Password-Authenticated Key Exchange from Group Actions

Michel Abdalla[1,2], Thorsten Eisenhofer[3], Eike Kiltz[3], Sabrina Kunzweiler[3], Doreen Riepel[3]

[1] DFINITY, Zürich, Switzerland
[2] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@ens.fr
[3] Ruhr-Universität Bochum, Bochum, Germany
{thorsten.eisenhofer,eike.kiltz,sabrina.kunzweiler,doreen.riepel}@rub.de

Abstract. We present two provably secure password-authenticated key exchange (PAKE) protocols based on a commutative group action. To date the most important instantiation of isogeny-based group actions is given by CSIDH. To model the properties more accurately, we extend the framework of cryptographic group actions (Alamati et al., ASIACRYPT 2020) by the ability of computing the quadratic twist of an elliptic curve. This property is always present in the CSIDH setting and turns out to be crucial in the security analysis of our PAKE protocols.
Despite the resemblance, the translation of Diffie-Hellman based PAKE protocols to group actions either does not work with known techniques or is insecure ("How not to create an isogeny-based PAKE", Azarderakhsh et al., ACNS 2020). We overcome the difficulties mentioned in previous work by using a "bit-by-bit" approach, where each password bit is considered separately.
Our first protocol $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ can be executed in a single round. Both parties need to send two set elements for each password bit in order to prevent offline dictionary attacks. The second protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ requires only one set element per password bit, but one party has to send a commitment on its message first. We also discuss different optimizations that can be used to reduce the computational cost. We provide comprehensive security proofs for our base protocols and deduce security for the optimized versions.

Keywords: Password-authenticated key exchange, group actions, CSIDH

1 Introduction

Password-authenticated key exchange (PAKE) enables two parties to securely establish a joint session key assuming that they only share a low-entropy secret known as the password. This reflects that passwords are often represented in short human-readable formats and are chosen from a small set of possible values, often referred to as dictionary.

Since the introduction of PAKE by Bellovin and Merritt [8], many PAKE protocols have been proposed, including SPEKE [20], SPAKE2 [4], J-PAKE [19] and CPace [18]. In particular over the last few years, the design and construction of PAKE protocols has attracted increasing attention, as the Crypto Forum Research Group (CFRG) which is part of the Internet Research Task Force (IETF) started a selection process to decide which PAKE protocols should be used in IETF protocols. Recently, CPace was selected as the recommended protocol for symmetric PAKE, where both parties share the same password.

Different models have been used to formally prove security of PAKE protocols, like indistinguishability-based models or the universal composability framework. In general, a PAKE protocol should resist offline and online dictionary attacks. On the one hand an adversary should not be able to perform an exhaustive search of the password offline. On the other hand, an active adversary should only be able to try a small number of passwords in one protocol execution. Furthermore, forward security ensures that session keys are still secure, even if the password is leaked at a later point in time. The same should hold if session keys are disclosed, which should not affect security of other session keys.

CSIDH and Group Actions. The PAKE protocols mentioned above are mostly based on a Diffie-Hellman key exchange in a prime order group. A promising post-quantum replacement is isogeny-based key exchange. The different isogeny-based protocols can be divided into two groups. On the one hand there are constructions based on commutative group actions on a set of elliptic curves. The first proposals by Couveignes [12], and Stolbunov and Rostovtsev [27] suggested to use the action of the class group $cl(\mathcal{O})$ on the set of $\mathbb{F}_q$-isomorphism classes of *ordinary* elliptic curves with endomorphism ring $\mathcal{O}$. In 2018, Castryck et al. showed that this idea can also be adapted to the class group action on the set of $\mathbb{F}_p$-isomorphism classes of *supersingular* elliptic curves [11]. The resulting scheme is called CSIDH and constitutes the first practical key exchange scheme based on class group actions.

In [12], Couveignes introduces *hard homogeneous spaces* - an abstract framework for group actions that models isogeny-based assumptions. This framework has been further refined by Alamati et al. in [5]. Using the abstract setting of *cryptographic group actions* the authors develop several new cryptographic primitives that can be instantiated with CSIDH. On the other hand there is the Supersingular Isogeny Diffie-Hellman (SIDH) protocol suggested by Jao and De Feo in 2011 [21]. Here, the set of $\mathbb{F}_{p^2}$-isomorphism classes of *supersingular* elliptic curves is considered. The endomorphism ring of a supersingular elliptic curve over $\mathbb{F}_{p^2}$ is non-commutative, hence protocols based on SIDH do not fall into the group action framework.

We now recall the framework of (restricted) effective group actions introduced in [5]. Throughout, $\mathcal{G}$ denotes a finite commutative group and $\mathcal{X}$ a set. We assume that $\mathcal{G}$ acts regularly on $\mathcal{X}$ via the operator $\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$. Regularity guarantees that for any $x, y \in \mathcal{X}$ there exists precisely one group element $g \in \mathcal{G}$ satisfying

$y = g \star x$. Broadly speaking, we are interested in group actions, where evaluation is easy, but the "discrete logarithm problem" is hard. Expressed differently:

- Given $x \in \mathcal{X}$ and $g \in \mathcal{G}$, one can efficiently compute the set element $y = g \star x$.
- Given $x, y \in \mathcal{X}$, it is hard to find the element $g \in \mathcal{G}$ satisfying $y = g \star x$.

These properties facilitate the definition of a Diffie-Hellman key exchange. Let x be some fixed set element. Alice chooses a secret $g_A \in \mathcal{G}$ and publishes $y_A = g_A \star x$. Similarly Bob chooses $g_B \in \mathcal{G}$ and publishes $y_B = g_B \star x$. They can both compute the shared secret $y_{AB} = g_A \star y_B = g_B \star y_A$. The group action computational Diffie-Hellman problem (GA-CDH) then states that given y_A and y_B, it is hard to compute y_{AB}. We refer to Section 3 for more precise definitions.

Contributions and Technical Details. Our main contributions are the two PAKE protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ based on commutative group actions. These are the first two provably secure PAKE protocols that are directly constructed from isogenies.

Group Actions with Twists. To date the most important instantiation of isogeny-based group actions is given by CSIDH. To model this situation more accurately, we suggest an enhancement of the framework which includes the ability of computing the quadratic twist of an elliptic curve efficiently. This property is inherent to CSIDH (cf. [11]) and it turns out to be crucial in the security analysis of our PAKE protocols. On the one hand, twisting allows us to construct an offline dictionary attack against our first natural PAKE attempt GA-PAKE$_\ell$. Notably, this first protocol is secure for group actions where twisting is not possible efficiently. On the other hand, twists play an important role in various security reductions applied to prove the security of our new protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$. Interestingly, this is also the case when twists are not part of any of the two problems involved in the reduction.

First attempt: GA-PAKE$_\ell$. Our two secure PAKE protocols are modifications of GA-PAKE$_\ell$. In order to illustrate the main idea behind the protocols, we describe GA-PAKE$_\ell$ in more detail here. The protocol (Figure 1) can be seen as an adaption of the simple password exponential key exchange protocol SPEKE [20] to the group action setting. In SPEKE the password is used to hash to a generator of the group. Then the user and the server establish a session key following the Diffie-Hellman key exchange. Directly translating this protocol to the group action setting requires to hash the password to a random set element $x \in \mathcal{X}$. For isogeny-based group actions, this is still an open problem, hence (at the moment) a straight-forward translation of SPEKE is not possible (see also [6, §4.1]). In GA-PAKE$_\ell$ we map the password to an ℓ-tuple of elements in $\mathcal{X}$ instead of hashing to one element. More precisely, two elements $\mathsf{crs} = (x_0, x_1) \in \mathcal{X}^2$ are fixed by a trusted party and a password $\mathsf{pw} = (b_1, \ldots, b_\ell) \in \{0,1\}^\ell$ is mapped to the tuple $(x_{b_1}, \cdots, x_{b_\ell}) \in \mathcal{X}^\ell$. Then a Diffie-Hellman key exchange is performed with basis x_{b_i} for each $i \in [\ell]$. This means the user generates ℓ random group elements $u_1, \ldots, u_\ell$ and computes the elements $x_1^{\mathsf{U}} = u_1 \star x_{b_1}, \ldots, x_\ell^{\mathsf{U}} = u_\ell \star x_{b_\ell}$ which it sends to the server. Similarly, the server generates ℓ random group elements $s_1, \ldots, s_\ell$ and computes $x_1^{\mathsf{S}} = s_1 \star x_{b_1}, \ldots, x_\ell^{\mathsf{S}} = s_\ell \star x_{b_\ell}$ which it sends

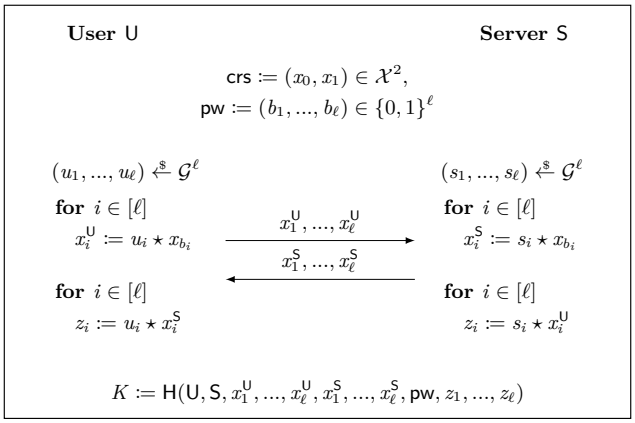

Fig. 1. First Attempt: Protocol $\mathsf{GA\text{-}PAKE}_\ell$.

to the user. Note that the messages may be sent simultaneously in one round. Then both parties compute $z_i = u_i \star x_i^{\mathsf{S}} = s_i \star x_i^{\mathsf{U}}$ for each $i \in [\ell]$. Finally the session key K is computed as $K = \mathsf{H}(\mathsf{U}, \mathsf{S}, x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}, x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}, \mathsf{pw}, z_1, ..., z_\ell)$, where $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ is a hash function into the key space $\mathcal{K}$.

In Section 5, we present an offline dictionary attack against $\mathsf{GA\text{-}PAKE}_\ell$ for group actions with twists. This attack is not captured by the abstract group action framework defined in [5] which underlines the necessity of our suggested enhancement of the framework. Roughly speaking, the attack uses the fact that an attacker can choose its message in dependence on the other party's message. Using twists, it can then achieve that certain terms in the key derivation cancel out and the session key no longer depends on the other party's input.

Secure PAKE: $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ and $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$. The protocol $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ is a modified version of $\mathsf{GA\text{-}PAKE}_\ell$. Here security is achieved by doubling the message length in the first round of the protocol and tripling it in the key derivation. Intuitively the additional parts of the message can be viewed as an additional challenge for the key derivation that inhibits an attacker from choosing its message depending on the other party's message. The security of the protocol relies on a new computational assumption, $\mathsf{SqInv\text{-}GA\text{-}StCDH}$, in which the adversary needs to compute the square and the inverse of its input at the same time (cf. Definition 7, Theorem 1).

The protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ is a modification of $\mathsf{GA\text{-}PAKE}_\ell$ as well. In order to achieve security against offline dictionary attacks, the protocol requires that the server sends a commitment before receiving the first message from the user. This prevents that any party chooses its message depending on the other party's message. We reduce the security of the protocol to the hardness of standard security assumptions in the isogeny-based setting (Theorem 2). An overview of our results is provided in Figure 2.

Optimizations. Both $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ and $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ require to compute multiple group action evaluations. In the last section, we discuss two optimiza-

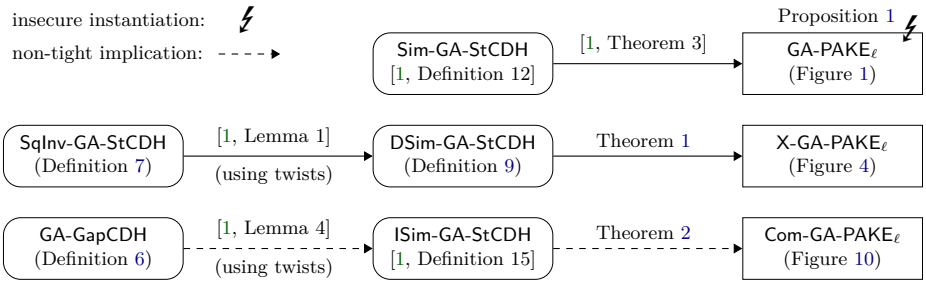

Fig. 2. Overview of our security implications between assumptions (round boxes) and schemes (square boxes). Note that there exists an attack against protocol GA-PAKE$_\ell$ using twists which makes it insecure for CSIDH. Our two main protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ are proven secure under protocol-specific assumptions, but we also give reductions to simpler assumptions making use of the twisting property. Solid arrows denote tight reductions, dashed arrows non-tight reductions.

tions that can be used to reduce the number of evaluations and show that these do not affect the security of the protocols. The first makes a tradeoff between the size of the public parameters (the common reference string crs) and the number of elements that have to be sent as well as the group actions that have to be performed. The second optimization relies on the possibility to compute twists efficiently, which is yet another advantage of adding this property to the framework and which allows to decrease the size of the public parameters by a factor of 2. We denote the final optimizations by Com-GA-PAKE$^{\mathsf{t}}_{\ell,N}$ and X-GA-PAKE$^{\mathsf{t}}_{\ell,N}$, where N is a parameter for the crs size. If N equals 1, we omit it. An overview and example of the parameter choice is provided in Table 1.

