE}}$.
   C. If $\mathcal{F}_{\text{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $H(z)$ in all future responses and store $\mathsf{serverPW}[\text{sid}] := z$.
7. Upon receiving $(\mathsf{TestPwd}, \text{sid}||\text{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ from $\mathcal{A}$ sent to $\mathcal{F}_{\text{PAKE}}$, if there is a record $\mathsf{clientSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R^*)$ marked "PAKE active":
   A. If (1) S is marked $\mathsf{compromised}$ and (S, sw) was previously given to $\mathcal{A}$ upon server compromise, and (2a) $[R^*]_{\boldsymbol{x}} = r^* + [S]_{\boldsymbol{x}}$ and $[rw^*]_{\boldsymbol{x}} = r^* + [sw]_{\boldsymbol{x}}$ or (2b) $[R^*]_{\boldsymbol{x}} = r^* + [-sw]_{\boldsymbol{x}}$ and $[rw^*]_{\boldsymbol{x}} = r^* + [-S]_{\boldsymbol{x}}$, then send $(\mathsf{Impersonate}, \text{sid}, \text{ssid})$ to $\mathcal{F}_{\text{saPAKE}}$ and relay the response ("correct guess" or "wrong guess") to $\mathcal{A}$ from $\mathcal{F}_{\text{PAKE}}$.
   B. Otherwise (*i.e.*, no $\mathsf{Impersonate}$ command was sent):
      I. Check if there exists an z such that $rw^* = H(z) \star R^*$. If so, z is uniquely defined. Otherwise set $z := \perp$.
      II. Send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{C}, z)$ to $\mathcal{F}_{\text{saPAKE}}$ and relay the response ("correct guess" or "wrong guess") to $\mathcal{A}$ from $\mathcal{F}_{\text{PAKE}}$.
      III. If $\mathcal{F}_{\text{saPAKE}}$ returns "correct guess", replace formal variable $\mathbb{P}$ with $H(z)$ in all future responses.

Key Generation

8. Upon receiving $(\mathsf{NewKey}, \text{sid}||\text{ssid}||R, \mathsf{C}, k^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$ such that there is a record $(\mathsf{C}, \mathsf{S}, R) := \mathsf{clientSession}[\text{sid}, \text{ssid}]$ marked "PAKE active":
   A. If there is a corresponding PAKE session for the server (*i.e.*, $\mathsf{serverSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R^*)$) and $R^* \neq R$ (*i.e.*, $\mathcal{A}$ modifies the message before PAKE), send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{C}, \perp)$ to $\mathcal{F}_{\text{saPAKE}}$.
   B. Regardless, send $(\mathsf{NewKey}, \text{sid}, \text{ssid}, \mathsf{C}, k^*)$ to $\mathcal{F}_{\text{saPAKE}}$ and mark $\mathsf{clientSession}[\text{sid}, \text{ssid}]$ as "PAKE completed".
9. Upon receiving $(\mathsf{NewKey}, \text{sid}||\text{ssid}||R, \mathsf{S}, k^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$ such that there is a record $(\mathsf{C}, \mathsf{S}, R) := \mathsf{serverSession}[\text{sid}, \text{ssid}]$ marked "PAKE active":
   A. If there is a corresponding PAKE session for the client (*i.e.*, $\mathsf{clientSession}[\text{sid}, \text{ssid}] = (\mathsf{C}, \mathsf{S}, R^*)$) and $R^* \neq R$ (*i.e.*, $\mathcal{A}$ modifies the message before PAKE), send $(\mathsf{TestPwd}, \text{sid}, \text{ssid}, \mathsf{S}, \perp)$ to $\mathcal{F}_{\text{saPAKE}}$.
   B. Regardless, send $(\mathsf{NewKey}, \text{sid}, \text{ssid}, \mathsf{S}, k^*)$ to $\mathcal{F}_{\text{saPAKE}}$ and mark $\mathsf{serverSession}[\text{sid}, \text{ssid}]$ as "PAKE completed".

