Multiple Group Action Dlogs with(out) Precomputation

Alexander May[⋆] and Massimo Ostuzzi[⋆⋆]

Ruhr-University Bochum, Bochum, Germany
alex.may@rub.de, massimo.ostuzzi@rub.de

Abstract. Let $\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$ be the action of a group $\mathcal{G}$ of size $N = |\mathcal{G}|$ on a set $\mathcal{X}$. Let $y = g \star x \in \mathcal{X}$ be a *group action dlog* instance, where our goal is to compute the unknown group element $g \in \mathcal{G}$ from the known set elements $x, y \in \mathcal{X}$. The Galbraith-Hess-Smart (GHS) collision finding algorithm solves the group action dlog in $N^{\frac{1}{2}}$ steps with polynomial memory.

We show that group action dlogs are suitable for precomputation attacks. More precisely, for any s, t our precomputation algorithm computes within st steps a hint of space complexity s, which allows to solve any group action dlog in an online phase within t steps. A typical instantiation is $s = t = N^{\frac{1}{3}}$, which gives precomputation time $N^{\frac{2}{3}}$ and space $N^{\frac{1}{3}}$, and online time only $N^{\frac{1}{3}}$.

Moreover, we show that solving multiple group action dlog instances $y_1, \ldots, y_m$ allows for speedups. Namely, our collision finding algorithm solves m group action dlogs in $\sqrt{m}N^{\frac{1}{2}}$ steps, instead of the straightforward $mN^{\frac{1}{2}}$ steps required for running m times GHS. Our multiple instance approach can be freely combined with our precomputations, allowing for a variety of tradeoffs.

Technically, our precomputation and multiple instance group action dlog attacks are adaptations of the techniques from the standard dlog setting in abelian groups. While such an adaptation seems natural, it is per se unclear which techniques transfer from the dlog to the more restricted group dlog setting, for which $\mathcal{X}$ does not offer a group structure.

Our algorithms have direct implications for all group action based cryptosystems, such as CSIDH and its variants. We provide experimental evidence that our techniques work well in the CSIDH setting.

Keywords: group actions · CSIDH · preprocessing · multi-instance dlogs · random walks.

⋆ Funded by DFG under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

⋆⋆ M.O. is part of the Quantum-Safe Internet (QSI) ITN which received funding from the European Union's Horizon-Europe programme as Marie Sklodowska-Curie Action (PROJECT 101072637 - HORIZON - MSCA-2021-DN-01)

1 Introduction

The invention of the discrete logarithm (dlog) based Diffie-Hellman (DH) key exchange in 1976 marks the birth of modern public key cryptography. DH is nowadays used ubiquitously in practice. As a consequence, the discovery of Shor's polynomial time quantum algorithm for computing discrete logarithms (dlogs) in any abelian group [Sho94] came as a shock to the cryptographic community, turning DH insecure in a quantum world.

While one would like to replace the group-based dlog problem by some quantum resistant problem, it is also desirable to retain the benefits of DH key exchange, such as small public keys, efficient computations, and compatibility with existing protocols.

The *group action dlog problem* evolved as a natural and elegant way to replace group-based dlogs, and group actions are the fruitful basis for a whole new research area called isogeny-based cryptography [BY91,Cou06,Sto10,JDF11] [ACC+17,DFKS18,CLM+18,CD20]. The hope is that group action dlogs preserve all benefits from group based dlogs, while providing more security, especially against quantum computers.

Indeed, the group action dlog problem is a *hidden shift problem* [CJS14], for which the best quantum algorithm by Kuperberg [Kup05] has subexponential complexity. Nowadays, it is still an open question whether concrete instantiations of Kuperberg's algorithm [Pei20,BS20] pose threats to current group action dlog parameters.

Moreover, the CSIDH key exchange protocol [CLM+18] provides an efficient instantiation of group actions that leads to a highly attractive, promising replacement of DH in a quantum world.

Adaptation from Dlog Algorithms. Before considering a widespread replacement of DH by some group action based scheme like CSIDH, it is crucial to understand to which extent dlog attacks transfer to the group action dlog setting.

Using Pollard's collision finding algorithm [Pol78], we can compute a single dlog in any abelian group of size N in $O(N^{\frac{1}{2}})$ steps. The counterpart for group actions dlogs of instance size N is the algorithm of Galbraith, Hess and Smart [GHS02], and for CSIDH the refined algorithm of Delfs and Galbraith [DG16], that both likewise require $O(N^{\frac{1}{2}})$ steps.

Other dlog algorithms do not transfer naturally. One example is again Shor's polynomial time dlog algorithms, for which its group action counterpart Kuperberg's algorithm requires subexponential time. A second example is the Silver-Pohlig-Hellman algorithm [PH78], that exploits smoothness of the group order, for which currently no counterpart is believed to exist in the group action dlog setting [CLM+18].

Therefore it is of great importance to understand which dlog algorithms can be transfered into the group action setting at all.

Dlog Precomputation and Multi-Instances. The discrete log setting has the nice feature that one can standardize groups that are believed to be especially efficient, and for which the dlog problem is considered hard. Examples are the current elliptic curve standards, like NIST P-256. These standardized groups provide an advantage over RSA based key exchange, for which users may generate insecure instances [HDWH12].

An analogous property holds for group action based schemes. For instance CSIDH-512 provides an efficient systemwide instantiation of a group action that all users are supposed to use securely. In fact, it seems that the group action setting is even more restrictive, in the sense that it is harder to find suitable instantiations that are both efficient and secure [CLM$^+$18].

The drawback of systemwide instantiations is that they are attractive targets for powerful adversaries. It appears plausible that a large scale adversary, such as a national state agency, has the capabilities to perform a heavy precomputation. For a cryptographic (group action) standard that is used by billions of devices, such a precomputation may run over several years to produce some hint. The hint in turn allows the large scale adversary in an online phase for significantly more efficient (group action) dlog computations. In the dlog setting, precomputations have been studied by [BL12,CGK18].

Moreover, large scale adversaries interested in mass surveillance of users not only desire to compute a single (group action) dlog, but aim to amortize their costs for recovering a plethora of cryptographic keys. In the dlog setting, it is known that m dlogs can be computed in time $\sqrt{m}N^{\frac{1}{2}}$ [KS01,FJM14,Yun15], instead of the naive $mN^{\frac{1}{2}}$ by applying Pollard's algorithm m times, thereby amortizing the attack costs.

Our main contribution is to transfer both the precomputation and the multi-instance attacks from the dlog to the group action dlog setting. To this end, let us work out in the following the similarities (and limitations) of both settings in a bit more detail.

Discrete Logarithms. Let us recap the discrete logarithm in a finite cyclic group H generated by $h \in H$, in which we denote the group operation as multiplication. We write $H = \langle h \rangle = \{h, h^2, \ldots, h^{\mathrm{ord}(h)}\}$, with $\mathrm{ord}(h) = |H|$. We denote the integers modulo $|H|$ by $\mathbb{Z}_{|H|} := \mathbb{Z}/|H|\mathbb{Z}$.

Let us consider the exponentiation map $\varphi_h : \mathbb{Z} \to H, v \mapsto h^v$. Notice that $\lambda = \ker(\varphi_h) = |H|\mathbb{Z}$ is a 1-dimensional lattice in $\mathbb{Z}$. Therefore, the following map is a bijection

$$f_h : \mathbb{Z}/\lambda \to H, \quad v \mapsto h^v.$$

Let $y = h^v = f_h(v)$. The *discrete logarithm (dlog) problem in H* is to invert f_h on y, namely to compute the unique $v = f_h^{-1}(y) \bmod \lambda$.

In *group action* terminology, for our discrete logarithm problem the group $\mathcal{G} := \mathbb{Z}/\lambda = \mathbb{Z}_{|H|}$ acts on the *group* H. The group structure of H itself is exploited in many algorithms, such as Shor's algorithm [Sho94] and Pollard's Rho algorithm [Pol78].

Pollard Rho uses the technique of *collision finding*. Let $y = h^v$ and define

$$f_{h,y} : \mathcal{G} \times \mathcal{G} \to H, \quad (x, z) \mapsto h^x y^z.$$

Moreover, suppose $(x,z) \neq (x',z')$ is a collision in $f_{h,y}$, namely $f_{h,y}(x,z) = f_{h,y}(x',z')$. We have

$$f_{h,y}(x,z) = h^x y^z = h^{x+vz},$$

and likewise

$$f_{h,y}(x',z') = h^{x'} y^{z'} = h^{x'+vz'}.$$

We conclude that the collision $(x,z),(x',z')$ directly yields the discrete logarithm as $v = \frac{x-x'}{z'-z} \bmod \lambda$, provided that $z'-z$ is invertible modulo λ. Notice that our reasoning relies on H's group structure.

Group Action Dlogs. Let us now introduce the *group action discrete logarithm* (GA-dlog) problem, and discuss its similarities and differences to the ordinary discrete logarithm problem in a group H.