Difficulties in constructing PAKE from Isogenies. Terada and Yoneyama [30] proposed isogeny-based PAKE based on the EKE approach. The basic idea is that the parties perform an SIDH or CSIDH key exchange where the messages are encrypted with the password. However, as shown in [6], these protocols are not only vulnerable to offline dictionary attacks, but a modified version is even vulnerable to man-in-the-middle attacks. The main reason for the insecurity is that the elliptic curves used in the key exchange and encrypted with the password are distinguishable from random bitstrings. An exhaustive search over all passwords just requires to check if the decrypted message is a valid curve.

Another proposal based on SIDH was made by Taraskin et al. [29]. In this protocol the password is used to obfuscate the auxiliary points that are exchanged during an SIDH key exchange. While their obfuscation method prevents a certain type of offline dictionary attack, the authors were not able to provide a security proof for their protocol. The same is true for a symmetric variant of the protocol proposed by Soukharev and Hess [28]. Until now these are the only PAKE protocol based on isogenies which are not broken.

As noted in [6], other popular Diffie-Hellman constructions may also not be directly translated into the isogeny setting. The main reason is that hashing into

Protocol	$\lvert\mathsf{crs}\rvert$	Elements	Evaluations	Rounds	Assumption	Rew.	ROM
$\mathsf{X\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$	2^{N-1}	$2\ell/N$	$5\ell/N$	1	SqInv-GA-StCDH	no	yes
$\hookrightarrow (\ell, N) = (128, 8)$	128	32	80				
$\mathsf{Com\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$	2^{N-1}	ℓ/N (+1)	$2\ell/N$	3	Sq-GA-GapCDH	yes	yes
$\hookrightarrow (\ell, N) = (128, 8)$	128	16 (+1)	32				
OT-based$_\ell$ [24,10]	1	3ℓ ($+6\ell$)	11ℓ	4	GA-CDH	yes	yes
$\hookrightarrow \ell = 128$	1	384 (+768)	1408				
OT-based$_\ell$ [5,25,10]	4	$> \ell^2$	$> \ell^2$	3	GA-DDH + CCA PKE	no	no
$\hookrightarrow \ell = 128$	4	$> 16,000$	$> 16,000$				

Table 1. Overview of our two optimized protocols $\mathsf{Com\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$ and $\mathsf{X\text{-}GA\text{-}PAKE}^{\mathsf{t}}_{\ell,N}$ and comparison to the only other CSIDH-based constructions. All protocols use a bit-wise approach, i.e., passwords are treated as bitstrings of length ℓ. Sample values for $\ell = 128$ are marked in gray. "Elements" refers to the number of set elements (+ strings or symmetric ciphertexts) that each party has to send. "Evaluations" refers to the number of group action evaluations that each party has to perform. "Rew." indicates that rewinding is used to reduce to the assumption indicated in the table and GA-DDH refers to the group action decisional Diffie-Hellman problem.

the set of supersingular elliptic curves is still an open problem. This approach is for example used in SPEKE. (However, we show how to non-trivially translate the idea.) Also the approach of J-PAKE seems difficult as in this scheme different public keys are combined to obtain certain "mixed" public keys. In isogeny-based protocols, the public keys are elliptic curves and there is no natural ring structure on the set of elliptic curves that would allow to combine two elliptic curves.

In the following, we elaborate known generic constructions of PAKE from hash proof systems (HPS) and oblivious transfer (OT). We explain that the only known isogeny-based HPS is not suitable for generic constructions. On the other hand, the isogeny-based OT protocols from the literature are suited for generic constructions. However, we show that the resulting PAKE protocols are less efficient than our new proposals.

Using the framework of cryptographic group actions, Alamati et al. construct a universal hash proof system [5, §4.1]. In general, it is known how to build CCA-secure encryption [13] and also PAKE from hash proof systems [17]. However, the details here are less clear. The hashing key consists of multiple elements linear in the size of the universality parameter. The reason being that we can only make use of one group operation provided by the group action. This also needs to be considered when constructing an encryption scheme. In order to construct PAKE, the framework by Gennaro and Lindell and follow-up works require a hash proof system for the language of ciphertexts of a public-key encryption scheme, which seems to be hard to construct given only the operation of the group action. The HPS in [5] is only based on a DDH-like assumption.

It is well known that PAKE can also be generically constructed from OT [10]. One construction uses a UC-secure OT protocol and the other one a statistically receiver-private OT protocol. In both, the password is interpreted as a bit string

and for each bit, the user and server run the oblivious transfer protocol for randomly chosen messages which will be used to derive the session key. In the following, we apply the construction to two existing OT protocols.

- Alamati et al. propose a two-message statistically *sender*-private OT, however we can construct a similar receiver-private OT protocol based on their dual-mode public-key encryption scheme and the transformation given in [25]. The resulting OT protocol already uses a "bit-by-bit" approach, hence the resulting PAKE will have communication and computation complexity quadratic in the parameter ℓ.
- Recently, Lai et al. proposed a new very efficient CSIDH-based OT protocol using twists and the random oracle model [24]. However, in order to achieve active security the protocol needs four rounds.[4] Additionally applying the generic PAKE compiler results in a protocol with complexity linear in ℓ.

The efficiency of the generic constructions is compared to our new protocols in Table 1. While the computational cost of our protocols $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell,N}$ and $\mathsf{X\text{-}GA\text{-}PAKE}_{\ell,N}$ is also linear in ℓ, the cost is considerably lower for concrete instantiations. Moreover, our scheme $\mathsf{X\text{-}GA\text{-}PAKE}_{\ell,N}$ is the only one-round protocol, where both parties send simultaneous flows.

Open Problems and Future Work. Until now, protocols based on CSIDH or group actions that use search problems together with the random oracle model do not consider quantum access to the ROM [31,16,22,23,24]. Since PAKE proofs are already complex, we also did not prove security in the QROM. Although no reprogramming of the random oracle is necessary, the main difficulty in the QROM is to simulate the real session keys using the decision oracle. We leave this as future work. We believe that we can easily allow quantum access to the additional random oracle that is used in $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell}$ to commit on the message. In this case, the output is transferred classically in the first message flow such that extraction is possible using recently developed techniques [15].

As [24], we use rewinding to reduce the interactive assumption underlying $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell}$ to a standard assumption. An interesting open question is whether current techniques enabling quantum rewinding are applicable here.

Outline. Section 3 sets the framework for our paper. We introduce (restricted) effective group actions with twists and define the computational assumptions underlying the security of our protocols. In Section 4, we give some background on the security model that is used in the subsequent sections. In Section 5 we present our first attempt for a PAKE protocol, $\mathsf{GA\text{-}PAKE}_{\ell}$, and explain its security gap. Section 6 contains a thorough analysis of our new secure protocol $\mathsf{X\text{-}GA\text{-}PAKE}_{\ell}$. In Section 7 we present the protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_{\ell}$ and sketch the security proof. Finally, we discuss possible optimizations of the protocols in Section 8.

[4] The original (three-round) version of this protocol was later found to have a (fixable) bug, cf. https://iacr.org/submit/files/slides/2021/eurocrypt/eurocrypt2021/20/slides.pdf.

2 Preliminaries

For integers m, n where $m < n$, $[m, n]$ denotes the set $\{m, m+1, ..., n\}$. For $m = 1$, we simply write $[n]$. For a set S, $s \xleftarrow{\$} S$ denotes that s is sampled uniformly and independently at random from S. $y \leftarrow \mathcal{A}(x_1, x_2, ...)$ denotes that on input $x_1, x_2, ...$ the probabilistic algorithm $\mathcal{A}$ returns y. $\mathcal{A}^{\mathrm{O}}$ denotes that algorithm $\mathcal{A}$ has access to oracle O. An adversary is a probabilistic algorithm. We will use code-based games, where $\Pr[\mathsf{G} \Rightarrow 1]$ denotes the probability that the final output of game G is 1.

3 (Restricted) Effective Group Actions (with Twists)

In this section we recall the definition of (restricted) effective group actions from [5], which provides an abstract framework to build cryptographic primitives relying on isogeny-based assumptions such as CSIDH. Moreover, we suggest an enhancement of this framework, by introducing (restricted) effective group actions with twists. This addition is essential for the security analysis of our new PAKE protocols.

Definition 1 (Group Action). *Let $(\mathcal{G}, \cdot)$ be a group with identity element $id \in \mathcal{G}$, and $\mathcal{X}$ a set. A map*

$$\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$$

is a group action if it satisfies the following properties:

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

Remark 1. Throughout this paper, we only consider group actions, where $\mathcal{G}$ is commutative. Moreover we assume that the group action is regular. This means that for any $x, y \in \mathcal{X}$ there exists precisely one $g \in \mathcal{G}$ satisfying $y = g \star x$.

Definition 2 (Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action satisfying the following properties:*

1. *The group $\mathcal{G}$ is finite and there exist efficient (PPT) algorithms for membership and equality testing, (random) sampling, group operation and inversion.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm to evaluate the group action, i.e. to compute $g \star x$ given g and x.*

Then we call $\tilde{x} \in \mathcal{X}$ the origin and $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ an effective group action (EGA).

In practice, the requirements from the definition of EGA are often too strong. Therefore we will consider the weaker notion of restricted effective group actions.

Definition 3 (Restricted Effective Group Action). *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a group action and let $\boldsymbol{g} = (g_1, ..., g_n)$ be a generating set for $\mathcal{G}$. Assume that the following properties are satisfied:*

1. *The group $\mathcal{G}$ is finite and $n = poly(\log(\#\mathcal{G}))$.*
2. *The set $\mathcal{X}$ is finite and there exist efficient algorithms for membership testing and to compute a unique representation.*
3. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$ with known representation.*
4. *There exists an efficient algorithm that given $g_i \in \boldsymbol{g}$ and $x \in \mathcal{X}$, outputs $g_i \star x$ and $g_i^{-1} \star x$.*

Then we call $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ a restricted effective group action (REGA).

3.1 Isogeny-based REGAs

An important instantiation of REGAs is provided by isogeny-based group actions. We will focus on the CSIDH setting and present a refined definition of REGAs tailored to this situation.

Let p be a large prime of the form $p = 4 \cdot \ell_1 \cdots \ell_n - 1$, where the ℓ_i are small distinct odd primes. Fix the elliptic curve $E_0 : y^2 = x^3 + x$ over $\mathbb{F}_p$. The curve E_0 is supersingular and its $\mathbb{F}_p$-rational endomorphism ring is $\mathcal{O} = \mathbb{Z}[\pi]$, where π is the Frobenius endomorphism. Let $\mathcal{E}\ell\ell_p(\mathcal{O})$ be the set of elliptic curves defined over $\mathbb{F}_p$, with endomorphism ring $\mathcal{O}$. The ideal class group $cl(\mathcal{O})$ acts on the set $\mathcal{E}\ell\ell_p(\mathcal{O})$, i.e., there is a map

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

satisfying the properties from Definition 1 [11, Theorem 7]. Moreover the analysis in [11] readily shows that $(cl(\mathcal{O}), \mathcal{E}\ell\ell_p(\mathcal{O}), \star, E_0)$ is indeed a REGA.

Elliptic curves in $\mathcal{E}\ell\ell_p(\mathcal{O})$ admit equations of the form $E_A : y^2 = x^3 + Ax^2 + x$, which allows to represent them by their Montgomery coefficient $A \in \mathbb{F}_p$. An intrinsic property of the CSIDH group action which is not covered by Definition 3, is the following. For any curve $E_A = [\mathfrak{a}] \star E_0 \in \mathcal{E}\ell\ell_p(\mathcal{O})$, its quadratic twist is easily computed as $(E_A)^t = E_{-A}$ and satisfies the property $(E_A)^t = [\mathfrak{a}]^{-1} \star E_0$.