C.2 Proof of Indistinguishability

We now show that the simulator in Appendix C.1 generates a view indistinguishable from the real world for any PPT environment $\mathcal{Z}$. We will proceed by a

series of hybrids starting in the real world and ending in the ideal world. We use $\mathbf{Dist}_{\mathcal{Z}}^{i,i+1}$ to denote $\mathcal{Z}$'s distinguishing advantage between Hybrids i and $i+1$.

Hybrid 0: Real world
In this hybrid, the environment instructs the "dummy" adversary to play the role of a man-in-the-middle attacker between C and S. Recall that C's and S's passwords are denoted pw' and pw, respectively.

Hybrid 1: Ruling out random oracle and generic group collisions
In this hybrid, Sim outputs $\mathsf{Collision}$ and aborts if there exist $z \neq z'$ such that $H(z) = H(z')$, or $A \neq A' \in \mathcal{X}$ such that their handles are equal.

Assuming $\mathcal{A}$ makes q_{RO} Eval queries to $\mathcal{F}_{\mathrm{RO}}$ and q_{GA} op and tw queries (jointly summed) to $\mathcal{F}_{\mathrm{GA}^\top}$, we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{0,1} \leq \Pr[\mathsf{Collision}] \leq \frac{q_{\mathrm{RO}}^2 + q_{\mathrm{GA}}^2}{2|\mathbb{G}|},$$

which is a negligible in κ since $2^\kappa \leq |\mathbb{G}| < 2^{\kappa+1}$.

Hybrid 2: Modifying R
In this hybrid, if $R^* \neq R$ (*i.e.*, $\mathcal{A}$ modifies the message from S to C before the PAKE session) and $\mathcal{A}$ does not send $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S}, \cdot)$ to $\mathcal{F}_{\mathrm{PAKE}}$ (resp. $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, \cdot)$), then when $\mathcal{A}$ sends $(\mathsf{NewKey}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S}, \cdot)$ to $\mathcal{F}_{\mathrm{PAKE}}$ (resp. $(\mathsf{NewKey}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, \cdot)$), Sim provides a uniform and independent key in $\{0,1\}^\kappa$ to S (resp. C) from $\mathcal{F}_{\mathrm{PAKE}}$.

In *Hybrid 1*, C's session id in $\mathcal{F}_{\mathrm{PAKE}}$ is $\mathrm{sid}||\mathrm{ssid}||R^*$, and S's session id is $\mathrm{sid}||\mathrm{ssid}||R$. Therefore, if $R^* \neq R$ and there is no active attack on the PAKE session, $\mathcal{F}_{\mathrm{PAKE}}$ will output independent random keys to C and S — exactly what *Hybrid 2* does. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{1,2} = 0.$$

Hybrid 3: Testing server's password
In this hybrid, when $\mathcal{A}$ sends $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R, \mathsf{S}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\mathrm{PAKE}}$ and the server PAKE sub-session is active, Sim has $\mathcal{F}_{\mathrm{PAKE}}$ return "correct guess" and mark the sub-session $\mathsf{compromised}$ if $\mathcal{A}$ has queried $H(pw) = h$ and $rw^* = h \star R$. Otherwise, Sim has $\mathcal{F}_{\mathrm{PAKE}}$ return "wrong guess" and mark the sub-session $\mathsf{interrupted}$.

In *Hybrid 2*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" (and marks the sub-session $\mathsf{compromised}$) if and only if $rw^* = H(pw) \star R$. Therefore, *Hybrid 3* and *Hybrid 2* are identical unless $\mathcal{A}$ includes $rw^* = H(pw) \star R$ in a $\mathsf{TestPwd}$ message without querying $z = H(pw)$. Call this event $\mathsf{GuessServerrw}$. Note that $\mathcal{A}$ only learns $R = r \star S$, and potentially S and $sw = H(pw) \star S$ (if S is compromised) and (S, sw, R, rw^*) forms a GADH tuple. Therefore, an environment $\mathcal{Z}$ that causes $\mathsf{GuessServerrw}$ can be turned into a reduction $\mathcal{B}_1$ that solves the GACDH problem in $(\mathbb{G}, \mathcal{X}, \star, \tilde{x})$: Suppose there are at most l sub-sessions. $\mathcal{B}_1(A, B)$ samples $i \leftarrow [l]$ as a guess that $\mathsf{GuessServerrw}$ happens in the i-th sub-session, runs the code of the *Hybrid 3* adversary with $S := s \star \tilde{x}$, $sw := s \star A$, and $R := s \star B$ where R is the S-to-C message in the i-th sub-session (note that S and sw remain the