Let $\mathcal{X}$ be a set, *without any group structure*. Let $x \in \mathcal{X}$ be a distinguished element, called the *origin*, that plays the role of a generator of $\mathcal{X}$. Namely, we let some finite abelian group $\mathcal{G}$ act on $\mathcal{X}$ with $\star$ via the origin x, such that $\{g \star x \mid g \in \mathcal{G}\} = \mathcal{X}$. Notice that our definition of $\mathcal{X}$ already implies $N = |\mathcal{G}| \geq |\mathcal{X}|$, but we will furthermore require that $|\mathcal{X}| = |\mathcal{G}|$.

Let us assume that $\mathbf{g} = \{g_1, \ldots, g_n\}$ is a finite set of generators for $\mathcal{G}$, denoted $\mathcal{G} = \langle \mathbf{g} \rangle = \langle g_1, \ldots, g_n \rangle$. Moreover, for any integer vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ we write $\mathbf{g}^{\mathbf{v}} = g_1^{v_1} \cdot \ldots \cdot g_n^{v_n}$.

Let us consider the exponentiation map $\varphi_{\mathbf{g}} : \mathbb{Z}^n \to \mathcal{G}, \mathbf{v} \mapsto \mathbf{g}^{\mathbf{v}}$. Notice that $\Lambda = \ker(\varphi_{\mathbf{g}})$ is an n-dimensional lattice in $\mathbb{Z}^n$. The following map is a bijection

$$f_{\mathbf{g},x} : \mathbb{Z}^n/\Lambda \to \mathcal{X}, \quad \mathbf{v} \mapsto \mathbf{g}^{\mathbf{v}} \star x. \tag{1}$$

Let $y = \mathbf{g}^{\mathbf{v}} \star x = f_{\mathbf{g},x}(\mathbf{v})$. Then the *GA-dlog problem in* $\mathcal{X}$ is to invert $f_{\mathbf{g},x}$ on y, namely to compute the unique $\mathbf{v} = f_{\mathbf{g},x}^{-1}(y) \bmod \Lambda$, that in turn represents the group element $\mathbf{g}^{\mathbf{v}} \in \mathcal{G}$.

Notice that the missing group structure of $\mathcal{X}$ prevents a straight-forward adaptation of Pollard's collision finding technique, as well as an adaptation of Shor's quantum dlog algorithm.

Our Contributions. Let $y = \mathbf{g}^{\mathbf{v}} \star x$ be a GA-dlog problem for a group $\mathcal{G} = \langle \mathbf{g} \rangle$ of size $N = |\mathcal{G}|$. We define the two functions

$$f_{\mathbf{g},x} : \mathbb{Z}^n/\Lambda \to \mathcal{X}, \quad \mathbf{v} \mapsto \mathbf{g}^{\mathbf{v}} \star x,$$
$$f_{\mathbf{g},y} : \mathbb{Z}^n/\Lambda \to \mathcal{X}, \quad \mathbf{v} \mapsto \mathbf{g}^{\mathbf{v}} \star y,$$

Let $(\mathbf{v}_1, \mathbf{v}_2)$ be a collision of $f_{\mathbf{g},x}, f_{\mathbf{g},y}$, namely $f_{\mathbf{g},x}(\mathbf{v}_1) = f_{\mathbf{g},y}(\mathbf{v}_2)$. Then we have

$$f_{\mathbf{g},x}(\mathbf{v}_1) = \mathbf{g}^{\mathbf{v}_1} \star x, \text{ and}$$
$$f_{\mathbf{g},y}(\mathbf{v}_2) = \mathbf{g}^{\mathbf{v}_2} \star y = \mathbf{g}^{\mathbf{v}_2} \star (\mathbf{g}^{\mathbf{v}} \star x) \overset{(*)}{=} (\mathbf{g}^{\mathbf{v}_2+\mathbf{v}}) \star x,$$

where $(*)$ holds by a group action property called *compatibility*. By injectivity of the map in Equation (1), we obtain the GA-dlog as $\mathbf{v} = \mathbf{v}_1 - \mathbf{v}_2 \bmod \Lambda$.

Precomputation. Observe that the function $f_{\mathbf{g},x}$, as opposed to $f_{\mathbf{g},y}$, solely depends on the group action defined via $(\mathbf{g}, x)$, but not on a GA-dlog instance y. Therefore, we call $f_{\mathbf{g},x}$ *instance-independent*, while $f_{\mathbf{g},y}$ is called *instance-dependent*. The instance-independence of $f_{\mathbf{g},x}$ allows us to precompute via $(\mathbf{g}, x)$ alone a hint for the group action. On obtaining a concrete ga-dlog instance y, one can then use the hint to compute more efficiently a collision via $f_{\mathbf{g},y}$.

More concretely, for $f_{\mathbf{g},x}$ we precompute s instance-independent random walks, each of them of length t. This requires time st and space s, by storing only their endpoints, which then serve as our hint. Let us look at a typical parameter choice $s = t = N^{\frac{1}{3}}$. Then precomputation requires time $st = N^{\frac{2}{3}}$ and space only $s = N^{\frac{1}{3}}$.

We show that our precomputation already touches roughly $st = N^{\frac{2}{3}}$ points in $\mathcal{X}$. Thus, we expect that any instance-dependent walk with $f_{\mathbf{g},y}$ of length $t = N^{\frac{1}{3}}$ collides with one of these points, thereby yielding a solution to the GA-dlog problem.

Multiple Instances. Let $y_1, \ldots, y_m$ be m GA-dlog instances. Solving all instances via the Galbraith-Hess-Smart algorithm requires time $mN^{\frac{1}{2}}$. We show that one can solve *all instances* in time only $\sqrt{m}N^{\frac{1}{2}}$, thereby saving a $\sqrt{m}$-factor and solving a single instance in amortized (over all m instances) cost $\left(\frac{N}{m}\right)^{\frac{1}{2}}$.

Precompution and Multiple Instances. The idea of precomputation allows for a combination with the multi-instance setting. Namely, one may precompute an *instance independent* structure. On obtaining m instances $y_1, \ldots, y_m$ one then lets m instance-dependent random walks $f_{\mathbf{g},y_1}, \ldots, f_{\mathbf{g},y_m}$ collide into the precomputed points.

This allows for various tradeoffs. For instance, one may precompute in time $mN^{\frac{2}{3}}$ a structure of size $m^2N^{\frac{1}{3}}$, which then in turn allows to solve *all m instances* in time only $N^{\frac{1}{3}}$.

Precomputation includes Multi-Instance. Technically, we apply and transfer the precomputation dlog framework by Corrigan-Gibbs and Kogan [CGK18]. This framework was already successfully applied to transfer precomputation and multi-instances from dlogs to the Legendre PRF setting [MZ22]. In fact, our analysis closely follows the reasoning for the precomputation setting in [CGK18,MZ22].

However, for the multi instance setting *without* precomputation we slightly deviate from [MZ22]. Namely, we observe that the multi-instance setting (without precomputation) is a special case of the multi-instance precomputation setting. While we will see that our observation is somewhat straight-forward (not to say trivial), to the best of our knowledge it has been overlooked in the cryptographic literature so far. Despite its triviality, our observation is also a bit counter-intuitive, probably explaining why it slipped through. Indeed, why should an algorithm *without* precomputation drop out from a precomputation scenario, which is usually expected to perform a heavy initial precomputation phase?

In fact, the trick is to perform only a light precomputation, balancing the cost between precomputation and online phase. This implies that we do not have to separate between precomputation and online phase any longer, thereby omitting precomputation altogether.

As a consequence of our observation, we obtain for free an appealingly simple multi-instance algorithm with a clean analysis for GA-dlogs. The same is true for ordinary dlogs, for which we explicitly provide a multi-instance algorithm from its precomputation algorithm. As opposed to other multi-instance dlog algorithms, our approach does not require the use of distinguished point techniques [KS01,FJM14], or heavy graph theory analysis [FJM14]. It is easy to see that our technique transfers to others settings as well, as e.g. to multi-instance Legendre PRF [MZ22] or to multi-user Even-Mansour [FJM14].

The Role of the Lattice Λ. Notice that in an ordinary discrete log setting over a group H, we assume the group order $|H|$ to be known. This helps us during random walks to update all discrete logs modulo $|H|$, thereby controlling their sizes. In elliptic curve groups the knowledge of $|H|$ is a reasonable assumption, since it can be computed via Schoof's algorithm [Sch95] in polynomial time.

Just as we assume knowledge of $|H|$ in the dlog setting, we assume in the group action dlog setting knowledge of a basis of Λ as input to our algorithms. Analogously, this allows us during random walks to update all GA-dlogs modulo Λ. In the CSIDH group action setting, a basis of Λ can be computed in quantum polynomial time [Hal05]. Classically, we may compute such a basis in subexponential time via the algorithm of [HM89]. For instance, for CSIDH-512 a basis has been computed in [BKV19]. Strictly speaking, our algorithms also work without knowing Λ, but without any reduction modulo Λ the GA-dlog output has exponential size.