Definition 4 ((Restricted) Effective Group Action with Twists). *We say that a* (R)EGA $(\mathcal{G}, \mathcal{X}, \star, \tilde{x})$ *is a (Restricted) Effective Group Action with Twists* ((R)EGAT) *if there exists an efficient algorithm that given $x = g \star \tilde{x} \in \mathcal{X}$ computes $x^t = g^{-1} \star \tilde{x}$.*

As noted in [11, §10], this property contrasts with the classical group-based setting. It has already been used for the design of new cryptographic primitives based on CSIDH such as the signature scheme CSIFiSh [9] and the OT protocol in [24]. Moreover, it is important to consider twists in the security analysis of schemes based on group actions. In Section 5 we use twists to construct an attack on the protocol GA-PAKE$_\ell$ showing that it cannot be securely instantiated with the CSIDH group action. On the other hand, we prove that GA-PAKE$_\ell$ is secure when instantiated with a group action without efficient twisting. The proof for that is given in the full version [1, Appendix C].

3.2 Computational Assumptions

For cryptographic applications, we are interested in (restricted) effective group actions that are equipped with the following hardness properties:

- Given $(x, y) \in \mathcal{X}^2$, it is hard to find $g \in \mathcal{G}$ such that $y = g \star x$.
- Given $(x, y_0, y_1) \in \mathcal{X}^3$, it is hard to find $z = (g_0 \cdot g_1) \star x$, where $g_0, g_1 \in \mathcal{G}$ are such that $y_0 = g_0 \star x$ and $y_1 = g_1 \star x$.

In [5] such group actions are called cryptographic group actions, and in [12] they are called hard homogeneous spaces.

The two hardness assumptions are the natural generalizations of the discrete logarithm assumption and the Diffie-Hellman assumption in the traditional group based setting. In analogy to this setting, we introduce the notation

$$\mathsf{GA\text{-}CDH}_x(y_0, y_1) = g_0 \star y_1, \qquad \text{where } g_0 \in \mathcal{G} \text{ such that } y_0 = g_0 \star x$$

and define the decision oracle

$$\text{GA-DDH}_x(y_0, y_1, z) = \begin{cases} 1 & \text{if } \mathsf{GA\text{-}CDH}_x(y_0, y_1) = z, \\ 0 & \text{otherwise.} \end{cases}$$

For both, GA-CDH and GA-DDH, we omit the index x if $x = \tilde{x}$, i. e., we set $\mathsf{GA\text{-}CDH}_{\tilde{x}}(y_0, y_1) = \mathsf{GA\text{-}CDH}(y_0, y_1)$ and equivalently for $\text{GA-DDH}_{\tilde{x}}(y_0, y_1, z)$.

We now introduce three computational problems GA-StCDH, GA-GapCDH, SqInv-GA-StCDH (Definitions 5 to 7). The security of our PAKE protocols relies on the hardness of these problems.

The first two problems are variants of the standard Diffie-Hellman problem, where an adversary is either given access to some fixed-basis decision oracles (indicated by the prefix *strong*) or to a general decision oracle (indicated by the prefix *gap*). Note that these problems were already defined and used in previous work [31,16,22,23]. Since the problem from Definition 7 has not been studied in any previous work, we provide evidence for its hardness in Remark 3.

Definition 5 (Group Action Strong Computational Diffie-Hellman Problem (GA-StCDH)). *On input $(g \star \tilde{x}, h \star \tilde{x}) \in \mathcal{X}^2$, the* GA-StCDH *problem requires to compute the set element $(g \cdot h) \star \tilde{x}$. To an effective group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$, we associate the advantage function of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr[\mathcal{A}^{\text{GA-DDH}(g\star\tilde{x},\cdot,\cdot)}(g \star \tilde{x}, h \star \tilde{x}) \Rightarrow (g \cdot h) \star \tilde{x}] \ ,$$

where $(g, h) \xleftarrow{\$} \mathcal{G}^2$ and $\mathcal{A}$ has access to decision oracle $\text{GA-DDH}(g \star \tilde{x}, \cdot, \cdot)$.

Definition 6 (Group Action Gap Computational Diffie-Hellman Problem (GA-GapCDH)). *On input $(g \star \tilde{x}, h \star \tilde{x}) \in \mathcal{X}^2$, the* GA-GapCDH *problem requires to compute the set element $(g \cdot h) \star \tilde{x}$. To an effective group action $\mathsf{XXX} \in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$, we associate the advantage function of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr[\mathcal{A}^{\text{GA-DDH}_*}(g \star \tilde{x}, h \star \tilde{x}) \Rightarrow (g \cdot h) \star \tilde{x}] \ ,$$

where $(g, h) \xleftarrow{\$} \mathcal{G}^2$ and $\mathcal{A}$ has access to a general decision oracle GA-DDH_*.

Remark 2. A group action where the group action computational Diffie-Hellman problem (without any decision oracle) is hard, is the same as a weak unpredictable group action as defined by Alamati et al. [5]. Further details are given in the full version [1, Appendix A]. Also note that the ability to compute the twist of a set element does not help in solving these problems. Hence, all results based on these problems remain true for (R)EGAT.

Definition 7 (Square-Inverse GA-StCDH (SqInv-GA-StCDH)). *On input $x = g \star \tilde{x}$, the* SqInv-GA-StCDH *problem requires to find a tuple $(y, y_0, y_1) \in \mathcal{X}^3$ such that $y_0 = g^2 \star y$ and $y_1 = g^{-1} \star y$. For a group action* XXX $\in \{$EGA, REGA, EGAT, REGAT$\}$*, we define the advantage function of $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) := \Pr\left[\begin{matrix} y_0 = \mathsf{GA\text{-}CDH}_{x^t}(x, y) \\ y_1 = \mathsf{GA\text{-}CDH}(x^t, y) \end{matrix} \;\middle|\; \begin{matrix} g \xleftarrow{\$} \mathcal{G} \\ x = g \star \tilde{x} \\ (y, y_0, y_1) \leftarrow \mathcal{A}^{\mathrm{O}}(x) \end{matrix}\right],$$

where $\mathrm{O} = \{\mathrm{GA\text{-}DDH}_{x^t}(x, \cdot, \cdot), \mathrm{GA\text{-}DDH}(x, \cdot, \cdot)\}$.

Remark 3. Intuitively SqInv-GA-StCDH is hard if we assume that the adversary can only use the group and twist operation. To go into more detail, $\mathcal{A}$ can choose y only based on known elements, that is either based on $\tilde{x}$, its input x or x^t.

If $\mathcal{A}$ chooses $y = \alpha \star \tilde{x}$ for some $\alpha \in \mathcal{G}$, then it can easily compute $y_1 = \alpha \star x^t$, but not $y_0 = \alpha g^2 \star \tilde{x}$. If $\mathcal{A}$ chooses $y = \alpha \star x$, then computing $y_1 = \alpha \star \tilde{x}$ is trivial, but computing $y_0 = \alpha g^3 \star \tilde{x}$ is hard. If $\mathcal{A}$ chooses $y = \alpha \star x^t$, then computing $y_0 = \alpha \star x$ is trivial, but computing $y_1 = \alpha g^{-2} \star \tilde{x}$ is hard.

4 Password Authenticated Key Exchange

Password-authenticated key exchange (PAKE) allows two parties, typically referred to as the user and the server, to establish a shared session key with the help of a short secret, known as a password, which can be drawn from a small set of possible values. To prove security of a PAKE protocol, we use the indistinguishability-based model by Bellare, Pointcheval and Rogaway [7] and its extension to multiple test queries by Abdalla, Fouque and Pointcheval [2].

The name spaces for users $\mathcal{U}$ and servers $\mathcal{S}$ are assumed to be disjoint. Each pair of user and server $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$ holds a shared password $\mathsf{pw}_{\mathsf{US}}$. A party P denotes either a user or server. Each party P has multiple instances π^i_{P} and each instance has its own state. We denote the session key space by $\mathcal{K}$. Passwords are bit strings of length ℓ and we define the password space as $\mathcal{PW} \subsetneq \{0,1\}^\ell$.

Instance State. The state of an instance π^i_{P} is a tuple $(\mathsf{e}, \mathsf{tr}, K, \mathrm{acc})$ where
- e stores the (secret) ephemeral values chosen by the party in that instance.
- tr stores the trace of that instance, i.e., the user and server name involved in the protocol execution and the messages sent and received by that instance.
- K is the accepted session key.
- acc is a Boolean flag that indicates whether the instance has accepted the session key. As long as the instance did not receive the last message, $\mathrm{acc} = \bot$.

To access individual components of the state, we write $\pi^t_{\mathsf{P}}.\{\mathsf{e}, \mathsf{tr}, K, \mathrm{acc}\}$.

Partnering. Partnering is defined via matching conversations. In particular, a user instance $\pi_{\mathsf{U}}^{t_0}$ and a server instance $\pi_{\mathsf{S}}^{t_1}$ are partnered iff

$$\pi_{\mathsf{U}}^{t_0}.\mathrm{acc} = \pi_{\mathsf{S}}^{t_1}.\mathrm{acc} = \textbf{true} \quad \textbf{and} \quad \pi_{\mathsf{U}}^{t_0}.\mathsf{tr} = \pi_{\mathsf{S}}^{t_1}.\mathsf{tr} \ .$$

Two user instances are never partnered, neither are two server instances. We define a partner predicate $\mathsf{Partner}(\pi_{\mathsf{P}_0}^{t_0}, \pi_{\mathsf{P}_1}^{t_1})$ which outputs 1 if the two instances $\pi_{\mathsf{P}_0}^{t_0}$ and $\pi_{\mathsf{P}_1}^{t_1}$ are partnered and 0 otherwise.

Security Experiment. The security experiment is played between a challenger and an adversary $\mathcal{A}$. The challenger draws a random challenge bit β and creates the public parameters. Then it outputs the public parameters to $\mathcal{A}$. Now $\mathcal{A}$ has access to the following oracles:

- $\textsc{Execute}(\mathsf{U}, t_0, \mathsf{S}, t_1)$: one complete protocol execution between user instance $\pi_{\mathsf{U}}^{t_0}$ and server instance $\pi_{\mathsf{S}}^{t_1}$. This query models security against passive adversaries.
- SendInit, SendResp, SendTermInit, SendTermResp: send oracles to model security against active adversaries. SendTermResp is only available for three-message protocols.
- $\textsc{Corrupt}(\mathsf{U}, \mathsf{S})$: outputs the shared password $\mathsf{pw}_{\mathsf{US}}$ of U and S.
- $\textsc{Reveal}(\mathsf{P}, t)$: outputs the session key of instance π_{P}^{t}.
- $\textsc{Test}(\mathsf{P}, t)$: challenge query. Depending on the challenge bit β, the experiment outputs either the session key of instance π_{P}^{t} or a uniformly random key. By $\pi_{\mathsf{P}}^{t}.\mathrm{test} = \textbf{true}$, we mark an instance as tested.

We denote the experiment by $\mathsf{Exp}_{\mathsf{PAKE}}$. The pseudocode is given in G_0 in Figure 5, instantiated with our first PAKE protocol.

Freshness. During the game, we register if a query is allowed to prevent trivial wins. Therefore, we define a freshness predicate $\mathsf{Fresh}(\mathsf{P}, i)$. An instance π_{P}^{t} is fresh iff

1. π_{P}^{t} accepted.
2. π_{P}^{t} was not queried to Test or Reveal before.
3. At least one of the following conditions holds:
   - 3.1 π_{P}^{t} accepted during a query to Execute.
   - 3.2 There exists more than one partner instance.
   - 3.3 A unique fresh partner instance exists.
   - 3.4 No partner exists and Corrupt was not queried.

Definition 8 (Security of PAKE). *We define the security experiment, partnering and freshness conditions as above. The advantage of an adversary $\mathcal{A}$ against a password authenticated key exchange protocol* PAKE *in* $\mathsf{Exp}_{\mathsf{PAKE}}$ *is defined as*

$$\mathsf{Adv}_{\mathsf{PAKE}}(\mathcal{A}) := \left| \Pr[\mathsf{Exp}_{\mathsf{PAKE}} \Rightarrow 1] - \frac{1}{2} \right| \ .$$

A PAKE is considered secure if the best the adversary can do is to perform an online dictionary attack. More concretely, this means that the advantage of the adversary should be negligibly close to $q_s/|\mathcal{PW}|$ when passwords are drawn uniformly and independently from $\mathcal{PW}$, where q_s is the number of send queries made by the adversary.

