





A Lower Bound on the Length of Signatures Based on Group Actions and Generic Isogenies

Dan Boneh¹, Jiaxin Guan²(✉) , and Mark Zhandry^{2,3} 

¹ Stanford University, Stanford, USA

² Princeton University, Princeton, USA
jiaxin@guan.io

³ NTT Research, Inc., Sunnyvale, USA

Abstract. We give the first black box lower bound for signature protocols that can be described as group actions, which include many based on isogenies. We show that, for a large class of signature schemes making black box use of a (potentially non-abelian) group action, the signature length must be $\Omega(\lambda^2/\log \lambda)$. Our class of signatures generalizes all known signatures that derive security exclusively from the group action, and our lower bound matches the state of the art, showing that the signature length cannot be improved without deviating from the group action framework.

1 Introduction

Post-quantum cryptography aims to develop classical cryptosystems that remain secure against an adversary who has access to a large-scale quantum computer. One approach to post-quantum cryptography relies on the observation that Shor’s discrete log algorithm [35] does not apply in an algebraic structure called a *group action*. This gives rise to group-action-based cryptography for post-quantum public key encryption, key exchange, digital signatures, and more [1, 4, 21]. The resulting cryptosystems look somewhat similar to classical systems that rely on the difficulty of discrete log in a finite cyclic group.

Informally, a group action is a mapping of the form $*$: $G \times X \rightarrow X$, where G is a finite group and X is a set, such that for any $g_1, g_2 \in G$ and any $x \in X$, we have $g_1 * (g_2 * x) = (g_1 g_2) * x$. Moreover, if $e \in G$ is the identity of G then $e * x = x$ for all $x \in X$. The discrete log problem for such a group action is to find a $g \in G$, if one exists, such that $x_0 = g * x_1$, given only $x_0, x_1 \in X$ as input. Note that Shor’s algorithm fails to solve this problem precisely because there is no efficiently computable group operation on the set X . The best known quantum algorithms for group action discrete log run in sub-exponential time in the security parameter [22, 23, 29, 31].

Currently the most widely studied cryptographic group action is derived from isogeny graphs of elliptic curves [7, 33]. To avoid the sub-exponential quantum algorithm mentioned above, some constructions use *supersingular* isogeny

graphs [17,20], which present less structure than a group action. However, a recent attack by Castryck and Decru [5] and Maino and Martindale [25] shows that certain key exchange protocols that rely on supersingular isogeny graphs (in particular, rely on the SIDH assumption) are insecure. The attack does not appear to affect various isogeny-based signature schemes such as SeaSign [9,12], CSI-FiSh [3,8,16], and SQISign [10,11].

Short Signatures. An important open problem in post-quantum cryptography is to construct a signature scheme for which the combined length of a public key and a signature is comparable to that of the Schnorr scheme, namely $32 + 64 = 96$ bytes (for 128-bit security). The four post-quantum NIST signature finalists [6] have the following combined public-key/signature lengths: 3740 bytes for Dilithium2, 1563 bytes for Falcon512, and over 50 KB for both Rainbow variants. These numbers are an order of magnitude higher than the combined length for the Schnorr scheme. We note that the combined public-key/signature length for SQISign is only 268 bytes—better than the NIST candidates, but still worse than Schnorr. SQISign uses specific properties of supersingular isogenies, and is not a generic group-action signature scheme.

Can we do better? One might expect that due to the similarity between group-action-based systems and systems using a finite cyclic group, one should be able to design a post-quantum Schnorr-like signature scheme using a generic group action. However, this remains an open problem. For example, the signature scheme SeaSign [9], which can be described as a generic group-action signature scheme (as in Sect. 1.3), has a combined public-key/signature length of about 3 KB (see column 3 of Table 2 from [9]). In this paper we show that this is no accident.

1.1 Our Results

Let λ be a security parameter. Our main result is a lower bound of $\Omega(\lambda^2/\log \lambda)$ for a wide class of group-action-based signatures. This lower bound matches the signature length of state-of-the-art constructions such as SeaSign. Concretely, we prove the following theorem about *identification* (ID) protocols:

Theorem 1.1 (Informal). *For any public-coin identification protocol secure against eavesdropping in a black box (potentially non-abelian) group action model, the sender must send at least $(\lambda - 1)/\log_2 \lambda$ set elements in order to achieve soundness $2^{-\lambda}$.*

Here, public coin means that the verifier generates its messages by simply sampling uniform bit strings. Note that Theorem 1.1 works for any such group action; in particular we do not assume any regularity or transitivity. Also note that by handling *non-abelian* group actions, our model easily incorporates features like twists, as twists can be seen as action by a slightly larger group arising from a semi-direct product.