Provability. We prove correctness, complexity and success probability of our algorithms without any heuristic assumptions, solely relying on a PRF realizing a mapping $\mathcal{X} \to \mathbb{Z}^n/\Lambda$ that we need for the analysis of our random walks. In our experiments for CSIDH, we show that we can easily realize a random mapping for CSIDH that works well in practice (although not being a PRF).

Notice that, by our mapping $\mathcal{X} \to \mathbb{Z}^n/\Lambda$, a single step in our random walks does not coincide with moving to a random neighbor in the so-called isogeny graph (as in other algorithms like [GHS02,GS13,DG16]), but we rather randomly jump in $\mathcal{X}$. This has the advantage that we do not have to care about the isogeny graph's mixing properties. On the downside, a single step in our walks is computationally more expensive. In our approach, we prefer clearity of exposition and provability over potential implementation practicality.

Implications for CSIDH-512. CSIDH-512 works with elliptic curves over $\mathbb{F}_p$ with 512-bit prime p, leading to $N = |\mathcal{G}| = |\mathcal{X}|$ of 256 bit size. Therefore, CSIDH-512 offers 128 bit security against the Delfs-Galbraith algorithm. Using our precomputation algorithm with parameter choice $s = t = N^{\frac{1}{3}}$ would lead to 171 bit of precomputations for a large scale adversary, resulting in a hint of size 85 bit.

Such a hint would allow to solve single GA-dlogs within only 85 bit. Since such a large scale attacker seems unrealistic, our precomputation attack currently does not directly affect the CSIDH-512 security level.

Organization of the paper. In Section 2, we define group action dlogs. Section 3 is devoted to our GA-dlog algorithm for a single instance with precomputation, which we generalize to multiple instances in Section 4. In Section 5, we show that a multi-instance GA-dlog without precomputation follows as a special case from the algorithm in Section 4. In Section 6, we provide as a further application and, for completeness, the analogous multi-instance dlog algorithm. In Section 7, we show that our precomputation GA-dlog algorithm works well in practice for small-scale parameter sets of CSIDH.

Implementation. The code for our CSIDH experiments is available at `https://github.com/maxostuzzi/precomputation_attack`.

2 Preliminaries

Let us define a group action and the discrete logarithm problem for group actions.

Definition 1 (Group action). *Let $(\mathcal{G}, \cdot)$ be a multiplicative group, and let $\mathcal{X}$ be a set. The map*

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

is called a group action of $\mathcal{G}$ on $\mathcal{X}$*, denoted $(\mathcal{G}, \mathcal{X}, \star)$, if it satisfies the properties*

1. Identity: $1 \star x = x$*, 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* $x \in \mathcal{X}$*.*

For efficient computations, we require that we can compactly represent $\mathcal{G}$ and $\mathcal{X}$. To this end we assume that $\mathcal{G}$ is finite and generated by $\mathbf{g} = \{g_1, \ldots, g_n\}$, denoted as $\mathcal{G} = \langle \mathbf{g} \rangle$. Moreover, we let $x \in \mathcal{X}$ be a distinguished element called *origin*, satisfying

$$\mathcal{X} = \{g \star x \mid g \in \mathcal{G}\} \quad \text{and} \quad |\mathcal{X}| = |\mathcal{G}| := N. \tag{2}$$

In other words, the map $\mathcal{G} \to \mathcal{X}, g \mapsto g \star x$ is a bijection. Group actions satisfying Equation (2) are called *regular* in the literature.

Definition 2 (Representation). *Let $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{Z}^n$. We denote $\mathbf{g}^{\mathbf{v}} := g_1^{v_1} \cdot \ldots \cdot g_n^{v_n} \in \mathcal{G}$. For some group element $\mathbf{g}^{\mathbf{v}} \in \mathcal{G}$, we call its exponent vector $\mathbf{v}$ a* representation*.*

Consider the exponentiation map $\phi_{\mathbf{g}} \colon \mathbb{Z}^n \to \mathcal{G}, \mathbf{v} \to \mathbf{g}^{\mathbf{v}}$. It is a surjective map and its kernel $\Lambda := \ker(\phi_{\mathbf{g}})$ is an n-dimensional lattice in $\mathbb{Z}^n$. Therefore, the map $\mathbb{Z}^n/\Lambda \to \mathcal{G}$ is an isomorphism, and every element in $\mathcal{G}$ has a representation that is unique modulo Λ.

Definition 3 (GA-dlog). *Let* $(\mathcal{G}, \mathcal{X}, \star)$ *be regular group action with generators* $\mathbf{g}$ *and origin* x*, namely*

$$\mathcal{G} = \langle \mathbf{g} \rangle = \langle g_1, \ldots, g_n \rangle \cong \mathbb{Z}^n/\Lambda, \quad \mathcal{X} = \{g \star x \mid g \in G\} \quad \text{and} \quad |\mathcal{G}| = |\mathcal{X}|.$$

In the group action discrete logarithm *(GA-dlog) problem, the goal is to find on input* $(\mathbf{g}, x, y) \in \mathcal{G}^n \times \mathcal{X}^2$ *the unique representation* $\mathbf{v} \in \mathbb{Z}^n/\Lambda$ *satisfying* $y = \mathbf{g}^{\mathbf{v}} \star x$*.*

3 Solving GA-dlogs with Precomputation

High-Level Description. Let us first give a high-level description of our group action dlog algorithm with precomputation, see also Figure 1.

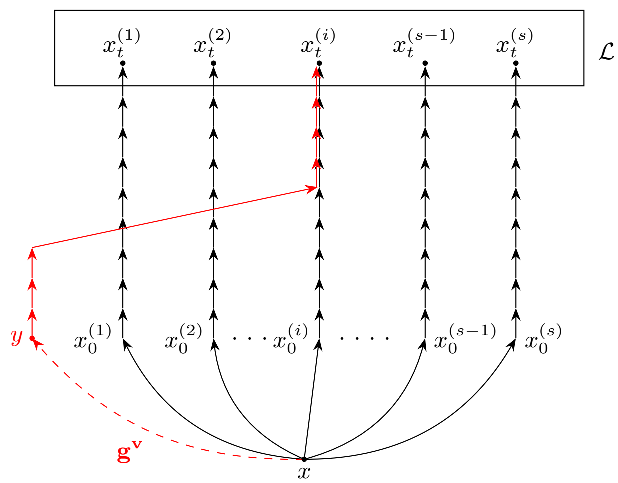

Fig. 1: Collision finding for GA-dlog instance $y = \mathbf{g}^{\mathbf{v}} \star x$ with precomputation.

Our *precomputation phase* is *instance-independent* and solely relies on the parameters $(\mathbf{g}, x)$ defining the group action. We start s instance-independent random walks $W^{(1)}, \ldots, W^{(s)}$ on $\mathcal{X}$, and store only their endpoints in a list $\mathcal{L}$.

In the *online phase*, we receive a group action dlog instance $y = \mathbf{g}^{\mathbf{v}} \star x$. We then start an *instance-dependent* random walk, and let it collide into one of the walks $W^{(1)}, \ldots, W^{(s)}$. The collision is identified via the stored endpoints. In the following we show that a collision immediately yields the desired GA-dlog $\mathbf{v}$.

Precomputation Phase. Let us describe a single random walk $W^{(i)}$ from the precomputation phase. To this end, we choose a random $\mathbf{w}_0^{(i)} \in \mathbb{Z}^n/\Lambda$ that defines a random starting point $x_0^{(i)} := \mathbf{g}^{\mathbf{w}_0^{(i)}} \star x \in \mathcal{X}$.

Let $h : \mathcal{X} \to \mathbb{Z}^n/\Lambda$ be a pseudorandom function (PRF), which allows us to define our random walk as

$$x_j^{(i)} := \mathbf{g}^{h(x_{j-1}^{(i)})} \star x_{j-1}^{(i)} \quad \text{for } j \geq 1. \tag{3}$$

In Section 7, we show how to instantiate h in a concrete setting. Moreover, let us define

$$\begin{aligned}\mathbf{w}_j^{(i)} &:= h(x_{j-1}^{(i)}) + \mathbf{w}_{j-1}^{(i)} \\ &= h(x_{j-1}^{(i)}) + h(x_{j-2}^{(i)}) + \mathbf{w}_{j-2}^{(i)} = \ldots = \mathbf{w}_0^{(i)} + \sum_{k=0}^{j-1} h(x_k^{(i)}) \quad \text{for } j \geq 1. \end{aligned} \tag{4}$$

Notice that, by the definition of our random walk and by the Compatibility property of Definition 1, we have

$$x_j^{(i)} = \mathbf{g}^{h(x_{j-1}^{(i)})} \star (\mathbf{g}^{h(x_{j-2}^{(i)})} \star \ldots \star (\mathbf{g}^{h(x_0^{(i)})} \star (\mathbf{g}^{\mathbf{w}_0^{(i)}} \star x)) \ldots) = \left(\mathbf{g}^{\mathbf{w}_0^{(i)}} \prod_{k=0}^{j-1} \mathbf{g}^{h(x_k^{(i)})} \right) \star x.$$

Therefore, we conclude that $x_j^{(i)}$ has GA-dlog $\mathbf{w}_j^{(i)}$, as defined in Eq. (4).