Note that this definition captures weak forward secrecy. In the full version of our paper, we give an extended security definition capturing also perfect forward secrecy, as well as proofs for our protocols [1, Appendix F].

5 First Attempt: Protocol GA-PAKE$_\ell$

The GA-PAKE$_\ell$ protocol was already introduced in the introduction (Section 1). We refer to Figure 1 for a description of the protocol. In contrast to the two PAKE protocols from Sections 6 and 7, GA-PAKE$_\ell$ is not secure for EGATs, i.e., if it is possible to compute twists of set elements efficiently. In particular it should not be instantiated with the CSIDH-group action. However, it is instructive to examine its security and it serves as a good motivation for the design of the two secure PAKE protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$.

In this section we present an offline dictionary attack against GA-PAKE$_\ell$ for (R)EGAT. However, if twisting is hard, then we can prove security of GA-PAKE$_\ell$ based on a hardness assumption that is similar to the simultaneous Diffie-Hellman problem which was introduced to prove the security of TBPEKE and CPace [26,3]. The proof for GA-PAKE$_\ell$ is given in the full version [1, Appendix C].

Proposition 1. *For* EGAT*s, the protocol* GA-PAKE$_\ell$ *is vulnerable to offline dictionary attacks.*

Proof. We construct an adversary $\mathcal{A}$ that takes the role of the server. The attack is summarized in Figure 3. After receiving x^{U}, the adversary computes

$$x_i^{\mathsf{S}} = \tilde{s}_i \star (x_i^{\mathsf{U}})^t = \tilde{s}_i \star (u_i \star x_{b_i})^t = (\tilde{s}_i \cdot u_i^{-1}) \star x_{b_i}^t = (\tilde{s}_i \cdot u_i^{-1} \cdot g_{b_i}^{-1}) \star \tilde{x}$$

for each $i \in [\ell]$ and sends $x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}$ to the user. Then the user computes $z_i = u_i \star x_i^{\mathsf{S}} = (\tilde{s}_i \cdot g_{b_i}^{-1}) \star \tilde{x} = \tilde{s}_i \star x_{b_i}^t$. For each $i \in [\ell]$, the adversary $\mathcal{A}$ can now compute z_i for both possibilities $b_i = 0$ and $b_i = 1$. This allows him to compute K for all possible passwords $\mathsf{pw} \in \mathcal{PW} \subsetneq \{0,1\}^\ell$ (being offline). □

This offline attack can easily be used to win the security experiment with high probability. $\mathcal{A}$ only needs to issue two send queries. It chooses any user U, initiates a session and computes its message $x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}$ as described in Figure 3. It reveals the corresponding session key and starts its offline attack by brute forcing all $\mathsf{pw} \in \mathcal{PW}$ until it finds a match for a candidate pw^*. Now $\mathcal{A}$ issues its second send query. This time it computes the message following the protocol using pw^* and derives a key K^*. It issues a test query and gets K_β. If $K^* = K_\beta$, then it outputs 0, otherwise it outputs 1. In case there is more than one password

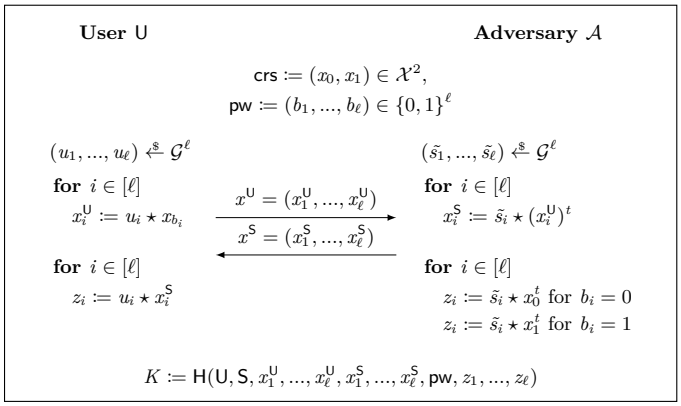

Fig. 3. Attack against $\mathsf{GA\text{-}PAKE}_\ell$ using twists.

candidate, i.e., two inputs to H lead to the same K^*, then $\mathcal{A}$ can issue another send and reveal query to rule out false positives. In the end, it can still happen that $\beta = 1$ and $K^* = K$, but this event only occurs with probability $1/|\mathcal{K}|$.

Corollary 1. *For any adversary $\mathcal{A}$ against $\mathsf{GA\text{-}PAKE}_\ell$ instantiated with an EGAT, we have* $\Pr[\mathsf{Exp}_{\mathsf{GA\text{-}PAKE}_\ell} \Rightarrow 1] = 1 - \frac{1}{|\mathcal{K}|}$.

6 X-GA-PAKE$_\ell$: One-Round PAKE from Group Actions

In the previous section we showed that $\mathsf{GA\text{-}PAKE}_\ell$ is insecure when instantiated with an EGAT. Here, we present the modification $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$, which impedes the offline dictionary attack presented in that section. Broadly speaking, the idea is to double the message size of both parties in the first flow. In the second flow it is then necessary to compute certain "cross products" which is only possible if the previous message has been honestly generated. The letter X in $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ stands for cross product.

By means of these modifications, the protocol $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ is provably secure for EGATs. We show that its security can be reduced to the hardness of the computational problems $\mathsf{GA\text{-}StCDH}$ and $\mathsf{SqInv\text{-}GA\text{-}StCDH}$ (Theorem 1).

The setup for $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ is the same as for $\mathsf{GA\text{-}PAKE}_\ell$. The $\mathsf{crs} = (x_0, x_1)$ comprises two elements of the set $\mathcal{X}$, and the shared password is a bit string $(b_1, \ldots, b_\ell)$ of length ℓ. In the first flow of the protocol the user generates $2 \cdot \ell$ random group elements, $u_1, \ldots, u_\ell$ and $\hat{u}_1, \ldots, \hat{u}_\ell$. Using these elements it computes the set elements $x_i^{\mathsf{U}} = u_i \star x_{b_i}$ and $\hat{x}_i^{\mathsf{U}} = \hat{u}_i \star x_{b_i}$ for each $i \in [\ell]$ and sends these to the server. Simultaneously, the server generates the random group elements $s_1, \ldots, s_\ell$ and $\hat{s}_1, \ldots, \hat{s}_\ell$, which it uses to compute the set elements $x_i^{\mathsf{S}} = s_i \star x_{b_i}$ and $\hat{x}_i^{\mathsf{S}} = \hat{s}_i \star x_{b_i}$ for each $i \in [\ell]$ and sends these to the user. Upon receiving the set elements from the other party, both the server and the user compute

$$z_{i,1} = u_i \star x_i^{\mathsf{S}} = s_i \star x_i^{\mathsf{U}}, \quad z_{i,2} = \hat{u}_i \star x_i^{\mathsf{S}} = s_i \star \hat{x}_i^{\mathsf{U}}, \quad z_{i,3} = u_i \star \hat{x}_i^{\mathsf{S}} = \hat{s}_i \star x_i^{\mathsf{U}},$$

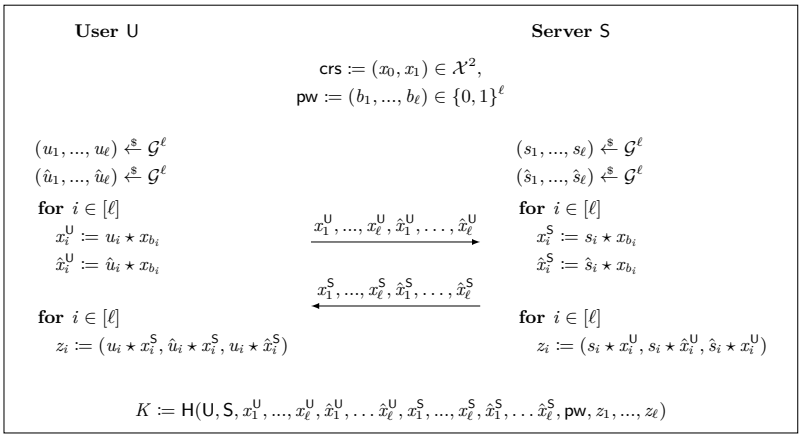

Fig. 4. PAKE protocol X-GA-PAKE$_\ell$ from group actions.

for each $i \in [\ell]$. Finally, these elements are used to compute the session key K. The protocol is sketched in Figure 4.

We now prove the security of X-GA-PAKE$_\ell$ for EGATs.

Theorem 1 (Security of X-GA-PAKE$_\ell$). *For any adversary $\mathcal{A}$ against* X-GA-PAKE$_\ell$ *that issues at most q_e execute queries and q_s send queries and where* H *is modeled as a random oracle, there exist an adversary $\mathcal{B}_1$ against* GA-StCDH *and an adversary $\mathcal{B}_2$ against* SqInv-GA-StCDH *such that*

$$\mathsf{Adv}_{\mathsf{X\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2) + \frac{q_s}{|\mathcal{PW}|} + \frac{(q_s+q_e)^2}{|\mathcal{G}|^{2\ell}} \ .$$

Before proving Theorem 1, we will introduce a new computational assumption which is tailored to the protocol.

Definition 9 (Double Simultaneous GA-StCDH **(**DSim-GA-StCDH**)).** *On input $(x_0, x_1, w_0, w_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x}) \in \mathcal{X}^4$, the* DSim-GA-StCDH *problem requires to find a tuple $(y, y_0, y_1, y_2, y_3) \in \mathcal{X}^5$ such that*

$$(y_0, y_1, y_2, y_3) = (g_0^{-1} \cdot h_0 \star y,\ g_0^{-1} \cdot h_1 \star y,\ g_1^{-1} \cdot h_0 \star y,\ g_1^{-1} \cdot h_1 \star y).$$

For a group action XXX $\in \{\mathsf{EGA}, \mathsf{REGA}, \mathsf{EGAT}, \mathsf{REGAT}\}$*, we define the advantage function of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{XXX}}(\mathcal{A}) \coloneqq \Pr\left[\begin{matrix} y_0 = \mathsf{GA\text{-}CDH}_{x_0}(w_0, y) \\ y_1 = \mathsf{GA\text{-}CDH}_{x_0}(w_1, y) \\ y_2 = \mathsf{GA\text{-}CDH}_{x_1}(w_0, y) \\ y_3 = \mathsf{GA\text{-}CDH}_{x_1}(w_1, y) \end{matrix} \middle| \begin{matrix} (g_0, g_1, h_0, h_1) \xleftarrow{\$} \mathcal{G}^4 \\ (x_0, x_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}) \\ (w_0, w_1) = (h_0 \star \tilde{x}, h_1 \star \tilde{x}) \\ (y, y_0, y_1, y_2, y_3) \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1, w_0, w_1) \end{matrix}\right],$$

where $\mathrm{O} = \{\mathrm{GA\text{-}DDH}_{x_j}(w_i, \cdot, \cdot)\}_{i,j \in \{0,1\}}$.

Remark 4. Note that DSim-GA-StCDH may be viewed as the doubled version of the Sim-GA-StCDH problem defined in the full version of the paper [1, Definition

12]. The latter is an assumption underlying the security of $\mathsf{GA\text{-}PAKE}_\ell$ and (in the notation of the above problem) it only requires to find the tuple (y, y_0, y_2). For a group action with twists, this admits the trivial solution $(y, y_0, y_2) = (w_0^t, x_0^t, x_1^t)$. Such a trivial solution is inhibited by requiring to find y_1 and y_3 as well.

The $\mathsf{DSim\text{-}GA\text{-}StCDH}$ problem is implied by $\mathsf{SqInv\text{-}GA\text{-}StCDH}$, more precisely

$$\mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SqInv\text{-}GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}) \ . \tag{1}$$

A proof of this implication is given in the full version [1, Lemma 1].

Proof (of Theorem 1). Let $\mathcal{A}$ be an adversary against $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$. Consider the games in Figures 5, 7, 8.