Since set elements need to be at a minimum λ bits to prevent solving discrete logarithms, we thus obtain a lower bound of $(\lambda - 1)^2 / \log_2 \lambda$ bits for the communication from prover to verifier.

All known efficient group-action-based signature schemes are built by transforming a public coin ID protocol into a signature, typically via Fiat-Shamir [18], but other transforms are also possible [37]. Thus, our lower bound yields a lower bound on the length of signatures in such protocols.

Our Model of Black Box Group Actions. We formalize black box group actions by adapting Maurer’s [26] generic group model to the black box group action setting. In this model, instead of getting set elements “in the clear”, all parties are only given handles to the set elements, and then operate on these handles via an oracle. This reflects how current group-action based signature and ID protocols are constructed. Below, we discuss why we choose to adapt Maurer’s model instead of Shoup’s [36] model.

Extensions. We also discuss several extensions to structures that generalize group actions. In particular, many isogenies cannot be framed straightforwardly as group actions. We therefore formalize a *graph action* model, which generalizes group actions to these more general structures, and observe that our impossibility readily applies in this more general setting as well.

1.2 Discussion

Schnorr identification requires sending only a single group element, and security can be proven under the discrete logarithm assumption in plain groups. Theorem 1.1 shows that the situation is quite different in the group action setting. In the language of [32], our result shows there is no semi-black box construction of an efficient ID protocol from hard discrete logarithms over group actions. Even more, “discrete logarithm” can be replaced by *any* problem that is (classically) unconditionally hard in generic group actions, including CDH and even more exotic assumptions such as the linear hidden shift assumption [1]. Thus, to sidestep our lower bound, one must design signatures that are *not* based on ID protocols, rely on non-generic use of the group action, or rely on cryptographic hardness assumptions beyond what a group action alone provides.

On Our Black Box Model. A natural question is whether our lower bound also applies to an analog of Shoup’s generic group model tailored to group actions, replacing handles with random labels. Unfortunately, lower bounding signatures in Shoup’s model appears to be very challenging. In particular, a lower bound in such a model would imply as a special case a lower bound in the *random oracle model* (ROM)¹. Even through the best-known (many-time) random oracle signatures have signature size $\Omega(\lambda^3)$ [27]², it is a long-standing open problem to

¹ Shoup’s generic groups imply random oracles [39], and the proof readily adapts to group actions with random labels.

² If the number of messages is a priori bounded, it is possible to have signatures of length $O(\lambda^2)$ [28].

obtain *any* non-trivial bound. We cannot even rule out that optimal $O(\lambda)$ -length signatures from random oracles exist. We sidestep this major barrier by instead utilizing Maurer’s model.

Our model of group actions using handles captures all known techniques for efficient ID protocols from group actions. However, it is known that such a model fails to capture a number of standard generic techniques [38]. These standard generic techniques are typically used in symmetric key settings, as they involve operations like breaking strings into individual bits or XORs. Such operations break algebraic structure, seemingly negating the purpose of introducing algebraic tools in the first place. Nevertheless, such techniques could perhaps be employed in combination with algebraic tools to achieve more efficient signatures. As such, our impossibility does not fully rule out short signatures from group actions, but still represents a significant barrier.

On ID Protocols. At a technical level, our lower bound is for ID protocols. This is because it is known that signatures are *impossible* in Maurer’s generic group model [15], and the impossibility readily extends to our formalization of the black box group action model using handles. As such, any direct lower bound for signatures in our group action model would be completely meaningless.

Thus, any attempt at proving a lower bound for signatures is presented with a conundrum: work in Shoup’s version of group actions, where the long-standing open problem of signature length from random oracles presents a major barrier. *Or* work in Maurer’s model, where signatures are simply impossible.