For each of the s random walks $W^{(i)}$ of length t, we store only their endpoint $x_t^{(i)}$ together with its GA-dlog $\mathbf{w}_t^{(i)}$.

The resulting precomputation is detailed in Precompute-GA (Algorithm 1).

Algorithm 1: Precompute-GA

Input: group action parameters $(\mathbf{g}, x) \in \mathcal{G}^n \times \mathcal{X}$, $N := |\mathcal{G}|$, basis for $\Lambda \subseteq \mathbb{Z}^n$, PRF $h : \mathcal{X} \to \mathbb{Z}^n/\Lambda$

Output: hint list $\mathcal{L}$ of endpoints/GA-dlogs $\left(x_t^{(i)}, \mathbf{w}_t^{(i)}\right) \in \mathcal{X} \times \mathbb{Z}^n/\Lambda$

Choose $s, t \in \mathbb{N}$ s.t. $4st^2 \leq N$ `// E.g.` $s, t = \lfloor \frac{1}{2} N^{\frac{1}{3}} \rfloor$
$\mathcal{L} \leftarrow \emptyset$
for $i = 1, \ldots, s$ `// Compute random walks` $W^{(1)}, \ldots, W^{(s)}$.
do
  Choose a random $\mathbf{w}_0^{(i)} \in \mathbb{Z}^n/\Lambda$.
  Let $x_0^{(i)} := \mathbf{g}^{\mathbf{w}_0^{(i)}} \star x \in \mathcal{X}$. `// Randomized starting point.`
  for $j = 1, \ldots, t$ `// Each walk` $W^{(i)}$ `has length` t.
  do
    Let $x_j^{(i)} := \mathbf{g}^{h(x_{j-1}^{(i)})} \star x_{j-1}^{(i)} \in \mathcal{X}$ and $\mathbf{w}_j^{(i)} := h(x_{j-1}^{(i)}) + \mathbf{w}_{j-1}^{(i)} \bmod \Lambda$.
  $\mathcal{L} \leftarrow \mathcal{L} \cup \left\{ \left(x_t^{(i)}, \mathbf{w}_t^{(i)} \right) \right\}$ `// Store endpoint/GA-dlog in hint` $\mathcal{L}$.
Sort $\mathcal{L}$ by first entry. `// Allows for binary search in` $\mathcal{L}$.
return $\mathcal{L}$

Online Phase. Let $y = \mathbf{g}^{\mathbf{v}} \star x$ be a GA-dlog instance. We start a random walk W as defined in Equation (3) from the starting point $x_0 := y$, see also Figure 1. As in the precomputation phase, we keep track of the GA-dlog. Namely, the random walk point x_j after j steps has GA-dlog

$$\mathbf{v}_j = \mathbf{v} + \sum_{k=0}^{j-1} h(x_k). \tag{5}$$

However, notice that $\mathbf{v}$ is the desired *unknown* GA-dlog of y, therefore we only store the value $\sum_{k=0}^{j-1} h(x_k)$.

Eventually, our walk W collides into one of the precomputed walks. Let $W^{(i)}$ be the colliding walk. Let us first show that once W collides into $W^{(i)}$, both walks subsequently visit the same points.

Algorithm 2: ONLINE-GA-DLOG

Input: $(\mathbf{g}, x, y = \mathbf{g}^{\mathbf{v}} \star x) \in \mathcal{G}^n \times \mathcal{X}^2, N := |\mathcal{G}|$, basis for $\Lambda \subseteq \mathbb{Z}^n$, precomputed hint $\mathcal{L} \in (\mathcal{X} \times \mathbb{Z}^n/\Lambda)^s$, t, PRF $h : \mathcal{X} \to \mathbb{Z}^n/\Lambda$

Output: GA-dlog $\mathbf{v} \in \mathbb{Z}^n/\Lambda$

Let $x_0 := y$ and $\mathbf{w}_0 := 0^n$.

for $j = 1, \ldots, 2t$ // $2t$-step walk $\overline{W}$

do

  Let $x_j := \mathbf{g}^{h(x_{j-1})} \star x_{j-1} \in \mathcal{X}$ and $\mathbf{w}_j := h(x_{j-1}) + \mathbf{w}_{j-1} \bmod \Lambda$.

  if $\left(x_j, \mathbf{w}_t^{(\ell)}\right) \in \mathcal{L}$ for some $\ell \in \{1, \ldots, s\}$ // Endpoint in $\mathcal{L}$?

  then

    return $\mathbf{v} := \mathbf{w}_t^{(\ell)} - \mathbf{w}_j \bmod \Lambda$

return FAIL

Colliding walks stay together. Let $x_j = x_k^{(i)}$ be the first collision between W and $W^{(i)}$. Then, by Equation (3), we have

$$x_{j+1} = \mathbf{g}^{h(x_j)} \star x_j = \mathbf{g}^{h(x_k^{(i)})} \star x_j = \mathbf{g}^{h(x_k^{(i)})} \star x_k^{(i)} = x_{k+1}^{(i)}.$$

Inductively, we obtain $x_{j+\ell} = x_{k+\ell}^{(i)}$ for all $\ell \geq 0$, which means that the walks stay together, see also Figure 1. As a consequence, the online phase walk W will eventually reach $W^{(i)}$'s stored endpoint $x_t^{(i)}$, together with its GA-dlog $\mathbf{v}^{(i)}$. It remains to show that the tuple $(x_t^{(i)}, \mathbf{v}_t^{(i)})$ reveals the solution of the GA-dlog instance y.

Endpoints solve GA-dlog. Let $x_m = x_t^{(i)}$ be the colliding endpoints of W and $W^{(i)}$. By Equation (5) and Equation (4), their GA-dlogs are

$$\mathbf{v}_m = \mathbf{v} + \sum_{k=0}^{m-1} h(x_k), \quad \text{respectively} \quad \mathbf{v}_t^{(i)} = \mathbf{v}_0^{(i)} + \sum_{k=0}^{t-1} h(x_k^{(i)}).$$

Since $\mathbf{v}_m = \mathbf{v}_t^{(i)} \bmod \Lambda$, we obtain the desired GA-dlog of y as

$$\mathbf{v} = \mathbf{v}_t^{(i)} - \sum_{k=0}^{m} h(x_k) \bmod \Lambda.$$

The resulting online phase is detailed in Online-GA-dlog (Algorithm 2).

Remark 1. Notice that Online-GA-dlog fails if we do not collide within $2t$ steps into one of the precomputed walks $W^{(1)}, \ldots, W^{(s)}$. In practice, one may then restart *only the online walk* with a fresh re-randomized starting point $x_0 := \mathbf{g}^{\mathbf{v}_0} \star y$, for some random $\mathbf{v}_0 \in \mathbb{Z}^n/\Lambda$. This amplifies the success probability arbitrarily close to 1, see also our experiments for CSIDH in Section 7.

Theorem 1. *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a regular group action with $N = |\mathcal{G}| = |\mathcal{X}|$. For any choice of $s, t \in \mathbb{N}$ with $4st^2 \leq N$,* Precompute-GA-dlog *(Algorithm 1) precomputes within st step a hint $\mathcal{L}$ of size $\tilde{O}(s)$. Using $\mathcal{L}$,* Online-GA-dlog *(Algorithm 2) solves a GA-dlog instance within $O(t)$ steps with success probability $\Omega\left(\frac{st^2}{N}\right)$.*

Proof. Precompute-GA computes s random walks of length t in a total of st steps. Since we store only their endpoints/GA-dlogs in our hint list $\mathcal{L}$, our memory requirement is $\tilde{O}(s)$. Online-GA-dlog performs at most $2t = O(t)$ steps to find a collision. If Online-GA-dlog collides within its first t steps into some precomputed walk $W^{(i)}$, then it reaches within its subsequent t steps an endpoint in $\mathcal{L}$. By the previous discussion, the endpoint yields the solution to the GA-dlog instance $y = \mathbf{g}^{\mathbf{v}} \star x$.

It remains to show that Online-GA-dlog succeeds to collide within t steps into a precomputed walk with probability $\Omega(\frac{st^2}{N})$. To this end, we show that Precompute-GA touches with probability at least $\frac{1}{2}$ within its st steps at least $st/2$ different elements in $\mathcal{X}$.