same across all sub-sessions), and upon receiving rw^*, $\mathcal{B}_1$ outputs $-s \star rw^*$.[18] Clearly $\mathcal{B}_1$ wins if and only if GuessServerrw happens in the i-th sub-session.

We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{2,3} \leq \Pr[\mathsf{GuessServerrw}] \leq l \cdot \mathbf{Dist}_{\mathcal{B}_1}^{\mathrm{GACDH}},$$

which is negligible in κ since we assume the GADL problem is hard in $(\mathbb{G}, \mathcal{X}, \star, \tilde{x})$, and the GACDH problem and the GADL problem are equivalent in the AGAM.

Hybrid 4: Impersonation attacks
In this hybrid, when $\mathcal{A}$ sends $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\mathrm{PAKE}}$ and the client PAKE sub-session is active, then if (1) S is compromised and (S, sw) was given to $\mathcal{A}$ upon server compromise, and (2a) there exists $r^* \in \mathbb{G}$ such that $[R^*]_{\boldsymbol{x}} = r^* + [S]_{\boldsymbol{x}}$ and $[rw^*]_{\boldsymbol{x}} = r^* + [sw]_{\boldsymbol{x}}$ or (2b) there exists $r^* \in \mathbb{G}$ such that $[R^*]_{\boldsymbol{x}} = r^* + [-sw]_{\boldsymbol{x}}$ and $[rw^*]_{\boldsymbol{x}} = r^* + [-S]_{\boldsymbol{x}}$:

- If $pw' = pw$, Sim has $\mathcal{F}_{\mathrm{PAKE}}$ return "correct guess" and mark the sub-session compromised;
- Otherwise, Sim has $\mathcal{F}_{\mathrm{PAKE}}$ return "wrong guess" and mark the sub-session interrupted.

Note that the change from *Hybrid 3* to *Hybrid 4* is made only if both (1) and ((2a) or (2b)) hold; in other words, if either (1) or ((2a) and (2b)) does not hold, there is no change from *Hybrid 3* to *Hybrid 4*. Now assume (1) and (2a) hold. Then we have:

- $R^* = r^* \star S$ and $rw^* = r^* \star sw$, so (S, R^*, sw, rw^*) forms a GADH tuple;
- $sw = H(pw) \star S$ and $rw' = H(pw') \star R^*$, so (S, R^*, sw, rw') forms a GADH tuple if and only if $pw' = pw$.

or assume that (1) and (2b) hold. Then we have:

- $R^* = r^* \star -sw$ and $rw^* = r^* \star -S$, so $(-S, rw^*, -sw, R^*)$ forms a GADH tuple;
- $-sw = -H(pw) \star -S$ and $rw' = H(pw') \star R^* = (H(pw') + r^* - H(pw)) \star -S$, so $(-S, rw', -sw, R^*)$ forms a GADH tuple if and only if $pw' = pw$.

Thus, in both cases, $rw^* = rw'$ if and only if (S, R^*, sw, rw') (respectively $(-S, rw^*, -sw, R^*)$) forms a GADH tuple, which in turn happens if and only if $pw' = pw$. In *Hybrid 3*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" if and only if $rw^* = rw'$, whereas in *Hybrid 4*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" if and only if $pw' = pw$. This means that the conditions on which $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" in *Hybrid 3* and in *Hybrid 4* are equivalent. Thus, *Hybrid 3* and *Hybrid 4* are identical in $\mathcal{Z}$'s view, and

$$\mathbf{Dist}_{\mathcal{Z}}^{3,4} = 0.$$

[18] Note that $\mathcal{A}$ never queries $H(pw)$ if GuessServerrw happens, so $\mathcal{B}_1$ can set S as $s \star \tilde{x}$ and $sw = H(pw) \star S$ as $s \star A$.