While signatures do not exist in Maurer’s version of black box groups/group actions, ID protocols *do* exist. The transformation from ID protocol to signature, say via Fiat-Shamir, is then the only part of the signature that doesn’t work in Maurer’s model. This is because applying Fiat-Shamir requires hashing a group element/set element into a bit string. Such hashing is of course allowed in the standard model, but it is forbidden in Maurer’s since only the group (action) operation is allowed to be applied to elements. Fortunately, the most efficient signature schemes from groups and group actions are obtained by transforming ID protocols.

The Fiat-Shamir transformation is well-understood, both classically [2, 30] and quantumly [14, 24], and adds zero signature-length overhead over the underlying ID protocol³. But the length of any signature based on ID protocols is always lower-bounded by the ID protocol itself. Thus, our lower bound immediately applies to signatures based on ID protocols, which captures all-known practical group-action based signatures. Thus, our lower bound shows that a $\Omega(\lambda^2/\log_2 \lambda)$ signature length is inherent with current techniques.

Note that our lower bound is only for public coin protocols. This is inherent, as group actions give public key encryption, and any public key encryption scheme can be turned into an ID protocol, as follows [13]: the verifier encrypts a random message and sends the ciphertext, and the prover simply decrypts the ciphertext and sends the resulting message. The number of set elements in the

³ Other transforms such as Unruh’s [37] do require overhead.

protocol is just the number of set elements in a ciphertext, which in the case of group-action-based public key encryption, is just a constant. This protocol, however, is secret coin, as the verifier's message is a ciphertext that hides both the message and the encryption randomness. Such secret coin ID protocols are not amenable to Fiat-Shamir or related transformations, and there is no known direct way to turn them into signatures. Thus, our restriction to public coin protocols is justified by the ultimate goal of lower-bounding signatures.

1.3 Technical Overview

Existing Group Action-Based Signatures. The main group-action-based signatures are built from a public coin identification (ID) protocol, and then by converting the ID protocol into a signature. This conversion is typically Fiat-Shamir [18], but other transforms are possible [37]. For reasons explained above, we focus on analyzing the underlying ID protocol.

Throughout, we will focus on the number of set elements sent by the prover, which is a proxy for the total communication of the ID protocol. Note that when converting into a signature scheme, usually not all the terms of the ID protocol need to be sent explicitly, since they can be computed from the other terms for a valid signature. Nevertheless, the number of set elements remains linear in the total signature size.

The usual way to build an ID protocol from group actions, is the following adaptation of Schnorr's identification protocol [34] for plain groups:

- The public key contains two set elements x_0, x_1 such that $x_0 = g * x_1$. The secret key is a random $g \in G$.
- The prover first chooses a random $h \in G$, and sends $a = h * x_1$.
- The verifier replies with a random bit b .
- The prover then outputs $r = hg^{b-1}$.
- The verifier checks that $a = r * x_b$.

The ID protocol is easily seen to be zero knowledge. The protocol has (classical) soundness error $1/2$: if an adversary can break security with probability non-negligibly greater than $1/2$, then a standard rewinding argument shows that it can compute hg^{b-1} for both $b = 0$ and $b = 1$; dividing gives g , the discrete log between x_0 and x_1 , which is presumably hard to compute. On the other hand, it is trivial to break security with probability $1/2$: the prover simply guesses the bit b , and computes $a = r * x_b$ for a random r . Conditioned on the guess for b being correct, the transcript seen by the verifier will have the correct distribution.

To achieve better soundness, one can run the protocol many times, either sequentially or in parallel. To get soundness error $2^{-\lambda}$, one would need λ trials, requiring λ set elements to be sent from the prover.

One can do slightly better, at the cost of a somewhat larger public key. Abstracting an optimization of De Feo and Galbraith [9] (See [9], Sect. 4) to the setting of group actions, consider the following protocol:

- The public key contains P set elements x_1, \dots, x_P . The secret key is g_2, \dots, g_P such that $x_i = g_i * x_1$ for $i > 1$.

- The prover chooses a random $h \in G$, and sends $a = h * x_1$.
- The verifier replies with a random $c \in [P]$
- The prover then outputs $r = hg_c^{-1}$, where $g_1 = 1$.
- The verifier checks that $a = r * x_c$

The above protocol achieves soundness error $1/P$, without any additional set elements in the protocol, but at the cost of expanding the public key to P set elements. To achieve soundness error $2^{-\lambda}$, we can set $P = \lambda / \log \lambda$, and repeat the protocol P times. The result is public keys and protocol transcripts containing P set elements.