Let X_i be a random variable for the number of $\mathcal{X}$-elements visited by walk W_i. Let $X = \sum_{i=1}^{s} X_i \leq st$. Using the randomness property of our PRF h, Bernoulli's inequality, and $4st^2 \leq N$, we show that each walk $W^{(i)}$ visits the maximum number of $X_i = t$ new elements from $\mathcal{X}$ with probability at least

$$\mathbf{P}[X_i = t] \geq \left(\frac{N - st}{N}\right)^t = \left(1 - \frac{st}{N}\right)^t \geq 1 - \frac{st^2}{N} \geq \frac{3}{4}.$$

Hence, we have on expectation $\mathbf{E}[X_i] \geq \frac{3t}{4}$ newly visited $\mathcal{X}$-elements for each precomputed walk and, by linearity of expectation,

$$\mathbf{E}[X] = \sum_{i=1}^{s} \mathbf{E}[X_i] \geq \frac{3}{4} st.$$

Using Markov's inequality, we obtain

$$\mathbf{P}\left[X < \frac{st}{2}\right] \leq \mathbf{P}\left[st - X \geq \frac{st}{2}\right] \leq \frac{st - \mathbf{E}[X]}{\frac{st}{2}} \leq \frac{1}{2}.$$

Hence, with probability at least $\frac{1}{2}$ Precompute-GA visits within its st steps at least $X \geq st/2$ distinct elements in $\mathcal{X}$.

Let C_t be the event that Online-GA-dlog collides within the first t steps with one of the precomputed $\mathcal{X}$-elements, which is sufficient for Online-GA-dlog to solve the GA-dlog problem.

Using $1 - x \leq e^{-x}$ and $1 - e^{-x} \geq x/2$ for $x \leq 1$, we obtain

$$\mathbf{P}\left[\mathrm{C}_t \mid X \geq \frac{st}{2}\right] \geq 1 - \left(1 - \frac{st}{2N}\right)^t \geq 1 - e^{-\frac{st^2}{2N}} \geq \frac{st^2}{4N}.$$

Hence, Online-GA-dlog succeeds with probability at least

$$\mathbf{P}\left[C_t \cap \left(X \geq \frac{st}{2}\right)\right] = \mathbf{P}\left[X \geq \frac{st}{2}\right] \cdot \mathbf{P}\left[\mathrm{C}_t \mid X \geq \frac{st}{2}\right] \geq \frac{st^2}{8N} = \Omega\left(\frac{st^2}{N}\right). \quad \square$$

4 Solving Multiple GA-Dlogs with Precomputation

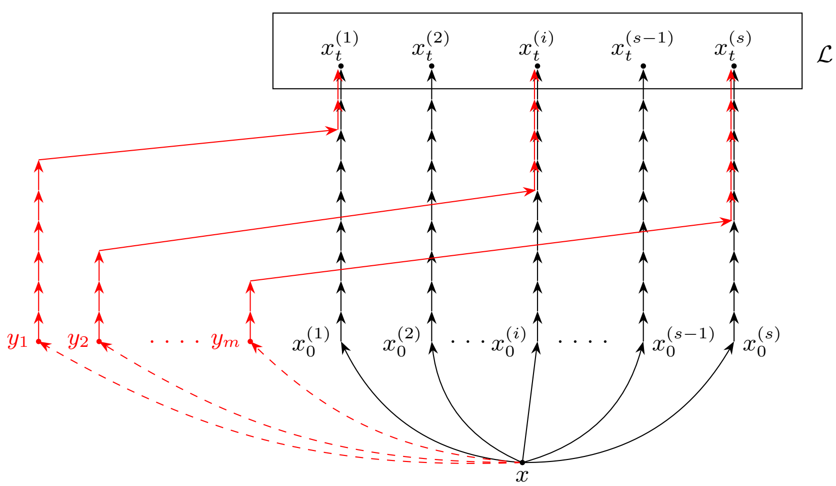

Fig. 2: Collision finding for multiple GA-dlogs with precomputation.

If an attacker makes the effort of a heavy precomputation for some group action instance $(\mathcal{G}, \mathcal{X}, \star)$, then the goal is usually not to online tackle just a single

GA-dlog instance, but rather to solve a large quantity of GA-dlog instances simultaneously.

Let $y_1 = \mathbf{g}^{\mathbf{v}_1} \star x, \ldots, y_m = \mathbf{g}^{\mathbf{v}_m} \star x$ be m GA-dlog instances. Our goal is to solve *all* instances. In a nutshell, a slightly heavier precomputation pays off in amortizing the cost over all m instances. As an example, we show in the following that a precomputation in time $m^2 N^{\frac{2}{3}}$ that produces a hint of size $m^2 N^{\frac{1}{3}}$ allows to solve *all m instances* in total time only $N^{\frac{1}{3}}$. Various other tradeoffs between precomputation time, hint size and online time are possible.

High-Level Idea. The multiple instance setting with precomputation is a natural generalization of the single instance setting from Section 3. Again, we precompute s random walks $W^{(1)}, \ldots, W^{(s)}$ in $\mathcal{X}$, and store only their endpoints, together with their GA-dlogs. During the online phase we then let all m online walks $\overline{W}^{(1)}, \ldots, \overline{W}^{(m)}$ collide into one of the precomputed walks. However, in order to obtain constant success probability for solving all m GA-dlog instances, we have to adjust the lengths of all walks during precomputation and online phase accordingly. Details of this adjustment follow.

Generalization to the Multi-Instance Setting. We provide our algorithms Precompute-mult-GA and Online-mult-GA-dlog in Algorithms 3 and 4. We advise the reader to compare to Precompute-GA and Online-GA-dlog (Algorithms 1 and 2) from Section 3.

Algorithm 3: Precompute-Mult-GA

Input: group action parameters $(\mathbf{g}, x) \in \mathcal{G}^n \times \mathcal{X}, N := |\mathcal{G}|$, number of instances m, basis for $\Lambda \subseteq \mathbb{Z}^n$, PRF $h : \mathcal{G} \to \mathbb{Z}^n/\Lambda$

Output: list $\mathcal{L}$ of endpoints/GA-dlogs $\left(x_{t/m}^{(i)}, \mathbf{w}_{t/m}^{(i)}\right) \in \mathcal{X} \times \mathbb{Z}^n/\Lambda$

Choose $s, t \in \mathbb{N}$ s.t. $4st^2 \leq m^2 N$ ▷ E.g. $s = m^2 N^{\frac{1}{3}}, t = \frac{1}{2} N^{\frac{1}{3}}$
$\mathcal{L} \leftarrow \emptyset$
for $i = 1, \ldots, s$ `// Compute random walks` $W^{(1)}, \ldots, W^{(s)}$.
do
  Choose a random $\mathbf{w}_0^{(i)} \in \mathbb{Z}^n/\Lambda$.
  Let $x_0^{(i)} := \mathbf{g}^{\mathbf{w}_0^{(i)}} \star x \in \mathcal{X}$. `// Randomized starting point.`
  for $j = 1, \ldots, t/m$ `// Each walk` $W^{(i)}$ `has length` t/m.
  do
    Let $x_j^{(i)} := \mathbf{g}^{h(x_{j-1}^{(i)})} \star x_{j-1}^{(i)} \in \mathcal{X}$ and $\mathbf{w}_j^{(i)} := h(x_{j-1}^{(i)}) + \mathbf{w}_{j-1}^{(i)} \bmod \Lambda$.
  $\mathcal{L} \leftarrow \mathcal{L} \cup \left\{\left(x_{t/m}^{(i)}, \mathbf{w}_{t/m}^{(i)}\right)\right\}$ `// Store endpoint/GA-dlog in hint` $\mathcal{L}$.
Sort $\mathcal{L}$ by first entry. `// Allows for binary search in` $\mathcal{L}$.
return $\mathcal{L}$

Algorithm 4: Online-mult-GA-dlog

Input: $(\mathbf{g}, x, y_1 = \mathbf{g}^{\mathbf{v}_1} \star x, \ldots, y_m = \mathbf{g}^{\mathbf{v}_m} \star x) \in \mathcal{G}^n \times \mathcal{X}^{m+1}, N := |\mathcal{G}|$,
basis for $\Lambda \subseteq \mathbb{Z}^n$, hint $\mathcal{L} \in (\mathcal{X} \times \mathbb{Z}^n/\Lambda)^s$, t, PRF $h : \mathcal{X} \to \mathbb{Z}^n/\Lambda$
Output: all GA-dlog $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{Z}^n/\Lambda$

for $i = 1, \ldots, m$ **do**
  Let $x_0 := y_i$ and $\mathbf{w}_0 := 0^n$. `// Start walk` $\overline{W}^{(i)}$ `for instance` y_i.
  for $j = 1, \ldots, 2t/m$ `//` $2rt/m$`-step walk`
  do
    Let $x_j := \mathbf{g}^{h(x_{j-1})} \star x_{j-1} \in \mathcal{X}$ and $\mathbf{w}_j := h(x_{j-1}) + \mathbf{w}_{j-1} \bmod \Lambda$.
    if $\left(x_j, \mathbf{w}^{(\ell)}_{t/m}\right) \in \mathcal{L}$ for some $\ell \in \{1, \ldots, s\}$ `// Endpoint in` $\mathcal{L}$`?`
    then
      return $\mathbf{v}_i := \mathbf{w}^{(\ell)}_{t/m} - \mathbf{w}_j \bmod \Lambda$
  return FAIL

Whereas in the single instance setting we obtained constant success probability by the parameter choice $st^2 = \Theta(N)$, in the multi instance setting the time/memory/instance tradeoff is $st^2 = \Theta(m^2N)$. This requires slightly larger s, t.