Hybrid 5: Testing client's password
In this hybrid, when $\mathcal{A}$ sends $(\mathsf{TestPwd}, \mathrm{sid}||\mathrm{ssid}||R^*, \mathsf{C}, [rw^*]_{\boldsymbol{x}})$ to $\mathcal{F}_{\mathrm{PAKE}}$ and the client PAKE sub-session is active, if either (1) or ((2a) and (2b)) defined in *Hybrid 4* does not hold: Sim has $\mathcal{F}_{\mathrm{PAKE}}$ return "correct guess" and mark the sub-session compromised if $\mathcal{A}$ has queried $H(pw') = h$ and $rw^* = h \star R^*$. Otherwise Sim has $\mathcal{F}_{\mathrm{PAKE}}$ return "wrong guess" and mark the sub-session interrupted.

In *Hybrid 4*, $\mathcal{F}_{\mathrm{PAKE}}$ returns "correct guess" (and marks the sub-session compromised) if and only if $rw^* = H(pw') \star R^*$. Therefore, *Hybrid 5* and *Hybrid 4* are identical unless $\mathcal{A}$ includes $rw^* = H(pw') \star R^*$ in a TestPwd message without querying $z = H(pw')$. Call this event GuessClientrw. If $pw' \neq pw$, then $H(pw')$ is independent of the rest of the experiment, so GuessClientrw happens with probability $1/|\mathbb{G}|$ over the choice of random oracle outputs.

If instead $pw' = pw$, an environment $\mathcal{Z}$ that causes GuessClientrw can be turned into a reduction $\mathcal{B}_2$ that solves the GADL problem in $(\mathbb{G}, \mathcal{X}, \star, \tilde{x})$. $\mathcal{B}_2(Q)$ samples $i \leftarrow [l]$ as before, and runs the code of the *Hybrid 5* adversary with $S := s \star \tilde{x}$ and $sw := s \star Q$ in the i-th sub-session (so $q = \log(Q)$ is embedded as $H(pw)$). When $\mathcal{B}_2$ receives R^* and rw^* along with their algebraic representations $(\pm\alpha, \pm\beta, \pm\gamma, \pm\tau_1, \ldots, \pm\tau_i)$ and $(\pm\alpha', \pm\beta', \pm\gamma', \pm\tau_1', \ldots, \pm\tau_i')$ based on $\pm\tilde{x}, \pm S, \pm sw, \pm R_1, \ldots, \pm R_i$ (where R_j is the S-to-C message in the j-th sub-session), $\mathcal{B}_2$ can obtain the expressions $g \star \pm\tilde{x}$ or $g \star \pm Q$ from these representations as $sw = s \star Q$ and $R_j = r_j \star S = r_j \star s \star \tilde{x}$. We can now consider R^* taking the form $(a + b \cdot q) \star \tilde{x}$ where $b \in \{-1, 0, 1\}$ and similarly rw^* is of the form $(c + d \cdot q) \star \tilde{x}$ (for $d \in \{-1, 0, 1\}$). Combining this with $rw^* = H(pw') \star R^* = H(pw) \star R^* = q \star R^*$ we have

$$\begin{aligned} rw^* &= q \star R^* \\ c + d \cdot q &= q + a + b \cdot q \\ d \cdot q - q - b \cdot q &= a - c \\ q(d - 1 - b) &= a - c, \end{aligned}$$

from which $\mathcal{B}_2$ may solve for q when $d - 1 - b \neq 0$. However, $d - b \neq 1$ with overwhelming probability as

1. If $d = 1, b = 0$: $R^* = a \star \tilde{x}$, $rw^* = (c + q) \star \tilde{x}$, and $a = c$, which we covered in *Hybrid 4* (2a).
2. If $d = 0, b = -1$: $R^* = (a - q) \star \tilde{x}$, $rw^* = c \star \tilde{x}$, and $a = c$, which we covered in *Hybrid 4* (2b).