Generalizing both protocols, if we let P, N be the number of set elements in the public key and protocol transcript, and S the soundness error, both protocols above have $S = P^{-N}$.

We note that if one relaxes zero knowledge, then smaller soundness error is possible. For example, for security against direct attacks, the prover can just reveal g , and now soundness matches the hardness of computing discrete logarithms. For eavesdropping security where the attacker sees t transcripts, one can modify the large public key protocol above to have the prover simply reveal a discrete logarithm between x_1 and a random choice of x_i . While this latter scheme has noticeable soundness error, in both cases here the prover actually sends no set elements at all. Other strategies are possible to improve soundness in the bounded eavesdropping setting. Nevertheless, for schemes of this nature, it seems to always be the case that $t \leq P$.

Our Lower Bound. Our main result is that for eavesdropping security under t transcripts, for any desired polynomial poly :

$$S \geq (1 - P/t - 1/\text{poly}) \times P^{-N} \quad (1)$$

For unbounded transcripts, this shows that the $S = P^{-N}$ of the known group-action-based protocols is essentially tight. It also shows to get non-trivial soundness when the prover sends no elements at all requires the number of elements in the public key to be at least as large as the number of transcripts, matching intuition for such schemes.

Intuition. We now provide the intuition for our lower bound. Consider the collection of set elements seen by the verifier, which we will call V . V includes both set elements in the public key, as well as set elements sent by the prover and any set elements computed by the verifier. Now, consider a group action query by the verifier, such as $g * x$, resulting in output y . The verifier therefore knows the discrete logarithm between x and y . Since the protocol is public coin, this means the discrete logarithm is also revealed by the protocol transcript.

By looking at all such queries, we induce a graph structure on V , where we connect the input and output nodes of any query by the verifier. Since discrete logarithms compose, the verifier knows the discrete logarithm between any two connected nodes.

We can now assume, essentially without loss of generality, that no two public key nodes are in the same connected component. After all, if they were, then the protocol transcript reveals the discrete logarithm between these nodes. If the protocol were zero-knowledge, this would mean the discrete logarithm can be computed from publicly-available information. Even in the eavesdropping setting, it means the discrete logarithm can be computed from the transcripts provided to the adversary. In either case, this means that one of the two nodes was in some sense superfluous. We can make this precise, showing that if the ID protocol is secure even if the adversary sees sufficiently many transcripts, then we can compile the protocol into one where all public key components are in different connected components. This transformation slightly impacts correctness, and results in the P/t term in Eq. 1.

We then give an adversary for any scheme where the public key nodes are in different connected components. Essentially, whenever the adversary is required to send a set element y , it simply guesses which of the public key nodes x that y will be connected to, and generates y such that it knows the discrete logarithm between y and x . We show, essentially, that conditioned on the guess being correct for every node sent by the prover, our adversary can correctly simulate the protocol execution, and convince the verifier. The probability of guessing correctly at every step is exactly P^{-N} , where P is the number of public key elements, and N is the number of elements sent by the prover.

For technical reasons, the above does not quite work perfectly. Essentially, our simulation ensures that the graph seen by the verifier has an edge everywhere it should, but does not guarantee that the graph has no edges where it should not. But we observe that if there is a bad edge in the simulated graph, this connects two nodes that should not be connected. We are able to argue, roughly, that this means we can remove nodes from the graph, somewhat analogous to how we handled public key elements in the same connected component. As in that case, there is still some error in the simulation, though it can be made an arbitrary small polynomial. This results in the $1/\text{poly}$ term in Eq. 1.

Formalizing the above intuition is non-trivial. The main difficulty, analogous to all black box separations, is that the construction and adversary could completely ignore the group action and just run some standard-model short signature scheme such as Schnorr.

Following Impagliazzo-Rudich [19], we block such a construction by giving the adversary unlimited private computation and only bound the number of queries to the group action to a polynomial. This captures constructions whose only source of hardness is the group action.

With unlimited private computation, we can brute force any signature scheme that does not use the group action. The challenge comes in attacking schemes that are a combination of using the group action, but also using standard-model building blocks, as a naive brute force will result in exponentially many queries.

We formalize the above intuition through a sequence of protocol simplification steps, where we gradually restrict the prover and verifier, showing that the simplifications are without loss of generality. Eventually we reach a simplified

protocol where we can apply the intuition above and prove our lower bound. See Sect. 3 for details.