However, whereas in the single instance setting we needed walk lengths t respectively $2t$ for precomputation respectively online walks, in the multi instance setting shorter walk lengths t/m respectively $2t/m$ are sufficient.

Theorem 2. *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a regular group action with $N = |\mathcal{G}| = |\mathcal{X}|$, and let $y_i = \mathbf{g}^{\mathbf{v}_i} \star x$, $1 \leq i \leq m$ be GA-dlog instances. For any choice of $s, t \in \mathbb{N}$ with $4st^2 \leq m^2N$,* Precompute-Mult-GA *(Algorithm 3) precomputes within st/m steps a hint $\mathcal{L}$ of size $\tilde{O}(s)$. Using $\mathcal{L}$,* Online-Mult-GA-dlog *(Algorithm 4) runs in a total of $O(t)$ steps, and solves each GA-dlog y_i instance with success probability $\Omega\left(\frac{st^2}{m^2N}\right)$.*

Proof. The proof will closely follow the one for Theorem 1. Precompute-Mult-GA computes s walks with t/m steps each, with a total of st/m steps. Storing s endpoints requires memory $\tilde{O}(s)$. Moreover Online-Mult-GA-dlog performs m walks $\overline{W}^{(1)}, \ldots, \overline{W}^{(m)}$ with $2t/m$ steps, that is a total of $\mathcal{O}(t)$ steps. It remains to show the success probability of Online-Mult-GA-dlog.

Let X_i be a random variable counting the number of new elements in $\mathcal{X}$ touched by the precomputed random walk W_i, for $i \in \{1, \ldots, s\}$. Let $X = \sum_{i=1}^{s} X_i$. As $X_i \leq \frac{t}{m}$, we have that $X \leq \frac{st}{m}$. Using Bernoulli's inequality and $4st^2 \leq m^2N$, each walk touches the maximum number $X_i = t/m$ of new elements with probability

$$\mathbf{P}[X_i = t/m] \geq \left(\frac{N - \frac{st}{m}}{N}\right)^{t/m} = \left(1 - \frac{st}{mN}\right)^{t/m} \geq 1 - \frac{st^2}{m^2N} \geq \frac{3}{4}.$$

Therefore, we have $\mathbf{E}[X_i] \geq \frac{3}{4} \cdot \frac{t}{m} = \frac{3t}{4m}$ and $\mathbf{E}[X] \geq \frac{3st}{4m}$. By Markov's inequality, we obtain

$$\mathbf{P}\left[X < \frac{st}{2m}\right] \leq \mathbf{P}\left[\frac{st}{m} - X \geq \frac{st}{2m}\right] \leq \frac{\frac{st}{m} - \mathbf{E}[X]}{\frac{st}{2m}} \leq \frac{1}{2}.$$

Let $E^{(i)}$ be the event that online walk $\overline{W}^{(i)}$ collides with some precomputed walk within the first t/m steps. In this case, $\overline{W}^{(i)}$ reaches within the subsequent t/m an endpoint in $\mathcal{L}$, thereby solving the GA-dlog instance $y_i = \mathbf{g}^{\mathbf{v}_i} \star x$.

Using $1 - x \leq e^{-x}$ and $1 - e^{-x} \geq x/2$ for $x \leq 1$, we obtain

$$\mathbf{P}\left[E^{(i)} \mid X \geq \frac{st}{2m}\right] \geq 1 - \left(1 - \frac{st}{2mN}\right)^{\frac{t}{m}} \geq 1 - e^{-\frac{st^2}{2m^2N}} \geq \frac{st^2}{4m^2|\mathcal{X}|}.$$

Thus, the i-th instance y_i can be solved with probability at least

$$\mathbf{P}\left[E^{(i)} \cap \left(X \geq \frac{st}{2m}\right)\right] \geq \frac{st^2}{8m^2N} = \Omega\left(\frac{st^2}{8m^2N}\right). \quad \square$$

Remark 2. Theorem 2 guarantees constant success probability of Online-mult-GA-dlog for each GA-dlog instance $y_i = \mathbf{g}^{\mathbf{v}_i} \star x$ if $st^2 = \Omega(m^2N)$. If Online-mult-GA-dlog fails for some y_i, one simply repeats online walk $\overline{W}^{(i)}$ with a fresh re-randomized starting point, see also Remark 1.

5 Solving Multiple GA-dlogs (without Precomputation)

Interestingly, our multi instance GA-dlog algorithms Precompute-mult-GA (Algorithm 3) and Online-mult-GA-dlog (Algorithm 4) also provide a solution to the multi instance GA-dlog problem *without precomputation.*

Let $y_1 = \mathbf{g}^{\mathbf{v}_1} \star x, \ldots, y_m = \mathbf{g}^{\mathbf{v}_m} \star x$ be m GA-dlog instances. Now apply Theorem 2 with the parameter choice $s = m$ and $t = \frac{1}{2}\sqrt{mN}$. We obtain the following corollary.

Corollary 1. *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a regular group action with $N = |\mathcal{G}| = |\mathcal{X}|$, and let $y_i = \mathbf{g}^{\mathbf{v}_i} \star x$, $1 \leq i \leq m$ be GA-dlog instances. Then* Precompute-Mult-GA *(Algorithm 3) precomputes within $t = O(\sqrt{mN})$ steps a hint $\mathcal{L}$ of size $\tilde{O}(s) = \tilde{O}(m)$. Using $\mathcal{L}$,* Online-Mult-GA-dlog *(Algorithm 4) runs in a total of $O(t) = O(\sqrt{mN})$ steps, and solves each GA-dlog instance y_i with constant success probability.*

Notice that our parameter choice balances the run times of the precomputation phase and the online phase. Thus, we can rewrite Corollary 1 more compactly as follows.

Corollary 2. *Let $(\mathcal{G}, \mathcal{X}, \star)$ be a regular group action with $N = |\mathcal{G}| = |\mathcal{X}|$, and let $y_i = \mathbf{g}^{\mathbf{v}_i} \star x$, $1 \leq i \leq m$ be GA-dlog instances. Then one can solve each GA-dlog y_i with constant success probability within a total of $O(\sqrt{mN})$ steps.*

Multiple GA-dlog Algorithm. Our algorithm behind Corollary 2 computes in a first phase (previously: precomputation) m instance-independent walks $W^{(1)}, \dots, W^{(m)}$. In a second phase (previously: online), we let all instance-dependent walks $\overline{W}^{(i)}$, $1 \le i \le m$ collide into the walks $W^{(1)}, \dots, W^{(m)}$, thereby solving all GA-dlog instances y_i.

Intuition of the Achieved $\sqrt{m}$ Speedup. The GHS algorithm can be considered a special case of our aforementioned *multiple GA-dlog algorithm* with $m = 1$. The GHS algorithm lets some instance-independent walk $W^{(1)}$ of length $\sqrt{N}$ collide with an instance-dependent walk of length $\sqrt{N}$. Since both walks have length $\sqrt{N}$ we have $(\sqrt{N})^2$ pairs of elements in $\mathcal{X}$ that can potentially collide, resulting in constant success probability.

Moreover, our multiple GA-dlog algorithm can be considered as running m copies of GHS *simultaneously.* So why do we actually achieve run time $O(\sqrt{mN})$? And why do we achieve a $\sqrt{m}$ speedup over the naive $O(m\sqrt{N})$?

Indeed, running m independent copies of GHS gives us run time $O(m\sqrt{N})$. Our speedup comes from the *simultaneous* instantiation. After the first phase, we have m walks $W^{(1)}, \dots, W^{(m)}$, each of length t/m, visiting a total of (roughly) $m \cdot \frac{t}{m} = t$ elements. For any instance y_i, it suffices that its walk $\overline{W}^{(i)}$ with length roughly t/m collides into *any* of these m walks. This happens with constant probability if the product of visited elements in $\mathcal{X}$ in *all* instance-independent walks with the instance-dependent walk length roughly equals N. More precisely, we require

$$\left(m \cdot \frac{t}{m}\right) \cdot \frac{t}{m} = \Theta(N).$$

Solving for run time t yields $t = \Theta(\sqrt{mN})$.

6 Classical Multiple Dlogs (without Precomputation)

The algorithm underlying Corollary 2 also applies in the classical multi instance dlog setting [CGK18], as well as for other multi instance settings [FJM14,MZ22].

We provide the *multi instance dlog algorithm* for completeness in Algorithm 5, since this setting is of great importance in cryptography. The algorithm is appealingly simple and allows for a clean analysis.