Note that since $\mathbb{G}$ may not be cyclic, the above equation is actually a system of equivalences

$$\begin{bmatrix} (d - 1 - b)\boldsymbol{q}_1 \equiv_{m_1} \boldsymbol{a}_1 + (b - d)\boldsymbol{s}_1 - \boldsymbol{c}_1 \\ (d - 1 - b)\boldsymbol{q}_2 \equiv_{m_2} \boldsymbol{a}_2 + (b - d)\boldsymbol{s}_2 - \boldsymbol{c}_2 \\ \vdots \\ (d - 1 - b)\boldsymbol{q}_n \equiv_{m_n} \boldsymbol{a}_n + (b - d)\boldsymbol{s}_n - \boldsymbol{c}_n \end{bmatrix}$$

where we have that $(d-b) \in \{-2,-1,0,2\}$, and $(d-1-b) \in \{-3,-2,-1,1\}$. The j-th equivalence in the system has has a single solution (solving for q_j) when $\gcd((d-1-b), m_j) = 1$ and at most $|d-1-b|$ solutions otherwise. As our reduction may verify possible solutions for q by computing $(q \star \tilde{x}) \stackrel{?}{=} Q$, we must show that the total number of solutions to this system is polynomial in κ. The total number of solutions is

$$\begin{aligned} |[\boldsymbol{q}]| &= \prod_{i \in [N]} \gcd((d-1-b), m_i) \\ &= 2^{\ell_2} \cdot 3^{\ell_3} \end{aligned}$$

where ℓ_2 is the number of m_i such that $\gcd(2, m_i) \neq 1$ and ℓ_3 is the number of m_i such that $\gcd(3, m_i) \neq 1$. If both 2^{ℓ_2} and 3^{ℓ_3} are polynomial in κ then there are a polynomial number of possible solutions and the reduction may extract the correct q.

We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{4,5} \leq \Pr[\mathsf{GuessClientrw}] \leq \max\left(l \cdot \mathbf{Dist}_{\mathcal{B}_2}^{\mathsf{GADL}}, \frac{1}{p}\right),$$

which is negligible in κ since we assume the DL problem is hard in $(\mathbb{G}, \mathcal{X}, \star, \tilde{x})$.

Hybrid 6: Offline attacks
In this hybrid, Sim defines S's password file $\mathsf{file}[\mathsf{sid}]$ as $(S, sw) \leftarrow \mathcal{G}^2$, rather than $S \leftarrow \mathcal{G}$ and $sw \leftarrow H(pw) \star S$. Furthermore, when $\mathcal{A}$ computes $H(pw) \star S$ via generic group queries, Sim programs the result as sw.

The difference between *Hybrid 6* and *Hybrid 5* is that in *Hybrid 5* sw is defined as $H(pw) \star S$, while in *Hybrid 6* it is chosen at random from $\mathcal{G}$ and when $\mathcal{A}$ computes $H(pw) \star S$, the result is programmed to be sw. We can see that $\mathcal{Z}$'s views in these two hybrids are identical, so

$$\mathbf{Dist}_{\mathcal{Z}}^{5,6} = 0.$$

Combining all results above, we get

$$\mathbf{Dist}_{\mathcal{Z}}^{0,6} \leq \frac{q_{\mathrm{RO}}^2 + q_{\mathrm{GA}}^2 + 4}{2p} + l(\mathbf{Dist}_{\mathcal{B}_1}^{\mathrm{DL}} + \mathbf{Dist}_{\mathcal{B}_2}^{\mathrm{DL}}),$$

which is negligible in κ.

By inspection, we can see that $\mathcal{Z}$'s views in *Hybrid 6* and the ideal world are identical unless $\mathsf{OfflineFailure}$ happens. The argument is almost identical to that in Section 4.2 (the only difference is that group operations need to be replaced by group action operations) and is thus omitted. We now upper-bound $\Pr[\mathsf{OfflineFailure}]$.