Extensions. In Sect. 4, we discuss a couple of extensions to our main lower bound. We first consider a generalized model where it is possible to directly sample set elements, without having to derive them from other elements. While no existing group-action-based signature utilizes such direct sampling, it is supported by elliptic curves and therefore important to consider. We show, with some key modifications to our main proof, that our lower bound applies in this model as well.

We also give a generalization of black box group actions, that we call black box *graph actions*. This captures many of the features of group actions, but eliminates the group structure on the acting set, instead viewing the action as a walk on a graph. This is how isogeny-based signatures tend to work anyway, and by generalizing to a less-structured object, we make our lower bound more general. Our lower bound does not use any particular features of the group structure, and trivially adapts to a graph action.

2 Preliminaries

Notation: We use $\lambda \in \mathbb{Z}$ to denote the security parameter. We use $x \leftarrow y$ to denote the assignment of the value of y to x . We write $x \xleftarrow{\$} S$ to denote sampling an element from the set S independently and uniformly at random. For a randomized algorithm \mathcal{A} we write $y \xleftarrow{\$} \mathcal{A}(x)$ to denote the random variable that is the output of $\mathcal{A}(x)$. We use $[n]$ for the set $\{1, \dots, n\}$. We denote vectors in bold font: $\mathbf{u} \in \mathbb{Z}_q^m$ is a vector of length m whose elements are each in \mathbb{Z}_q .

2.1 Group Actions

A group action consists of a (not necessarily abelian) group G , a set X , and a binary operation $* : G \times X \rightarrow X$ satisfying the following properties:

- **Identity:** If $e \in G$ is the identity element, then $e * x = x$ for any $x \in X$.
- **Compatibility:** For all $g, h \in G$ and $x \in X$, $(gh) * x = g * (h * x)$.

For applications to cryptography, we want the group action to have certain computationally intractible problems. A typical minimal hard problem is that of computing “discrete logarithms”: computing g from x and $g * x$.

Our Model of Black Box Group Actions. Here, we give our model of a black box group action. Our model is analogous to Maurer’s [26] model for generic groups, but adapted to group actions. In our case, we model the group itself as a standard-model object, but then the set elements are only provided via handles. In more detail, the following oracles are provided to all parties:

- $\text{Eq}(\langle x \rangle, \langle y \rangle)$ takes as input two handles for set elements x, y , and returns 1 if $x = y$ and 0 otherwise.
- $\text{Act}(g, \langle x \rangle)$ takes as input a group element g and a handle $\langle x \rangle$ to a set element, and returns a handle $\langle y \rangle$ for the set element $y = g * x$.

Additionally, all parties are provided with a handle $\langle x_0 \rangle$ to a starting set element x_0 . Each query incurs unit cost, and all computation outside of queries is zero cost. Algorithms are not allowed any computation on handles, except to pass them to other algorithms or send as inputs to the oracles Eq, Act . The only handles an algorithm can query to Eq, Act are those provided explicitly as input (including $\langle x_0 \rangle$), or provided as output of prior queries to Act . A probabilistic polynomial time algorithm is a probabilistic algorithm in this model whose total cost is bounded by a polynomial.

Remark 2.1. The above model assumes there is a single starting handle $\langle x_0 \rangle$, and the only way to derive additional set elements is to act on this handle. This is how existing isogeny-based identification protocols work. However, isogenies provide a bit more functionality: in particular, it is possible to sample directly into the set elements. This does not give the adversary any more power, since such directly sampled elements will be essentially random and unrelated to any other element. However, such sampling could potentially be used in protocol design.

We will not allow such sampling for the rest of this section, as it allows us to explain our main ideas in a simpler manner. In Sect. 4.1 we extend the black box group action model to capture such a functionality, and show that our impossibility also extends to this model.

Verification Oracle. We can augment our black box group action model with the following oracle:

- $\text{Ver}(g, \langle x \rangle, \langle y \rangle)$ which returns 1 if $g * x = y$ and 0 otherwise.