Game G_0. This is the original game, hence

$$\mathsf{Adv}_{\mathsf{X\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq |\Pr[\mathsf{G}_0 \Rightarrow 1] - 1/2| \ .$$

Game G_1. In game G_1, we raise flag $\mathbf{bad}_{\text{coll}}$ whenever a server instance computes the same trace as any other accepted instance (line 69) or a user instance computes the same trace as any other accepted user instance (line 84). In this case, SendResp or SendTermInit return $\perp$. We do the same if a trace that is computed in an Execute query collides with one of a previously accepted instance (line 28). Due to the difference lemma,

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{coll}}] \ .$$

Note that when $\mathbf{bad}_{\text{coll}}$ is not raised, each instance is unique and has at most one partner. In order to bound $\mathbf{bad}_{\text{coll}}$, recall that the trace of an oracle π_{P}^t consists of $(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}} = (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}), \hat{x}^{\mathsf{U}} = (\hat{x}_1^{\mathsf{U}}, ...\hat{x}_\ell^{\mathsf{U}}), x^{\mathsf{S}} = (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}), \hat{x}^{\mathsf{S}} = (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}))$, where at least one of the message pairs $(x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$ or $(x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ was chosen by the game. Thus, $\mathbf{bad}_{\text{coll}}$ can only happen if all those $2 \cdot \ell$ set elements collide with all $2 \cdot \ell$ set elements of another instance. The probability that this happens for two (fixed) sessions is $|\mathcal{G}|^{-2\ell}$, hence the union bound over q_e and q_s sessions yields

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \Pr[\mathbf{bad}_{\text{coll}}] \leq \binom{q_e + q_s}{2} \cdot \frac{1}{|\mathcal{G}|^{2\ell}} \leq \frac{(q_e + q_s)^2}{|\mathcal{G}|^{2\ell}} \ .$$

Game G_2. In game G_2, we make the freshness explicit. To each oracle π_{P}^t, we assign an additional variable $\pi_{\mathsf{P}}^t.\mathsf{fr}$ which is updated during the game. In particular, all instances used in execute queries are marked as fresh (line 34).

An instance is fresh if the password was not corrupted yet (lines 72, 89). Otherwise, it is not fresh (lines 74, 91). For user instances we also check if there exists a fresh partner (line 87). If $\mathcal{A}$ issues a Corrupt query later, the freshness variable will also be updated (line 103). When the session key of an instance is revealed, this instance and its potential partner instance are marked as not fresh (line 41). On a query to test, the game then only checks the freshness variable (line 44). These are only a conceptual changes, hence

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1] \ .$$

GAMES G_0-G_4

00 $(g_0, g_1) \xleftarrow{\$} \mathcal{G}^2$
01 $(x_0, x_1) \coloneqq (g_0 \star \tilde{x}, g_1 \star \tilde{x})$
02 $(\mathcal{C}, T) \coloneqq (\varnothing, \varnothing)$
03 $\mathbf{bad}_{\text{coll}} \coloneqq \mathbf{false}$
04 $\beta \xleftarrow{\$} \{0,1\}$
05 **for** $(\mathsf{U}, \mathsf{S}) \in \mathcal{U} \times \mathcal{S}$
06 $\quad \mathsf{pw}_{\mathsf{US}} \xleftarrow{\$} \mathcal{PW}$
07 $\beta' \leftarrow \mathcal{A}^{\mathrm{O}}(x_0, x_1)$
08 **return** $[\![\beta = \beta']\!]$

EXECUTE$(\mathsf{U}, t_0, \mathsf{S}, t_1)$
09 **if** $\pi_{\mathsf{U}}^{t_0} \neq \bot$ **or** $\pi_{\mathsf{S}}^{t_1} \neq \bot$
10 $\quad$ **return** $\bot$
11 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$ // G_0-G_3
12 $u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
13 $\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
14 $s \coloneqq (s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
15 $\hat{s} \coloneqq (\hat{s}_1, ..., \hat{s}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
16 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$ // G_0-G_3
17 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star x_{b_1}, ..., \hat{u}_\ell \star x_{b_\ell})$ // G_0-G_3
18 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$ // G_0-G_3
19 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star x_{b_1}, ..., \hat{s}_\ell \star x_{b_\ell})$ // G_0-G_3
20 **for** $i \in [\ell]$: // G_0-G_3
21 $\quad z_i \coloneqq (z_{i,1}, z_{i,2}, z_{i,3}) \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$ // G_0-G_3
22 $z \coloneqq (z_1, \ldots, z_\ell)$ // G_0-G_3
23 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star \tilde{x}, ..., u_\ell \star \tilde{x})$ // G_4
24 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star \tilde{x}, ..., \hat{u}_\ell \star \tilde{x})$ // G_4
25 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star \tilde{x}, ..., s_\ell \star \tilde{x})$ // G_4
26 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star \tilde{x}, ..., \hat{s}_\ell \star \tilde{x})$ // G_4
27 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ // G_1-G_4
28 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_4
29 $\quad$ **return** $\bot$ // G_1-G_4
30 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$ // G_0-G_2
31 $K \xleftarrow{\$} \mathcal{K}$ // G_3-G_4
32 $\pi_{\mathsf{U}}^{t_0} \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
33 $\pi_{\mathsf{S}}^{t_1} \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
34 $(\pi_{\mathsf{U}}^{t_0}.\mathsf{fr}, \pi_{\mathsf{S}}^{t_1}.\mathsf{fr}) \coloneqq (\mathbf{true}, \mathbf{true})$ // G_2-G_4
35 **return** $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

REVEAL(P, t)
36 **if** $\pi_{\mathsf{P}}^t.\mathsf{acc} \neq \mathbf{true}$ **or** $\pi_{\mathsf{P}}^t.\mathsf{test} = \mathbf{true}$
37 $\quad$ **return** $\bot$
38 **if** $\exists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
$\quad$ **and** $\pi_{\mathsf{P}'}^{t'}.\mathsf{test} = \mathbf{true}$
39 $\quad$ **return** $\bot$
40 $\forall (\mathsf{P}', t')$ s.t. $\pi_{\mathsf{P}'}^{t'}.\mathsf{tr} = \pi_{\mathsf{P}}^t.\mathsf{tr}$ // G_2-G_4
41 $\quad \pi_{\mathsf{P}'}^{t'}.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
42 **return** $\pi_{\mathsf{P}}^t.K$

TEST(P, t)
43 **if** $\mathsf{Fresh}(\pi_{\mathsf{P}}^t) = \mathbf{false}$ **return** $\bot$ // G_0-G_1
44 **if** $\pi_{\mathsf{P}}^t.\mathsf{fr} = \mathbf{false}$ **return** $\bot$ // G_2-G_4
45 $K_0^* \coloneqq \text{REVEAL}(\mathsf{P}, t)$
46 **if** $K_0^* = \bot$ **return** $\bot$
47 $K_1^* \xleftarrow{\$} \mathcal{K}$
48 $\pi_{\mathsf{P}}^t.\mathsf{test} \coloneqq \mathbf{true}$
49 **return** K_β^*

$\mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z)$
50 **if** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z] = K \neq \bot$
51 $\quad$ **return** K
52 $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, Z] \xleftarrow{\$} \mathcal{K}$
53 **return** $T[\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}, z]$

SENDINIT$(\mathsf{U}, t, \mathsf{S})$
54 **if** $\pi_{\mathsf{U}}^t \neq \bot$ **return** $\bot$
55 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
56 $u \coloneqq (u_1, ..., u_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
57 $\hat{u} \coloneqq (\hat{u}_1, ..., \hat{u}_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
58 $x^{\mathsf{U}} \coloneqq (x_1^{\mathsf{U}}, ..., x_\ell^{\mathsf{U}}) \coloneqq (u_1 \star x_{b_1}, ..., u_\ell \star x_{b_\ell})$
59 $\hat{x}^{\mathsf{U}} \coloneqq (\hat{x}_1^{\mathsf{U}}, ..., \hat{x}_\ell^{\mathsf{U}}) \coloneqq (\hat{u}_1 \star x_{b_1}, ..., \hat{u}_\ell \star x_{b_\ell})$
60 $\pi_{\mathsf{U}}^t \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
61 $\pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
62 **return** $(\mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$

SENDRESP$(\mathsf{S}, t, \mathsf{U}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}})$
63 **if** $\pi_{\mathsf{S}}^t \neq \bot$ **return** $\bot$
64 $(b_1, ..., b_\ell) \coloneqq \mathsf{pw}_{\mathsf{US}}$
65 $(s_1, ..., s_\ell) \xleftarrow{\$} \mathcal{G}^\ell$
66 $x^{\mathsf{S}} \coloneqq (x_1^{\mathsf{S}}, ..., x_\ell^{\mathsf{S}}) \coloneqq (s_1 \star x_{b_1}, ..., s_\ell \star x_{b_\ell})$
67 $\hat{x}^{\mathsf{S}} \coloneqq (\hat{x}_1^{\mathsf{S}}, ..., \hat{x}_\ell^{\mathsf{S}}) \coloneqq (\hat{s}_1 \star x_{b_1}, ..., \hat{s}_\ell \star x_{b_\ell})$
68 **if** $\exists \mathsf{P} \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ // G_1-G_4
69 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_4
70 $\quad$ **return** $\bot$ // G_1-G_4
71 **if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ // G_2-G_4
72 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_2-G_4
73 **else** // G_2-G_4
74 $\quad \pi_{\mathsf{S}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
75 **for** $i \in [\ell]$:
76 $\quad z_i \coloneqq (z_{i,1}, z_{i,2}, z_{i,3}) \coloneqq (s_i \star x_i^{\mathsf{U}}, s \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \star x_i^{\mathsf{U}})$
77 $z \coloneqq (z_1, ..., z_\ell)$
78 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
79 $\pi_{\mathsf{S}}^t \coloneqq ((s, \hat{s}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
80 **return** $(\mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$

SENDTERMINIT$(\mathsf{U}, t, \mathsf{S}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
81 **if** $\pi_{\mathsf{U}}^t \neq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, \bot, \bot), \bot, \bot)$
82 $\quad$ **return** $\bot$
83 **if** $\exists \mathsf{P} \in \mathcal{U}, t'$ s.t. $\pi_{\mathsf{P}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$ // G_1-G_4
84 $\quad \mathbf{bad}_{\text{coll}} \coloneqq \mathbf{true}$ // G_1-G_4
85 $\quad$ **return** $\bot$ // G_1-G_4
86 **if** $\exists t'$ s.t. $\pi_{\mathsf{S}}^{t'}.\mathsf{tr} = (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}})$
$\quad$ **and** $\pi_{\mathsf{S}}^{t'}.\mathsf{fr} = \mathbf{true}$ // G_2-G_4
87 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_2-G_4
88 **else if** $(\mathsf{U}, \mathsf{S}) \notin \mathcal{C}$ // G_2-G_4
89 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{true}$ // G_2-G_4
90 **else** // G_2-G_4
91 $\quad \pi_{\mathsf{U}}^t.\mathsf{fr} \coloneqq \mathbf{false}$ // G_2-G_4
92 **for** $i \in [\ell]$:
93 $\quad z_i \coloneqq (z_{i,1}, z_{i,2}, z_{i,3}) \coloneqq (u_i \star x_i^{\mathsf{S}}, \hat{u}_i \star x_i^{\mathsf{S}}, u_i \star \hat{x}_i^{\mathsf{S}})$
94 $z \coloneqq (z_1, ..., z_\ell)$
95 $K \coloneqq \mathsf{H}(\mathsf{U}, \mathsf{S}, x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}, \mathsf{pw}_{\mathsf{US}}, z)$
96 $\pi_{\mathsf{U}}^t \coloneqq ((u, \hat{u}), (\mathsf{U}, \mathsf{S}, x^{\mathsf{U}} x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}), K, \mathbf{true})$
97 **return true**

CORRUPT(U, S)
98 **if** $(\mathsf{U}, \mathsf{S}) \in \mathcal{C}$ **return** $\bot$
99 **for** $\mathsf{P} \in \{\mathsf{U}, \mathsf{S}\}$
100 $\quad$ **if** $\exists t$ s.t. $\pi_{\mathsf{P}}^t.\mathsf{test} = \mathbf{true}$
$\quad$ **and** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$
101 $\quad\quad$ **return** $\bot$
102 $\quad \forall \pi_{\mathsf{P}}^t$: **if** $\nexists \mathsf{P}' \in \mathcal{U} \cup \mathcal{S}, t'$ s.t. $\mathsf{Partner}(\pi_{\mathsf{P}}^t, \pi_{\mathsf{P}'}^{t'}) = 1$ // G_2-G_4
103 $\quad\quad \pi_{\mathsf{P}}^t.\mathsf{fr} = \mathbf{false}$ // G_2-G_4
104 $\mathcal{C} \coloneqq \mathcal{C} \cup \{(\mathsf{U}, \mathsf{S})\}$
105 **return** $\mathsf{pw}_{\mathsf{US}}$

Fig. 5. Games G_0-G_4 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} \coloneqq \{\text{EXECUTE}, \text{SENDINIT}, \text{SENDRESP}, \text{SENDTERMINIT}, \text{REVEAL}, \text{CORRUPT}, \text{TEST}, \mathsf{H}\}$.