Lemma 10. $\Pr[\mathit{OfflineFailure}]$ *is a negligible function of* κ.

Proof. Since the server's storage is $(S, sw) = (s \star \tilde{x}, (s + h) \star \tilde{x})$ (where $h = H(pw)$), every op and tw query that $\mathcal{A}$ makes to $\mathcal{F}_{\mathrm{GA}^\top}$ computes $(u + b \cdot h) \star \tilde{x}$ for some $u \in \mathbb{G}, b \in \{-1, 0, 1\}$; below we will often use (u, b) to denote the corresponding query or the group element $(u + b \cdot h) \star \tilde{x}$. Before compromise of the server, $\mathcal{Z}$'s view only consists of $\tilde{x} = (0, 0)$, so any query must be of form $(u, 0)$. After compromise of the server, $\mathcal{A}$ receives $f_1 := s \star \tilde{x} = (s, 0)$ and $f_2 := (s + b \cdot h) \star \tilde{x} = (s, 1)$ from Sim, which allows $\mathcal{A}$ to make queries (u, b) for $b \neq 0$; thus, we denote the i-th op or tw query $\mathcal{A}$ makes to $\mathcal{F}_{\mathrm{GA}^\top}$ after server compromise as $f_{i+2} := (u_{i+2} + b_{i+2} \cdot h) \star \tilde{x} = (u_{i+2}, b_{i+2})$. (As we have just seen, $(u_1, v_1) = (s, 0)$ and $(u_2, v_2) = (s, 1)$.) Without loss of generality, we assume that $\mathcal{A}$ never repeats a query.

Suppose $\mathcal{A}$'s $\mathcal{F}_{\mathrm{RO}}$ queries result in a set of distinct integers $\mathbf{H} = \{h_1, \ldots, h_{q_{\mathrm{RO}}}\} \subseteq \mathbb{G}$. Consider $\mathcal{A}$'s first t $\mathcal{F}_{\mathrm{GA}^\top}$ queries *after compromising the server*; Sim sends an $\mathsf{OfflineTestPwd}$ command to $\mathcal{F}_{\mathrm{saPAKE}}$ only for $1 \leq i < j \leq t + 2$ such that the solution to $f_i = f_j$

$$\boldsymbol{h}_{i,j} = (\boldsymbol{u}_j - \boldsymbol{u}_i)/(b_i - b_j) \in \mathbf{H}.$$

But as we are working over a possibly non-cyclic group, we must treat these equations as a system of equations

$$\begin{bmatrix} (b_i - b_j)\boldsymbol{h}_{(i,j),1} \equiv_{m_1} \boldsymbol{u}_{j,1} - \boldsymbol{u}_{i,1} \\ (b_i - b_j)\boldsymbol{h}_{(i,j),2} \equiv_{m_2} \boldsymbol{u}_{j,2} - \boldsymbol{u}_{i,2} \\ \vdots \\ (b_i - b_j)\boldsymbol{h}_{(i,j),n} \equiv_{m_n} \boldsymbol{u}_{j,n} - \boldsymbol{u}_{i,n} \end{bmatrix}$$

which then are satisfied when each of the internal equations hold. We will then consider the equivalence classes $[\boldsymbol{h}]$ defined by the system in the natural way. As $b_i, b_j \in \{-1, 0, 1\}$, we have that $(b_i - b_j) \in \{-2, -1, 0, 1, 2\}$; however, we exclude the case where $(b_i - b_j) = 0$ just as we did in the proof of Lemma **??**. As equivalences of the form $\pm h \equiv_m u$ have a single solution (in variable h) and equivalences of the form $\pm 2h \equiv_m u$ have a single solution when $\gcd(2, m) = 1$ and at most two solutions otherwise, we can see that $|[\boldsymbol{h}]| \leq 2^{\ell_2}$ where ℓ_2 is the number of m_k such that $\gcd(2, m_k) \neq 1$.