This oracle can readily be simulated as $\text{Eq}(\text{Act}(g, \langle x \rangle), \langle y \rangle)$, so including Ver does not change the model. However, this oracle will still be convenient for our proofs. Concretely, we will make crucial use of the following lemma:

Lemma 2.2. *Let A be a deterministic algorithm in the black box group action model that may take as input handles $\langle x_1 \rangle, \dots, \langle x_n \rangle$ and non-handle terms, and outputs k handles $\langle y_1 \rangle, \dots, \langle y_k \rangle$, as well as non-handle terms. Let q be the number of queries A makes. Then there is another algorithm A' with identical input/output behavior to A . However, A' is restricted in the following way:*

- It makes no queries to Eq .
- It makes at most $O(q)$ queries to Ver , which must all come before any Act query.
- After making its queries to Ver , it makes exactly k queries to Act in parallel to produce its handle outputs: $\langle y_1 \rangle = \text{Act}(g_1, \langle x_{i_1} \rangle), \dots, \langle y_k \rangle = \text{Act}(g_k, \langle x_{i_k} \rangle)$. After making the Act queries, A' is not allowed to make any queries to any oracle.

Lemma 2.2 allows us to reduce general algorithms to relatively simple forms, which will make analyzing them easier. Note that Lemma 2.2 applies also to randomized algorithms by considering the random coins as an input. Then A' will also get these same random coins. We now prove Lemma 2.2.

Proof. Consider a general algorithm A in the black box group action model, which makes arbitrary queries to **Eq** and **Act**. We construct A' as follows. We assume that integers and set elements are encoded such that they are disjoint. A' creates “dummy” handles $\langle 1 \rangle, \dots, \langle n \rangle$, which it feeds into A along with any non-handle inputs. These dummy handles will be stand-ins for the true handles $\langle x_1 \rangle, \dots, \langle x_n \rangle$ provided to A' . We will also create a table T containing tuples (j, g, i) , which correspond to the dummy handle $\langle j \rangle$ being a stand-in for the real handle $\langle g * x_i \rangle$. Therefore, T is initialized to contain the tuples $(i, 1, i)$ for $i = 1, \dots, n$. We will maintain that A only ever sees dummy handles.

A' simulates A on the dummy handles $\langle 1 \rangle, \dots, \langle n \rangle$ as well as any non-handle inputs to A' . However, A' will intercept all the queries A makes. On each query:

- If the query has the form **Act**($g, \langle j \rangle$) query, A' looks up an entry (j, g', i) in T , which will be guaranteed to exist. It will then add the entry $(j', g \cdot g', i)$ to T , where j' is one more than the number of entries in T so far. A' then replies with the dummy handle $\langle j' \rangle$. Note that the entry $(j, g', i) \in T$ means that $\langle j \rangle$ is a stand-in for $\langle g' * x_i \rangle$. Therefore, A expects the result of the query to be $\langle (g \cdot g') * x_i \rangle$, corresponding exactly to the newly added entry $(j', g \cdot g', i)$.
- If the query has the form **Eq**($\langle j_0 \rangle, \langle j_1 \rangle$), look up entries $(j_0, g_0, i_0), (j_1, g_1, i_1)$ in T , which are guaranteed to exist. Then it makes a query $b \leftarrow \text{Ver}(g_1^{-1} \cdot g_0, \langle x_{i_0} \rangle, \langle x_{i_1} \rangle)$ and replies with b . Note that since $\langle j_0 \rangle$ is a stand-in for $\langle g_0 * x_{i_0} \rangle$ and $\langle j_1 \rangle$ is a stand-in for $\langle g_1 * x_{i_1} \rangle$, we have equality if and only if $g_0 * x_{i_0} = g_1 * x_{i_1} \Leftrightarrow (g_1^{-1} \cdot g_0) * x_{i_0} = x_{i_1}$, which is exactly the result of the **Ver** query.

Finally, when it A outputs handles $\langle j_1 \rangle, \dots, \langle j_k \rangle$, A' will look up entries $(j_t, g_t, i_t) \in T$ for $t = 1, \dots, k$. It will then make a single round of **Act** queries $\langle y_t \rangle = \text{Act}(g_t, x_{i_t})$. Observe that $\langle j_t \rangle$ is exactly a stand-in for $\langle g_t * x_{i_t} \rangle = \langle y_t \rangle$. A' will output $\langle y_1 \rangle, \dots, \langle y_k \rangle$, as well as any non-handle outputs of A .

At every step, we therefore see that A' simply replaces the handles A sees with appropriate stand-ins, but correctly answers the **Eq** queries and produces the correct output handles and non-handle elements. Thus A' perfectly simulates the outputs of A . \square

We then define an abstract model for ID protocols that use a graph action.