Let $(G, \cdot) = \langle g \rangle$ be a finite abelian group with $N = |G|$. Let $y_1 = g^{v_1}, \dots, y_m = g^{v_m}$ be m dlog instances with $v_i \in \mathbb{Z}_N$. Then MULT-DLOG (Algorithm 5) describes an algorithm for solving all dlog instances.

Theorem 3. *Let $(G, \cdot) = \langle g \rangle$ be an abelian group with order $N = |G|$, and let $y_i = g^{v_i}$, $1 \le i \le m$ be dlog instances. Then* MULT-DLOG *(Algorithm 5) outputs within an expected $O(\sqrt{mN})$ steps all dlogs $v_i \in \mathbb{Z}^n$.*

Proof. The proof is a special case of the proof of Theorem 2 with the parameter choice $s = m$ and $t = \frac{1}{2}\sqrt{mN}$. Theorem 2 guarantees for this parameter choice

Algorithm 5: MULT-DLOG

Input: $(g, y_1 = g^{v_1}, \dots, y_m = g^{v_m}) \in G^{m+1}, N := |G|$, PRF $h : G \to \mathbb{Z}_N$
Output: all dlogs $v_1, \dots, v_m \in \mathbb{Z}_N$

$\mathcal{L} \leftarrow \emptyset$
for $i = 1, \dots, m$ **do** // Compute random walks $W^{(1)}, \dots, W^{(m)}$.
  Choose a random $w_0^{(i)} \in \mathbb{Z}_N$.
  Let $x_0^{(i)} := g^{w_0^{(i)}} \in G$. // Randomized starting point.
  for $j = 1, \dots, t/m$ **do** // Each walk $W^{(i)}$ has length t/m.
    Let $x_j^{(i)} := g^{h(x_{j-1}^{(i)})} \cdot x_{j-1}^{(i)} \in G$ and $w_j^{(i)} := h(x_{j-1}^{(i)}) + w_{j-1}^{(i)} \bmod N$.
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(x_{t/m}^{(i)}, w_{t/m}^{(i)})\}$ // Store endpoint/dlog in $\mathcal{L}$.
Sort $\mathcal{L}$ by first entry. // Allows for binary search in $\mathcal{L}$.
for $i = 1, \dots, m$ **do**
  while instance y_i is unsolved **do**
    Let $x_0 := y_i$ and $w_0 \in_R \mathbb{Z}_N$. // Start walk $\overline{W}^{(i)}$ for instance y_i.
    for $j = 1, \dots, 2t/m$ **do** // $2t/m$-step walk
      Let $x_j := g^{h(x_{j-1})} \cdot x_{j-1} \in G$ and $w_j := h(x_{j-1}) + w_{j-1} \bmod N$.
      if $\left(x_j, w_{t/m}^{(\ell)}\right) \in \mathcal{L}$ for some $\ell \in \{1, \dots m\}$ **then** // Endpoint in $\mathcal{L}$?
        return $v_\ell := w_{t/m}^{(\ell)} - w_j \bmod N$

for every dlog instance y_i constant success probability $\epsilon = \Omega(1)$. Therefore, we solve every instance y_i after an expected $\frac{1}{\epsilon} = O(1)$ re-randomized runs of $\overline{W}^{(i)}$. □

7 Experimental Results for CSIDH

In this section we instantiate our precomputation algorithm from Section 3 for the prominent group action based post-quantum scheme CSIDH.

Recall that, by Definition 1, for a group $\mathcal{G}$ and a set $\mathcal{X}$ a group action is given by a mapping $\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$. Moreover, a group action is called *regular* if $\mathcal{X} = \{g \star x \mid g \in \mathcal{G}\}$ and $|\mathcal{X}| = |\mathcal{G}|$. In order to instantiate our algorithms, we first need to define $(\mathcal{G}, \mathcal{X}, \star)$ for CSIDH.

The Group $\mathcal{G}$ in CSIDH. Let E be an elliptic curve defined over $\mathbb{F}_p$, for some prime number $p \geq 5$, and let $\mathcal{O}$ denote the $\mathbb{F}_p$-rational endomorphism ring of E. We say that E is *supersingular* if it satisfies $|E(\mathbb{F}_p)| = p + 1$. We choose p such that $p + 1 = 4 \cdot \prod_{i=1}^{n} \ell_i$, with $\ell_1 < \ldots < \ell_n$ small odd primes and $\ell_1 = 3$. This property ensures that the ideal $\ell_i\mathcal{O}$ decomposes as the product of two prime ideals $\mathfrak{l}_i$ and $\overline{\mathfrak{l}_i}$, namely $\ell_i\mathcal{O} = \mathfrak{l}_i\overline{\mathfrak{l}_i}$. Moreover, let us denote by $\mathrm{Cl}(\mathcal{O})$ the class group of $\mathcal{O}$, and by $[\mathfrak{a}] \in \mathrm{Cl}(\mathcal{O})$ the equivalence class of $\mathfrak{a}$ in $\mathrm{Cl}(\mathcal{O})$, see [Cox22] for details.

We now define the (sub)group $\mathcal{G} = \langle \mathbf{g} \rangle \subseteq \mathrm{Cl}(\mathcal{O})$, where $\mathbf{g} = \{[\mathfrak{l}_1], \dots, [\mathfrak{l}_n]\}$. According to a heuristic from [Sie35], for a given prime p of this form, the size of the group $\mathcal{G}$ is approximately $N \approx p^{1/2}$.

The Set $\mathcal{X}$ and the Action $\star$ in CSIDH. Every supersingular elliptic curve over $\mathbb{F}_p$ is defined by an equation

$$Y^2 = X^3 + AX^2 + X,$$

where $A \in \mathbb{F}_p$ is called *Montgomery coefficient*. As shown in [CLM+18], supersingular elliptic curves can be uniquely represented (up to isomorphism) by their Montgomery coefficient A. Let $\mathcal{M} \subseteq \mathbb{F}_p$ be the set of all Montgomery coefficients of supersingular curves over $\mathbb{F}_p$. By our choice of the prime p, we have that the curve with equation $y^2 = x^3 + x$ is supersingular with Montgomery coefficient $A = 0 \in \mathcal{M}$. We define $x = 0 \in \mathcal{M}$ as the origin of $\mathcal{X}$.

It is known that the class group $\mathrm{Cl}(\mathcal{O})$ acts regularly on $\mathcal{M}$, see [CLM+18,Sil09]. Let us denote this action by $\star : \mathrm{Cl}(\mathcal{O}) \times \mathcal{M} \to \mathcal{M}$, and let us define the set $\mathcal{X} = \{g \star x \mid g \in \mathcal{G}\}$. This implies regularity of the action $\star : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$ restricted to $\mathcal{G} \subseteq \mathrm{Cl}(\mathcal{O})$ and $\mathcal{X} \subseteq \mathcal{M} \subseteq \mathbb{F}_p$.

We now have a well-defined regular group action $(\mathcal{G}, \mathcal{X}, \star)$ with origin $x = 0$ and generators $\mathbf{g} = \{[\mathfrak{l}_1], \dots, [\mathfrak{l}_n]\}$. As in Definition 2, we have representations for the group $\mathcal{G}$, namely for $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{Z}^n$ we obtain $\mathbf{g}^{\mathbf{v}} = \prod_{i=1}^n [\mathfrak{l}_i]^{v_1}$.

Admissible Representations. Let $\mathbf{v} \in \mathbb{Z}^n$ be a representation. The complexity for evaluating the action of the group element $\mathbf{g}^{\mathbf{v}}$ scales proportionally with $||\mathbf{v}||_1$, see [BDFLS20,Vél71]. Therefore, only group elements with small 1-norm representation can be efficiently computed via the action. For this reason, CSIDH only uses representations within the set $\mathcal{R} = \{-d, ..., d\}^n$, for some integer $d \in \mathbb{N}$, see [CLM+18]. We say that a representation $\mathbf{v}$ is *admissible* if $\mathbf{v} \in \mathcal{R}$. In practice, CSIDH parameters require the inequality $(2d+1)^n \geq N$ to ensure that $\{-d, ..., d\}^n$ properly covers the whole $\mathcal{G}$. The smallest d satisfying this inequality is $d = \left\lceil \frac{\sqrt[n]{N}-1}{2} \right\rceil$.

Instantiating a Random Function for CSIDH. Let us identify $\{-d, \dots, d\}$ with $\mathbb{Z}_{2d+1}$. For instantiating our algorithm from Section 3 in the CSIDH setting we need a function $h : \mathcal{X} \to \mathbb{Z}_{2d+1}^n$, where $\mathcal{X} \subseteq \mathbb{F}_p$ and $p^{\frac{1}{2}} \approx N = |\mathcal{X}| \leq (2d+1)^n$. It follows that $p \leq (2d+1)^{2n}$, and thus our function is $(2:1)$-compressing.

We provide our function $h_{a,b}$ in Algorithm 6 for some $a, b \in \mathbb{F}_p$ chosen uniformly at random. The purpose of a, b is to re-randomize elements from $\mathcal{X}$ in $\mathbb{F}_p$. Clearly, our function is not a PRF in the cryptographic sense, but our experiments indicate that its randomness properties are sufficient for practical purposes.