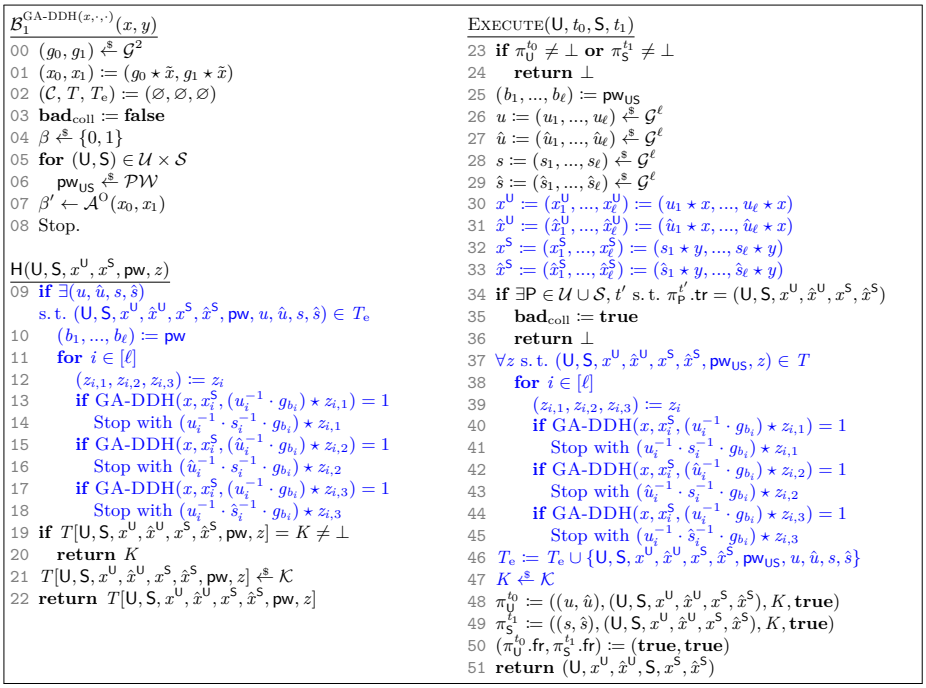

Fig. 6. Adversary $\mathcal{B}_1$ against GA-StCDH for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTermInit}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{H}\}$. Oracles SendInit, SendResp, SendTermInit, Reveal, Corrupt and Test are defined as in G_2. Lines written in blue show how $\mathcal{B}_1$ simulates the game.

Game G_3. In game G_3, we choose random keys for instances queried to Execute. We construct adversary $\mathcal{B}_1$ against GA-StCDH in Figure 6 and show that

$$|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) \ .$$

Adversary $\mathcal{B}_1$ inputs a GA-StCDH challenge $(x, y) = (g \star \tilde{x}, h \star \tilde{x})$ and has access to a decision oracle GA-DDH$(x, \cdot, \cdot)$. First, it generates the crs elements (x_0, x_1) as in game G_3 and then runs adversary $\mathcal{A}$. Queries to Execute are simulated as follows: It chooses random group elements $u_i, \hat{u}_i$ and $s_i, \hat{s}_i$ for user and server instances and $i \in [\ell]$, but instead of using (x_0, x_1) to compute the set elements, $\mathcal{B}_1$ uses x for the user instance and y for the server instance, independent of the password bits b_i (lines 30 to 33). We can rewrite this as

$$x_i^{\mathsf{U}} = u_i \star x = (u_i \cdot g) \star \tilde{x} = (u_i \cdot g \cdot g_{b_i} \cdot g_{b_i}^{-1}) \star \tilde{x} = \underbrace{(u_i \cdot g \cdot g_{b_i}^{-1})}_{u_i'} \star x_{b_i} \ ,$$

where u_i' is the group element that the user actually needs in order to compute the session key. In the same way, $\hat{u}_i' = \hat{u}_i \cdot g \cdot g_{b_i}^{-1}$, $s_i' = s_i \cdot h \cdot g_{b_i}^{-1}$ and $\hat{s}_i' = \hat{s}_i \cdot h \cdot g_{b_i}^{-1}$.

Note that $z_i = (z_{i,1}, z_{i,2}, z_{i,3})$ is implicitly set to

$$\begin{aligned} z_{i,1} &= (u_i' \cdot s_i') \star x_{b_i} = u_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ , \\ z_{i,2} &= (\hat{u}_i' \cdot s_i') \star x_{b_i} = \hat{u}_i \cdot g \cdot s_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ , \\ z_{i,3} &= (u_i' \cdot \hat{s}_i') \star x_{b_i} = u_i \cdot g \cdot \hat{s}_i \cdot h \cdot g_{b_i}^{-1} \star \tilde{x} \ . \end{aligned}$$

Before choosing a random session key, we check if there has been a query to the random oracle H that matches the session key (line 37-45). We iterate over the entries in T, where U, S, x^{U}, $\hat{x}^{\mathsf{U}}$, x^{S}, $\hat{x}^{\mathsf{S}}$ and $\mathsf{pw}_{\mathsf{US}}$ match, and check if one of the entries in z is correct. Note that we can use the following equivalences:

$$\begin{aligned} \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = z_{i,1} \quad &\Leftrightarrow \quad \mathsf{GA\text{-}CDH}(x, x_i^{\mathsf{S}}) = (u_i^{-1} \cdot g_{b_i}) \star z_{i,1}, \\ \mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = z_{i,2} \quad &\Leftrightarrow \quad \mathsf{GA\text{-}CDH}(x, x_i^{\mathsf{S}}) = (\hat{u}_i^{-1} \cdot g_{b_i}) \star z_{i,2}, \\ \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, \hat{x}_i^{\mathsf{S}}) = z_{i,3} \quad &\Leftrightarrow \quad \mathsf{GA\text{-}CDH}(x, \hat{x}_i^{\mathsf{S}}) = (u_i^{-1} \cdot g_{b_i}) \star z_{i,3}, \end{aligned}$$

which allows us to use the restricted decision oracle $\mathrm{GA\text{-}DDH}(x, \cdot, \cdot)$. If one of $z_{i,1}, z_{i,2}, z_{i,3}$ is correct, $\mathcal{B}_1$ aborts and outputs the solution $(g \cdot h) \star \tilde{x}$ which is respectively given by $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_{i,1}$, $(\hat{u}_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_{i,2}$ or $(u_i^{-1} \cdot \hat{s}_i^{-1} \cdot g_{b_i}) \star z_{i,3}$.

Otherwise, we store the values $u_i, \hat{u}_i$ and $s_i, \hat{s}_i$ in list T_{e} together with the trace and the password (line 46) and choose a session key uniformly at random. We need list T_{e} to identify relevant queries to H. In particular, if the trace and password appear in a query, we retrieve the values $u_i, \hat{u}_i$ and $s_i, \hat{s}_i$ to check whether the provided z_i are correct. We do this in the same way as described above using the decision oracle (lines 09-18). If the oracle returns 1 for any $z_{i,j}$, $\mathcal{B}_1$ aborts and outputs the solution for $(g \cdot h) \star \tilde{x}$ which is respectively given by $(u_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_{i,1}$, $(\hat{u}_i^{-1} \cdot s_i^{-1} \cdot g_{b_i}) \star z_{i,2}$ or $(u_i^{-1} \cdot \hat{s}_i^{-1} \cdot g_{b_i}) \star z_{i,3}$.

Game G_4. In game G_4, we remove the password from execute queries. In particular, we do not compute $x^{\mathsf{U}}, \hat{x}^{\mathsf{U}}, x^{\mathsf{S}}, \hat{x}^{\mathsf{S}}$ to the basis x_{b_i}, but simply use $\tilde{x}$. Note that the values have the same distribution as in the previous game. Also, the group elements $u, \hat{u}, s$ and $\hat{s}$ are not used to to derive the key. Hence, this change is not observable by $\mathcal{A}$ and

$$\Pr[\mathsf{G}_4 \Rightarrow 1] = \Pr[\mathsf{G}_3 \Rightarrow 1] \ .$$

Game G_5. G_5 is given in Figure 7. In this game we want to replace the session keys by random for all fresh instances in oracles SendResp and SendTermInit (lines 62, 83). Therefore, we introduce an additional independent random oracle T_{s} which maps only the trace of an instance to a key (lines 63, 84). We keep partner instances consistent, i.e., in case the adversary queries SendTermInit for a user instance and there exists a fresh partner instance, then we retrieve the corresponding key from T_{s} and also assign it to this instance (line 78). For all instances that are not fresh, we simply compute the correct key using random oracle H (lines 66-69, 87-90). If a session is fresh and there is an inconsistency between T and T_{s}, we raise flag **bad**. This happens in the following cases:

- a server instance is about to compute the session key, the password was not corrupted, but there already exists an entry in T with the correct password and z (lines 60-61).
- a user instance is about to compute the session key, there exists no partner instance and the password was not corrupted, but there already exists an entry in T with the correct password and z (lines 81-82).
- the random oracle is queried on some trace that appears in T_{s} together with the correct password and z (lines 36-47). At this point, we also check if the password was corrupted in the meantime and if this is the case and the adversary issues the correct query, we output the key stored in T_{s} (line 46) as this instance cannot be tested. This case corresponds to perfect forward secrecy which we cover in the full version of the paper [1, Appendix E].

When **bad** is not raised, there is no difference between G_4 and G_5. Hence,

$$|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \ .$$

Game G_6. G_6 is given in Figure 8. In this game we remove the password from send queries and generate passwords as late as possible, that is either when the adversary issues a corrupt query (line 21) or after it has stopped with output β' (line 07). In SendInit and SendResp we still choose group elements $u_i, \hat{u}_i, s_i$ and $\hat{s}_i$ uniformly at random, but now compute $x_i^{\mathsf{U}}, \hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}$ and $\hat{x}_i^{\mathsf{S}}$ using the origin element (lines 26, 27, 51 and 52). Thus, depending on which password is chosen afterwards, we implicitly set

$$x_i^{\mathsf{U}} = u_i \cdot \tilde{x} = (u_i \cdot g_0^{-1}) \star x_0 = (u_i \cdot g_1^{-1}) \star x_1$$

and analogously for $\hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}$ and $\hat{x}_i^{\mathsf{S}}$. For all instances that are not fresh, we have to compute the real session key using $z_i = (s_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}}, s_i \cdot g_{b_i}^{-1} \star \hat{x}_i^{\mathsf{U}}, \hat{s}_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{U}})$ (line 70) or $z_i = (u_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, \hat{u}_i \cdot g_{b_i}^{-1} \star x_i^{\mathsf{S}}, u_i \cdot g_{b_i}^{-1} \star \hat{x}_i^{\mathsf{S}})$ (line 97). Note that the password is already defined for these instances.

Recall that event **bad** in game G_5 is raised whenever there is an inconsistency in the random oracle queries and the keys of fresh instances. In this game, we split event **bad** into two different events:

- $\mathbf{bad}_{\mathsf{pw}}$ captures the event that there exists more than one valid entry in T for the same trace of a fresh instance, but different passwords.
- $\mathbf{bad}_{\mathrm{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen and is raised if there exists a valid entry in T for the trace of a fresh instance and the correct password, where the password was not corrupted when the query to H was made.