2.2 ID Protocols Using a Group Action Oracle

Here, we define the abstract model for an ID protocol using a group action oracle. An ID protocol in the black box group action model consists of the following algorithms:

- $\text{Gen}()$, a probabilistic algorithm which makes a polynomial number of queries, and samples a public key/secret key pair (pk, sk) . We will always assume without loss of generality that sk is just the random coins used in $\text{Gen}()$. On the other hand, pk may contain a combination of both (handles to) set elements and non-set element terms.
- $\mathcal{P}(\text{pk}, \text{sk})$, a probabilistic *interactive* algorithm that makes a polynomial number of queries, which takes as input (pk, sk) , and interacts with a verifier through several rounds of interaction. In general, the prover's messages may contain any combination of handles to set elements and also non-set element terms.
- $\mathcal{V}(\text{pk})$, a probabilistic interactive algorithm that makes a polynomial number of queries, which takes as input pk , and interacts with the prover. In general, the verifier's messages may contain any combination of handles to set elements and also non-set element terms. At the end of the interaction, \mathcal{V} outputs a bit b .

We denote the interaction of \mathcal{P} and \mathcal{V} by $b \stackrel{\$}{\leftarrow} \mathcal{V}(\text{pk}) \iff \mathcal{P}(\text{pk}, \text{sk})$. The *transcript* of the interaction is the list T of all messages sent. As we are in the black box group action model, we bound the number of queries of each algorithm to polynomial, but do not otherwise bound the computation outside of the queries.

Definition 2.3. A protocol $\Pi = (\text{Gen}, \mathcal{P}, \mathcal{V})$ has completeness C if

$$\Pr[1 \stackrel{\$}{\leftarrow} \mathcal{V}(\text{pk}) \iff \mathcal{P}(\text{pk}, \text{sk})] \geq C ,$$

where the probability is over $(\text{pk}, \text{sk}) \stackrel{\$}{\leftarrow} \text{Gen}()$ and the random coins of \mathcal{P}, \mathcal{V} .

We do not define soundness, but instead define the *opposite* of soundness, since we are interested showing that protocols with too little communication are unsound:

Definition 2.4. A protocol $\Pi = (\text{Gen}, \mathcal{P}, \mathcal{V})$ is (t, S) -unsound if there exists an algorithm \mathcal{A} making polynomially many queries such that

$$\Pr[1 \stackrel{\$}{\leftarrow} \mathcal{V}(\text{pk}) \iff \mathcal{A}(\text{pk}, T_1, \dots, T_t)] \geq S ,$$

where T_1, \dots, T_t are t transcripts of independent trials of $\mathcal{V}(\text{pk}) \iff \mathcal{P}(\text{pk}, \text{sk})$. Here, the probability is over $(\text{pk}, \text{sk}) \stackrel{\$}{\leftarrow} \text{Gen}()$, the randomness of the transcripts T_i , and the random coins of \mathcal{A}, \mathcal{V} .

Definition 2.5. We say a protocol Π is public coin if \mathcal{V} 's random coins can be written as (c_1, \dots, c_k) such that the i th message of \mathcal{V} is c_i .

For a public coin protocol, we will equivalently think of \mathcal{V} as just being an algorithm which takes as input the transcript and outputs a bit b . The execution of the protocol itself simply chooses each message from the verifier uniformly at random.

Notation. We will be using the following notation for ID protocols throughout this paper:

C : the correctness probability t : number of transcripts given to the adversary
 S : the soundness error P : the number of set elements in the public key
 R : the number of rounds N : the number of set elements sent by the prover

We will be considering multiple ID protocols throughout this paper, which we distinguish by subscripts, e.g. Π_1, Π_2, \dots . In such cases, we will use the same subscripts for our notation: e.g. C_1, C_2, \dots for correctness probability, etc.

3 The Lower Bound

This section contains our main theorem, a lower bound on the communication of any secure group-action-based ID protocol.