Algorithm 6: CSIDH $h_{a,b}$

Input: Montgomery coefficient $A \in \mathcal{X} \subseteq \mathbb{F}_p, a, b \in \mathbb{F}_p, p, d, n \in \mathbb{N}$
Output: admissible CSIDH representation $\mathbf{v} \in \mathbb{Z}_{2d+1}^n$

```
Let y := aA + b mod p.                                    // Re-randomize A.
Expand y in base (2d+1) as y = sum_{i=0}^{2n-1} y_i(2d+1)^i with y_i ∈ Z_{2d+1}.
for i = 0, ..., n-1 do
    Let v_i := y_i - y_{i+n} mod 2d+1.                    // (2:1)-compression
return v = (v_1, ..., v_n) ∈ Z_{2d+1}^n
```

Experiments. We implemented our GA-dlog algorithms Precompute-GA (Algorithm 1) and Online-GA-dlog (Algorithm 2) from Section 3 together with the function $h_{a,b}$ (Algorithm 6). Our code is available at `https://github.com/maxostuzzi/precomputation_attack`.

In our implementation, we slightly deviate from the description of Online-GA-dlog by restarting an online walk with a re-randomized starting point if it fails to collide into an endpoint after $2t$ step, see also Remark 1 and Algorithm 5. Our algorithms are instantiated with the parameter choice $s, t = N^{\frac{1}{3}}$.

We applied our algorithm Online-GA-Dlog on ten different primes p. For each prime p, we ran ten GA-dlog instances $y_1, \ldots, y_{10}$, until all of them where successfully solved. In Table 1, we list our CSIDH primes p with their respective small odd primes ℓ_i, where $p + 1 = 4\prod_i \ell_i$. For every prime p, we measured the number of random walks of length $2t$ we had to perform for $y_1, \ldots, y_{10}$, and then averaged the number over all 10 instances.

From Table 1, we see that all averages are close to 1. In fact, out of our total of 100 solved instances, 91 were solved by running a single $2t$-step random walk, indicating a large success probability per random walk.

Our experiments also clearly show that our function $h_{a,b}$ (Algorithm 6) provides sufficient randomness in practice.

Table 1: CSIDH instances and average number of random walk.

p	ℓ_i small odd primes	# of runs
1019	$3, 5, 17$	1.3
78539	$3, 5, 7, 11, 17$	1.0
1021019	$3, 5, 7, 11, 13, 17$	1.2
19399379	$3, 5, 7, 11, 13, 17, 19$	1.0
1450388939	$3, 5, 7, 11, 13, 19, 31, 41$	1.0
53664390779	$3, 5, 7, 11, 13, 19, 31, 37, 41$	1.0
8575569646643	$3, 7, 11, 13, 17, 19, 31, 37, 41, 47$	1.1
454505191272131	$3, 7, 11, 13, 17, 19, 31, 37, 41, 47, 53$	1.2
26815806285055787	$3, 7, 11, 13, 17, 19, 31, 37, 41, 47, 53, 59$	1.1
138624083338000259	$3, 5, 7, 11, 13, 17, 19, 31, 37, 41, 47, 53, 61$	1.1

Figure 3 shows the logarithm of the number of steps (as a function of $\log N = \frac{1}{2}\log p$) that ONLINE-GA-DLOG performed until it successfully recovered a GA-dlog, including potential restarts of a walk. Again, for every p we averaged the number of steps over all 10 solved instances.

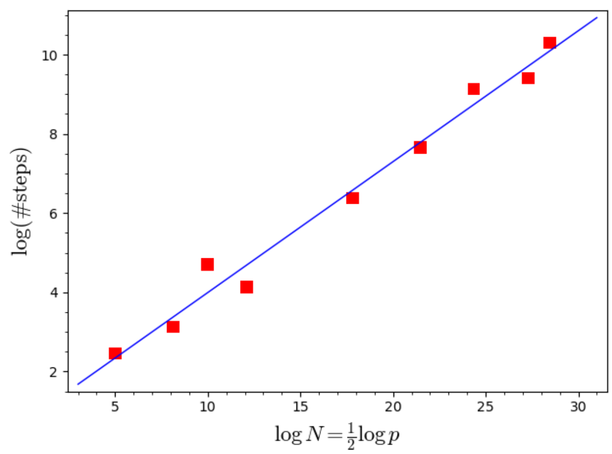

Fig. 3: Performance of ONLINE-GA-DLOG on CSIDH. For every of our 10 choices for p, we averaged the number of steps over 10 solved instances.

Despite the re-randomization, which clearly increases the number of steps in the online phase, the slope of the fitting line for our experiments is 0.33, as expected.

References

ACC$^+$17. Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 152:154–155, 2017.

BDFLS20. Daniel J Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *Open Book Series*, 4(1):39–55, 2020.

BKV19. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. Csi-fish: efficient isogeny based signatures through class group computations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 227–247. Springer, 2019.

BL12. Daniel J Bernstein and Tanja Lange. Computing small discrete logarithms faster. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*, pages 317–338. Springer, 2012.

BS20. Xavier Bonnetain and André Schrottenloher. Quantum security analysis of csidh. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 493–522. Springer, 2020.

BY91. Gilles Brassard and Moti Yung. One-way group actions. In *Advances in Cryptology-CRYPTO'90: Proceedings 10*, pages 94–107. Springer, 1991.

CD20. Wouter Castryck and Thomas Decru. Csidh on the surface. In *International Conference on Post-Quantum Cryptography*, pages 111–129. Springer, 2020.

CGK18. Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 415–447. Springer, 2018.

CJS14. Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.

CLJ06. Henri Cohen and Hendrik W Lenstra Jr. Heuristics on class groups of number fields. In *Number Theory Noordwijkerhout 1983: Proceedings of the Journées Arithmétiques held at Noordwijkerhout, The Netherlands July 11–15, 1983*, pages 33–62. Springer, 2006.

CLM+18. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: an efficient post-quantum commutative group action. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 395–427. Springer, 2018.

Cou06. Jean-Marc Couveignes. Hard homogeneous spaces. *Cryptology ePrint Archive*, 2006.

Cox22. David A Cox. *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication with Solutions*, volume 387. American Mathematical Soc., 2022.

CRSCS22. Maria Corte-Real Santos, Craig Costello, and Jia Shi. Accelerating the delfs–galbraith algorithm with fast subfield root detection. In *Annual International Cryptology Conference*, pages 285–314. Springer, 2022.

DFJP14. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

DFKS18. Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 365–394. Springer, 2018.

DG16. Christina Delfs and Steven D Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Designs, Codes and Cryptography*, 78:425–440, 2016.

ER+60. Paul Erdös, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60, 1960.

FJM14. Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logarithm, even-mansour and prince. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 420–438. Springer, 2014.

GHS02. Steven D Galbraith, Florian Hess, and Nigel P Smart. Extending the ghs weil descent attack. In *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21*, pages 29–44. Springer, 2002.

GS13. Steven Galbraith and Anton Stolbunov. Improved algorithm for the isogeny problem for ordinary elliptic curves. *Applicable Algebra in Engineering, Communication and Computing*, 24(2):107–131, 2013.

Hal05. Sean Hallgren. Fast quantum algorithms for computing the unit group and class group of a number field. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 468–474, 2005.

HDWH12. Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220, 2012.

HM89. James L Hafner and Kevin S McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American mathematical society*, 2(4):837–850, 1989.

JDF11. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*, pages 19–34. Springer, 2011.

Koh96. David Russell Kohel. *Endomorphism rings of elliptic curves over finite fields*. University of California, Berkeley, 1996.

KS01. Fabian Kuhn and René Struik. Random walks revisited: Extensions of pollard's rho algorithm for computing multiple discrete logarithms. In *International Workshop on Selected Areas in Cryptography*, pages 212–229. Springer, 2001.

Kup05. Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.

MZ22. Alexander May and Floyd Zweydinger. Legendre prf (multiple) key attacks and the power of preprocessing. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pages 428–438. IEEE, 2022.

Pei20. Chris Peikert. He gives c-sieves on the csidh. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 463–492. Springer, 2020.

PH78. S Pohlig and M Hellman. An improved algorithm for computing logarithms over gf (p) and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

Pol78. John M Pollard. Monte carlo methods for index computation. *Mathematics of computation*, 32(143):918–924, 1978.

Sch95. René Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.

Sho94. Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

Sie35. Carl Siegel. Über die Classenzahl quadratischer Zahlkörper. *Acta Arithmetica*, 1(1):83–86, 1935.

Sil09. Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

Sto10. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.*, 4(2):215–235, 2010.

Vél71. Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes-Rendus de l'Académie des Sciences*, 273:238–241, 1971.

VOW99. Paul C Van Oorschot and Michael J Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12:1–28, 1999.

Yun15. Aaram Yun. Generic hardness of the multiple discrete logarithm problem. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 817–836. Springer, 2015.