To identify the different events, we introduce a new set T_{bad}. For all fresh instances in SendResp and SendTermInit, we now iterate over all entries in T that contain the corresponding trace. We check if the given password and z are valid for this trace by computing the real values z' in the same way as for non-fresh instances. If $z = z'$, we add this entry to the set T_{bad} (lines 57-63, 84-90). We essentially do the same when the random oracle H is queried on a trace that

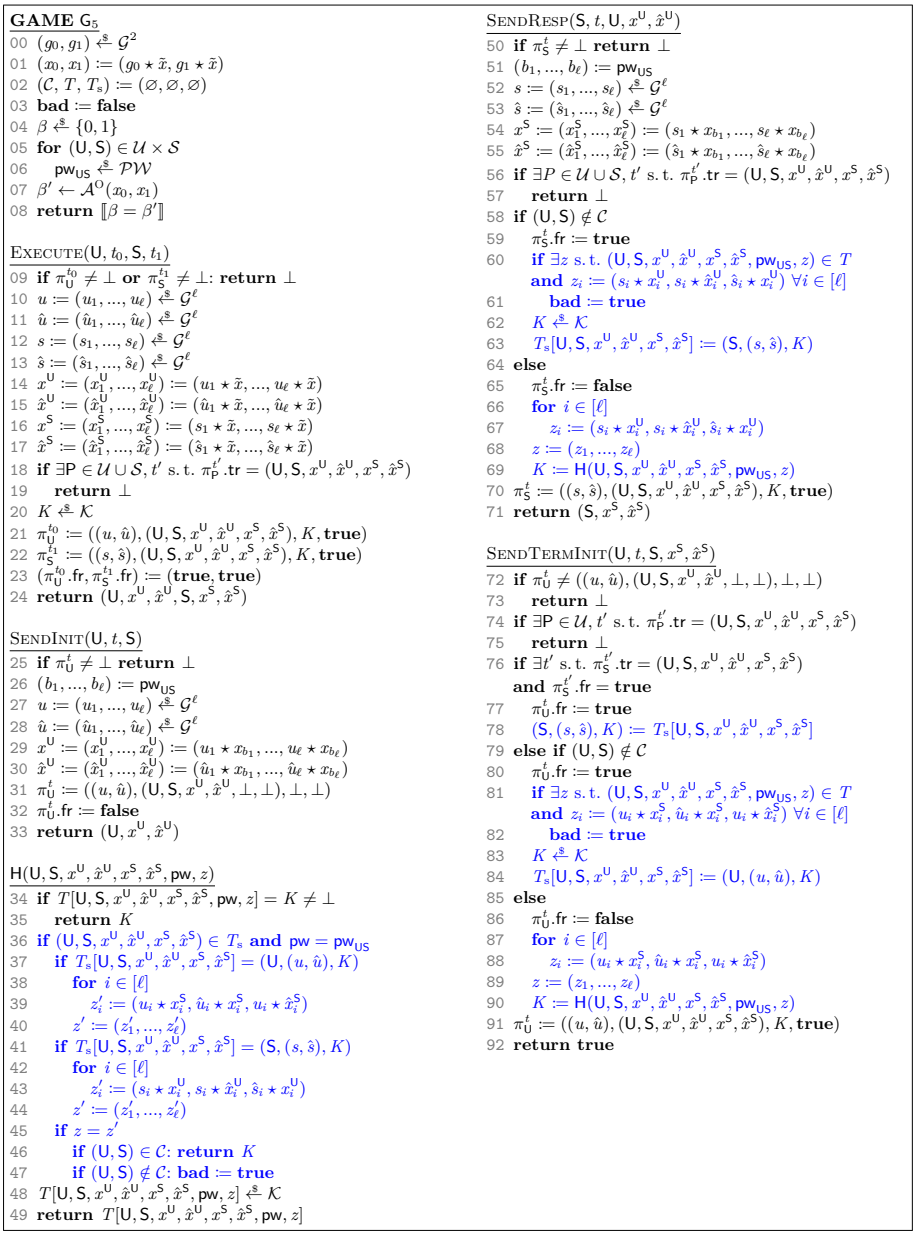

Fig. 7. Game G_5 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{$EXECUTE, SENDINIT, SENDRESP, SENDTERMINIT, REVEAL, CORRUPT, TEST, H$\}$. REVEAL, TEST and CORRUPT are defined as in Figure 5. Differences to G_4 are highlighted in blue.

appears in T_s. Here, the adversary specifies the password and we check if z is valid for that password using the $u_i, \hat{u}_i$ stored in T_s for user instances and $s_i, \hat{s}_i$ for server instances. If z is valid and the instance is still fresh, we add the query

Fig. 8. Game G_6 for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\mathrm{O} := \{\textsc{Execute}, \textsc{SendInit}, \textsc{SendResp}, \textsc{SendTermInit}, \textsc{Reveal}, \textsc{Corrupt}, \textsc{Test}, \mathsf{H}\}$. Oracles REVEAL and TEST are defined as in game G_4 in Figure 5. Oracle EXECUTE is defined as in Figure 7. Differences to G_5 are highlighted in blue.

to T_{bad} (lines 33-45). In case the password was corrupted in the meantime, we output the key stored in T_s as introduced in the previous game.

After the adversary terminates, we check T_{bad} whether event $\mathbf{bad}_{\mathsf{pw}}$ (line 09) or event $\mathbf{bad}_{\text{guess}}$ (line 12) occurred. We will bound these events below. First note that whenever **bad** is raised in G_5, then either flag $\mathbf{bad}_{\text{guess}}$ or $\mathbf{bad}_{\mathsf{pw}}$ is raised in G_6, thus

$$\Pr[\mathsf{G}_5 \Rightarrow \mathbf{bad}] \leq \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] + \Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] \ .$$

Finally, we bound the probabilities of the two events. We start with $\mathbf{bad}_{\mathsf{pw}}$. In Figure 9, we construct adversary $\mathcal{B}_2$ against DSim-GA-StCDH that simulates G_6.

We show that when $\mathbf{bad}_{\mathsf{pw}}$ occurs, then $\mathcal{B}_2$ can solve DSim-GA-StCDH. Hence,

$$\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\mathsf{pw}}] \leq \mathsf{Adv}_{\mathsf{EGA}}^{\mathsf{DSim\text{-}GA\text{-}StCDH}}(\mathcal{B}_2) \ .$$

Adversary $\mathcal{B}_2$ inputs (x_0, x_1, w_0, w_1), where $x_0 = g_0 \star \tilde{x}$, $x_1 = g_1 \star \tilde{x}$, $w_0 = h_0 \star \tilde{x}$ and $w_1 = h_1 \star \tilde{x}$ for group elements $g_0, g_1, h_0, h_1 \in \mathcal{G}$ chosen uniformly at random. Adversary $\mathcal{B}_2$ also has access to decision oracles $\text{GA-DDH}_{x_j}(w_i, \cdot, \cdot)$ for $(i, j) \in \{0,1\}^2$. It runs adversary $\mathcal{A}$ on (x_0, x_1). Queries to SendInit are simulated as follows: $\mathcal{B}_2$ chooses group elements u_i and $\hat{u}_i$ uniformly at random and sets

$$x_i^{\mathsf{U}} = u_i \star w_0 = (u_i \cdot h_0 \cdot g_0^{-1}) \star x_0 = (u_i \cdot h_0 \cdot g_1^{-1}) \star x_1 \ ,$$
$$\hat{x}_i^{\mathsf{U}} = \hat{u}_i \star w_1 = (\hat{u}_i \cdot h_1 \cdot g_0^{-1}) \star x_0 = (\hat{u}_i \cdot h_1 \cdot g_1^{-1}) \star x_1 \ .$$

The simulation of x_i^{S} and $\hat{x}_i^{\mathsf{S}}$ in SendResp is done in the same way, choosing random s_i and $\hat{s}_i$. In case the server instance is fresh, we must check if there already exists an entry in T that causes an inconsistency. As in G_6, we iterate over all pw, z, in T that contain the trace of this instance. In particular, we must check whether

$$z_{i,1} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, x_i^{\mathsf{U}}) = s_i^{-1} \star z_{i,1} \ ,$$
$$z_{i,2} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(\hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_0, \hat{x}_i^{\mathsf{U}}) = s_i^{-1} \star z_{i,2} \ ,$$
$$z_{i,3} = \mathsf{GA\text{-}CDH}_{x_{b_i}}(x_i^{\mathsf{U}}, \hat{x}_i^{\mathsf{S}}) \quad \Leftrightarrow \quad \mathsf{GA\text{-}CDH}_{x_{b_i}}(w_1, x_i^{\mathsf{U}}) = \hat{s}_i^{-1} \star z_{i,3} \ ,$$

which can be done with the decision oracles $\text{GA-DDH}_{x_{b_i}}(w_j, \cdot, \cdot)$. If all z_i are valid, then we add this entry to T_{bad} (lines 56-59).

If the instance is not fresh, then we have to compute the correct key. We check list T for a valid entry z as explained above and if it exists, we assign this value to the session key (line 66). Otherwise, we choose a random key and add a special entry to T, which instead of z contains the secret group elements s_i and $\hat{s}_i$ (line 69) so that we can patch the random oracle later. SendTermInit is simulated analogously, using the secret group elements u_i and $\hat{u}_i$.

Now we look at the random oracle queries. If the trace is contained in set T_{s} which means the corresponding instance was fresh when the send query was issued, we check if z is valid using the GA-DDH oracle. We do this as described above, depending on whether it is a user or a server instance (lines 25, 31). In case z is valid, we first check if the instance is still fresh (i.e., the password was not corrupted in the meantime) and if this is the case, we add the query to T_{bad}

Fig. 9. Adversary $\mathcal{B}_2$ against DSim-GA-StCDH for the proof of Theorem 1. $\mathcal{A}$ has access to oracles $\text{O} \coloneqq \{\text{Execute}, \text{SendInit}, \text{SendResp}, \text{SendTermInit}, \text{Reveal}, \text{Corrupt}, \text{Test}, \mathsf{H}\}$. Oracles Execute, Reveal, Corrupt and Test are defined as in G_6. Lines written in blue show how $\mathcal{B}_2$ simulates the game.

(lines 28, 34). Otherwise, if the password was corrupted and is specified in the query, we return the session key stored in T_s (lines 30, 36).

Next, we check if the query matches a special entry in T that was added in SendResp or SendTermInit for a non-fresh instance, which means we have to output the same key that was chosen before. Again, we can use the GA-DDH oracle and differentiate between user and server instances (lines 37-44).

After $\mathcal{A}$ terminates with output β', $\mathcal{B}_2$ chooses the passwords which have not been generated in a Corrupt query yet. If $\mathbf{bad}_{\mathsf{pw}}$ occurred (lines 05-13), then there must be two entries in T_{bad} for the same trace and different passwords $\mathsf{pw} \neq \mathsf{pw}'$ along with values z and z'. Let i be the first index where the two passwords differ, i.e., $b_i \neq b'_i$. Without loss of generality assume that $b_i = 0$ and $b'_i = 1$, otherwise swap pw, z and pw', z'. If the entries in T_{bad} are those of a user instance, we retrieve the secret group elements $u, \hat{u}_i$ from T_{s}.

Recall that the DSim-GA-StCDH problem requires to compute $y_0 = \mathsf{GA\text{-}CDH}_{x_0}(w_0, y)$, $y_1 = \mathsf{GA\text{-}CDH}_{x_0}(w_1, y)$, $y_2 = \mathsf{GA\text{-}CDH}_{x_1}(w_0, y)$ and $y_3 = \mathsf{GA\text{-}CDH}_{x_1}(w_1, y)$, where y can be chosen by the adversary. $\mathcal{B}_2$ sets $y = x_i^{\mathsf{S}}$, and outputs y and

$$
\begin{aligned}
y_0 &= u_i^{-1} \star z_{i,1} = \mathsf{GA\text{-}CDH}_{x_0}(u_i^{-1} \star x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_0, x_i^{\mathsf{S}}) \ , \\
y_1 &= \hat{u}_i^{-1} \star z_{i,2} = \mathsf{GA\text{-}CDH}_{x_0}(\hat{u}_i^{-1} \star \hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_0}(w_1, x_i^{\mathsf{S}}) \ , \\
y_2 &= u_i^{-1} \star z'_{i,1} = \mathsf{GA\text{-}CDH}_{x_1}(u_i^{-1} \star x_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_0, x_i^{\mathsf{S}}) \ , \\
y_3 &= \hat{u}_i^{-1} \star z'_{i,2} = \mathsf{GA\text{-}CDH}_{x_1}(\hat{u}_i^{-1} \star \hat{x}_i^{\mathsf{U}}, x_i^{\mathsf{S}}) = \mathsf{GA\text{-}CDH}_{x_1}(w_1, x_i^{\mathsf{S}}) \ .
\end{aligned}
$$

If the instance is a server instance, $\mathcal{B}_2$ outputs $(y, y_0, y_1, y_2, y_3) = (x_i^{\mathsf{U}}, s_i^{-1} \star z_{i,1}, \hat{s}_i^{-1} \star z_{i,3}, s_i^{-1} \star z'_{i,1}, \hat{s}_i^{-1} \star z'_{i,3})$. This concludes the analysis of $\mathbf{bad}_{\mathsf{pw}}$.