Recall that $\mathsf{OfflineFailure}$ happens if for some t, Sim makes more than t queries that correspond to $\mathcal{A}$'s $\mathcal{F}_{\mathrm{GA}^\top}$ queries. We now upper bound $\Pr[|\mathbf{H}_{\boldsymbol{u},\boldsymbol{b}}| > t]$, or equivalently, the probability that there exist more than t $h_{i,j} \in \mathbf{H}$. Any $\boldsymbol{u}, \boldsymbol{b}$ uniquely define all $h_{i,j}$; fixing $h_{i,j}$, consider the following linear equations of variables $U_3, \ldots, U_{t+2}, B_3, \ldots, B_{t+2}$

$$\begin{aligned} (b_1 - b_2)\boldsymbol{h}_{1,2} + \boldsymbol{u}_1 - \boldsymbol{u}_2 &= 0 \\ (b_1 - B_j)\boldsymbol{h}_{1,j} + \boldsymbol{u}_1 - \boldsymbol{U}_j &= 0 \qquad \text{for } 3 \leq j \leq t + 2 \\ (b_2 - B_j)\boldsymbol{h}_{2,j} + \boldsymbol{u}_2 - \boldsymbol{U}_j &= 0 \qquad \text{for } 3 \leq j \leq t + 2 \\ (B_i - B_j)\boldsymbol{h}_{i,j} + \boldsymbol{U}_i - \boldsymbol{U}_j &= 0 \qquad \text{for } 3 \leq i < j \leq t + 2. \end{aligned}$$

There are $\binom{t+2}{2}$ such linear equations but only t variables, so the dimension of the linear system is at most t. Denoting the set of equivalence classes for random

oracle queries $\mathbf{H} := \{[H(x_1)], \ldots, [H(x_{q_{\mathrm{RO}}})]\}$, $\mathsf{OfflineFailure}$ occurs exactly when either $|\mathbf{H}| < q_{\mathrm{RO}}$ (a single equivalence class holds two password guesses) or $|\mathbf{H} \cap \{[\boldsymbol{h}_{i,j}]\}| > t$.

$$
\begin{aligned}
\Pr[\mathsf{OfflineFailure}] &\leq \Pr_{\mathbf{H}}[|\mathbf{H}| < q_{\mathrm{RO}}] + \Pr_{\mathbf{H}}[|\mathbf{H} \cap \{[\boldsymbol{h}_{i,j}]\}| > t] \\
&\leq \frac{2^{2\ell_2} q_{\mathrm{RO}}^2}{N} + 1 - \frac{\binom{N - 2^{\ell_2}|\{[\boldsymbol{h}_{i,j}]\}|}{2^{\ell_2} q_{\mathrm{RO}}}}{\binom{N}{2^{\ell_2} q_{\mathrm{RO}}}} \\
&= \frac{2^{2\ell_2} q_{\mathrm{RO}}^2}{N} + 1 - \frac{(N - 2^{\ell_2}|\{[\boldsymbol{h}_{i,j}]\}|)!}{N!} \cdot \frac{(N - 2^{\ell_2} q_{\mathrm{RO}})!}{(N - 2^{\ell_2} q_{\mathrm{RO}} - 2^{\ell_2}|\{[\boldsymbol{h}_{i,j}]\}|)!} \\
&= \frac{2^{2\ell_2} q_{\mathrm{RO}}^2}{N} + 1 - \prod_{i=1}^{2^{\ell_2}|\{[\boldsymbol{h}_{i,j}]\}|} \frac{N - 2^{\ell_2} q_{\mathrm{RO}} - (i-1)}{N - (i-1)} \\
&\leq \frac{2^{2\ell_2} q_{\mathrm{RO}}^2}{N} + 1 - \left(1 - \frac{2^{\ell_2} q_{\mathrm{RO}}}{N}\right)^{2^{\ell_2}\binom{t+2}{2}},
\end{aligned}
$$

which is negligible for 2^{ℓ_2} polynomial in κ.