Next, we analyze event $\mathbf{bad}_{\text{guess}}$. Recall that $\mathbf{bad}_{\text{guess}}$ happens only if $\mathbf{bad}_{\mathsf{pw}}$ does not happen. Hence, for each instance there is at most one entry in T_{bad} and the size of T_{bad} is at most q_s. As all entries were added before the corresponding password was sampled, the probability is bounded by

$$
\Pr[\mathsf{G}_6 \Rightarrow \mathbf{bad}_{\text{guess}}] \leq \frac{q_s}{|\mathcal{PW}|} \ .
$$

Finally, note that if none of the bad events happens in G_6, all session keys output by Test are uniformly random and the adversary can only guess β. Hence, $\Pr[\mathsf{G}_6 \Rightarrow 1] = \frac{1}{2}$. Collecting the probabilities and using Equation 1 yields the bound in Theorem 1. □

7 Com-GA-PAKE$_\ell$: Three-Round PAKE from Group Actions

In this section we present a second modification of GA-PAKE$_\ell$, which can be securely instantiated with an EGAT. The protocol Com-GA-PAKE$_\ell$ extends GA-PAKE$_\ell$ by a commitment that has to be sent before sending the actual messages. This ensures that the server cannot choose the set elements depending on the message it receives from the user which was the crucial step in the attack against

$\mathsf{GA\text{-}PAKE}_\ell$. In the second round, the user sends its message to the server and only after receiving that message, the server sends its message to the user. The protocol is sketched in Figure 10 and its security is established in Theorem 2. While this protocol adds two rounds to the original protocol, the total computational cost is lower than for $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$.

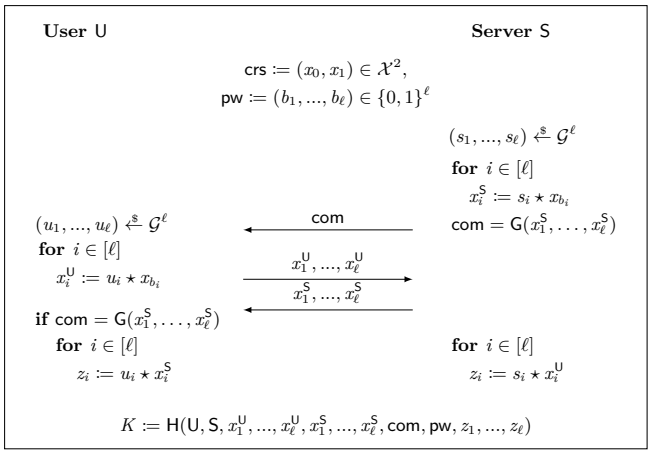

Fig. 10. PAKE protocol $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ from group actions.

Theorem 2 (Security of $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$). *For any adversary $\mathcal{A}$ against $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ that issues at most q_e execute queries, q_s send queries and at most q_G and q_H queries to random oracles G and H, there exist an adversary $\mathcal{B}_1$ against $\mathsf{GA\text{-}StCDH}$ and an adversary $\mathcal{B}_2$ against $\mathsf{GA\text{-}GapCDH}$ such that*

$$\mathsf{Adv}_{\mathsf{Com\text{-}GA\text{-}PAKE}_\ell}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{GA\text{-}StCDH}}_{\mathsf{EGAT}}(\mathcal{B}_1) + q_s \ell \cdot \sqrt{\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{EGAT}}(\mathcal{B}_2)} + \frac{(q_s + q_e)^2}{|\mathcal{G}|^\ell} + \frac{q_\mathsf{G} q_s}{|\mathcal{G}|^\ell} + \frac{2 \cdot (q_\mathsf{G} + q_s + q_e)^2}{2^\lambda} + \frac{q_s}{|\mathcal{PW}|} \ ,$$

where λ is the output length of G in bits.

The proof is similar to the one of Theorem 1 so we will only sketch it here. The full proof is given in the long version of the paper [1, Appendix E].

Proof (Sketch). After ensuring that all traces are unique, we need to deal with the commitment and in particular collisions. First, we require that there are never two inputs to the random oracle G that return the same commitment. This is to ensure that the adversary cannot open a commitment to a different value, which might depend on previous messages.

Second, we need to ensure that after the adversary has seen a commitment, it does not query G on the input, which is the hiding property of the commitment.

What we actually do here is that we choose a random commitment in the first round. Only later we choose the input and patch the random oracle accordingly.

Now we can replace the session keys of instances which are used in execute queries. Here, the freshness condition allows the adversary to corrupt the password. However, as both x^{S} and x^{U} are generated by the experiment, the only chance to notice this change is to solve the GA-StCDH problem, where the decision oracle is required to simulate instances correctly.

In order to replace the session keys of fresh instances which are used in send queries, we make the key independent of the password. The session key of a fresh instance is now defined by the trace of that instance. The only issue that may arise here is an inconsistency between the session key that is derived using the trace and the session key that is derived using the random oracle H. Whenever such an inconsistency occurs, we differentiate between two cases:

- There exists more than one valid entry in T_{H} for the same trace of a fresh instance, but different passwords.
- There exists a valid entry in T_{H} for the trace of a fresh instance and the correct password, where the password was not corrupted when the query to H was made.

Finally, we bound the probabilities of the two cases. Similar to Theorem 1, we will define a new computational problem that reflects exactly the interaction in the protocol. We show that this problem is implied by GA-GapCDH using the reset lemma. The general idea is that the adversary can always compute the session key for one password guess, but not for a second one. After excluding this, we choose the actual password, which is possible because session keys are computed independently of the password. Thus, looking at one fixed instance, the probability that the adversary guessed the password correctly is $1/|\mathcal{PW}|$. □

8 Variants of the PAKE Protocols

Both protocols X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ require that the user and the server generate multiple random group elements and evaluate their action on certain set elements. In this section we present two optimizations that allow us to reduce the number of random group elements and the number of evaluations.

8.1 Increasing the Number of Public Parameters

In X-GA-PAKE$_\ell$ and Com-GA-PAKE$_\ell$ the common reference string is set to $\mathsf{crs} := (x_0, x_1) \in \mathcal{X}^2$. Increasing the number of public parameters allows to reduce the number of group action evaluations in the execution of the protocol. The idea is similar to the optimizations deployed to speed up the CSIDH-based signatures schemes SeaSign [14] and CSI-FiSh [9]. We refer to Table 1 in the introduction for an overview and example of the parameter choice.

We explain the changes on the basis of protocol X-GA-PAKE$_\ell$. The variant of Com-GA-PAKE$_\ell$ is similar and is provided in the full version of the paper

[1, Appendix E], together with a security analysis for both variants. For some positive integer N dividing ℓ, we set

$$\mathsf{crs} \coloneqq (x_0, \ldots, x_{2^N-1}) \in \mathcal{X}^{2^N} \quad \text{and} \quad \mathsf{pw} = (b_1, ..., b_{\ell/N}) \in \{0, ..., 2^N - 1\}^{\ell/N}.$$

Note that as before, the password is a bitstring of length ℓ, but it is divided into ℓ/N blocks of length N. In particular x_{b_i} refers to one of the 2^N different set elements in the crs. The general outline of the protocol does not change. The only difference is that in the first step both the server and the user only generate $2 \cdot \ell/N$ random group elements (instead of $2 \cdot \ell$). Hence they only need to perform $2 \cdot \ell/N$ group action evaluations in the first round and $3 \cdot \ell/N$ evaluations in the session key derivation. We write $\mathsf{X\text{-}GA\text{-}PAKE}_{\ell,N}$ for this variant of the protocol.

8.2 Using Twists in the Setup

Both $\mathsf{X\text{-}GA\text{-}PAKE}_\ell$ and $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell$ require that some trusted party generates two random set elements $\mathsf{crs} = (x_0, x_1)$. Here, we shortly discuss the setup where x_1 is replaced by the twist of x_0, i.e. $\mathsf{crs} \coloneqq (x_0, x_0^t)$.

This simplification is particularly helpful when applied to one of the variants from the previous subsection. These modified versions require to generate 2^N random set elements for the crs. Using twists it suffices to generate 2^{N-1} elements $(x_0, \ldots, x_{2^{N-1}-1}) \in \mathcal{X}^{2^{N-1}}$ and setting $x_{i+2^{N-1}} = x_i^t$ for each $i \in [0, 2^{N-1} - 1]$.

The security of $\mathsf{X\text{-}GA\text{-}PAKE}_\ell^{\mathsf{t}}$ and $\mathsf{Com\text{-}GA\text{-}PAKE}_\ell^{\mathsf{t}}$ (the twisted versions of the protocols) is discussed in the full version [1, Appendices D,E].

Acknowledgments. Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler and Doreen Riepel were supported by the DFG under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

1. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. Cryptology ePrint Archive, Report 2022/770 (2022), https://eprint.iacr.org/2022/770
2. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (Jan 2005). https://doi.org/10.1007/978-3-540-30580-4_6
3. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. Cryptology ePrint Archive, Report 2021/114 (2021), https://eprint.iacr.org/2021/114
4. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30574-3_14
5. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_14

6. Azarderakhsh, R., Jao, D., Koziel, B., LeGrow, J.T., Soukharev, V., Taraskin, O.: How not to create an isogeny-based PAKE. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 20, Part I. LNCS, vol. 12146, pp. 169–186. Springer, Heidelberg (Oct 2020). https://doi.org/10.1007/978-3-030-57808-4_9
7. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_11
8. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992). https://doi.org/10.1109/RISP.1992.213269
9. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_9
10. Canetti, R., Dachman-Soled, D., Vaikuntanathan, V., Wee, H.: Efficient password authenticated key exchange via oblivious transfer. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 449–466. Springer, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8_27
11. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_15
12. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006), https://eprint.iacr.org/2006/291
13. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_4
14. De Feo, L., Galbraith, S.D.: SeaSign: Compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 759–789. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17659-4_26
15. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Online-extractability in the quantum random-oracle model. Cryptology ePrint Archive, Report 2021/280 (2021), https://eprint.iacr.org/2021/280
16. Fujioka, A., Takashima, K., Yoneyama, K.: One-round authenticated group key exchange from isogenies. In: Steinfeld, R., Yuen, T.H. (eds.) ProvSec 2019. LNCS, vol. 11821, pp. 330–338. Springer, Heidelberg (Oct 2019). https://doi.org/10.1007/978-3-030-31919-9_20
17. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 524–543. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_33, https://eprint.iacr.org/2003/032.ps.gz
18. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. IACR TCHES **2019**(2), 1–48 (2019). https://doi.org/10.13154/tches.v2019.i2.1-48, https://tches.iacr.org/index.php/TCHES/article/view/7384
19. Hao, F., Ryan, P.: J-PAKE: Authenticated key exchange without PKI. Cryptology ePrint Archive, Report 2010/190 (2010), https://eprint.iacr.org/2010/190

20. Jablon, D.P.: Strong password-only authenticated key exchange. ACM SIGCOMM Computer Communication Review **26**(5), 5–26 (1996)
21. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011. pp. 19–34. Springer, Heidelberg (Nov / Dec 2011). https://doi.org/10.1007/978-3-642-25405-5_2
22. Kawashima, T., Takashima, K., Aikawa, Y., Takagi, T.: An efficient authenticated key exchange from random self-reducibility on CSIDH. In: Hong, D. (ed.) ICISC 20. LNCS, vol. 12593, pp. 58–84. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-68890-5_4
23. de Kock, B., Gjøsteen, K., Veroni, M.: Practical isogeny-based key-exchange with optimal tightness. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) Selected Areas in Cryptography. pp. 451–479. Springer International Publishing, Cham (2021)
24. Lai, Y.F., Galbraith, S.D., de Saint Guilhem, C.: Compact, efficient and UC-secure isogeny-based oblivious transfer. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 213–241. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_8
25. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (Aug 2008). https://doi.org/10.1007/978-3-540-85174-5_31
26. Pointcheval, D., Wang, G.: VTBPEKE: Verifier-based two-basis password exponential key exchange. In: Karri, R., Sinanoglu, O., Sadeghi, A.R., Yi, X. (eds.) ASIACCS 17. pp. 301–312. ACM Press (Apr 2017)
27. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006), https://eprint.iacr.org/2006/145
28. Soukharev, V., Hess, B.: PQDH: A quantum-safe replacement for Diffie-Hellman based on SIDH. Cryptology ePrint Archive, Report 2019/730 (2019), https://eprint.iacr.org/2019/730
29. Taraskin, O., Soukharev, V., Jao, D., LeGrow, J.: An isogeny-based password-authenticated key establishment protocol. Cryptology ePrint Archive, Report 2018/886 (2018), https://eprint.iacr.org/2018/886
30. Terada, S., Yoneyama, K.: Password-based authenticated key exchange from standard isogeny assumptions. In: Steinfeld, R., Yuen, T.H. (eds.) ProvSec 2019. LNCS, vol. 11821, pp. 41–56. Springer, Heidelberg (Oct 2019). https://doi.org/10.1007/978-3-030-31919-9_3
31. Yoneyama, K.: Post-quantum variants of iso/iec standards: Compact chosen ciphertext secure key encapsulation mechanism from isogeny. In: Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop. p. 13–21. SSR'19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338500.3360336