3.1 The Main Theorem

Theorem 3.1. *If a public coin ID protocol Π in the black box group action model has completeness C , then for any polynomial t , the protocol is (t, S) -unsound, for $S \geq (C - P/t - 1/\text{poly}) \times P^{-N}$, where poly is any polynomial. In particular, if $S \leq 2^{-\lambda}$, $C \geq 0.99$ and $t \geq 2.05P$, then $N \geq (\lambda - 1)/\log_2 P$.*

In other words, if we want λ -bit security, we need the number of set elements sent by the prover to be at least $(\lambda - 1)/\log_2 P$. As each set element itself will generally be at least λ bits, and the number of public key elements is a polynomial, this means λ bits of security requires total prover communication size of $\Omega(\lambda^2/\log \lambda)$. This corresponds to the size of signatures once we apply Fiat-Shamir.

In the remainder of this section, we now prove this theorem using a sequence of protocol simplification steps.

3.2 Normal Form Protocols

Label the set elements of the public key $1, \dots, P$. Given a transcript T , we will then number the set elements in T as $P + 1, \dots, P + N$ in the order they appear in T . Let $V = [P + N]$. We will somewhat abuse notation and refer to $\{1, \dots, P\} \subseteq V$ as public key elements, and $\{P + 1, \dots, P + N\}$ as transcript elements.

Definition 3.2. *A public coin ID protocol is in normal form if the following are true:*

- Verification is deterministic conditioned on the transcript.
- Verification only queries Ver and not Act, Eq .

- The final message from the prover contains a list Q , where each entry in Q has the form (g, i, j, b) . Here, $i, j \in [P + N]$ index into the combined set elements of the public key and transcript, g is a group element, and b is a bit. Let x_i, x_j be the elements at position i and j , respectively. (g, i, j, b) corresponds to querying $\text{Ver}(g, \langle x_i \rangle, \langle x_j \rangle)$ and receiving outcome b .
- The verifier first makes verification queries corresponding to those in Q : for a tuple (g, i, j, b) , it queries $b' \leftarrow \text{Ver}(g, \langle x_i \rangle, \langle x_j \rangle)$. These are the only queries it makes. If any of the query responses are inconsistent with Q , that is if $b \neq b'$, the verifier immediately aborts and rejects.

Assuming all queries are consistent, the verifier is allowed arbitrary subsequent deterministic computation to decide whether to accept or reject, but it can make no additional queries.

Lemma 3.3. *If there is a public coin ID protocol Π in the group action model, then there is also a normal form ID protocol Π_1 such that $t_1 = t, C_1 = C, S_1 = S, N_1 = N, P_1 = P, R_1 = R + 2$.*

Proof. First, observe that we can trivially make any protocol have deterministic verification by adding to the end of the protocol a message from \mathcal{V} to \mathcal{P} containing the random coins of \mathcal{V} . We therefore assume deterministic verification. By Lemma 2.2, since verification outputs a bit (and therefore no handles), we can also assume the verifier only makes queries to Ver and not Eq, Act .

Now that verification is deterministic, let \mathcal{P}_1 be the new prover, which runs \mathcal{P} . Then, at the end of running \mathcal{P} , \mathcal{P}_1 runs the verifier for itself, to see exactly what queries the verifier will make, assembling the query list Q .

We now explain how to construct \mathcal{V}_1 . First, for each $(g, i, j, b) \in Q$, \mathcal{V}_1 makes the corresponding query to Ver , obtaining b' . If $b \neq b'$, then \mathcal{V}_1 immediately aborts and rejects.

If $b = b'$ for each $(g, i, j, b) \in Q$, then \mathcal{V}_1 runs \mathcal{V} on the first $r + 1$ messages of the transcript, except that it has to intercept all of the Ver queries \mathcal{V} makes, which correspond to an entry $(g, i, j, b) \in Q$, and answers the query with b .

It is straightforward that $\mathcal{V}_1 \iff \mathcal{P}_1$ exactly simulates the behavior of $\mathcal{V} \iff \mathcal{P}$, and so $C_1 = C$. For soundness, consider an adversary \mathcal{A}_1 that convinces \mathcal{V}_1 with probability S_1 . We construct an adversary \mathcal{A} that convinces \mathcal{V} with probability S . \mathcal{A} runs \mathcal{A}_1 , and just discards the query list Q that \mathcal{A}' outputs. If \mathcal{A}_1 wins, then it must be that all queries \mathcal{V}_1 (and hence \mathcal{V}) makes are consistent with Q , and also that \mathcal{V} accepts. In other words, \mathcal{V} accepts transcript T whenever \mathcal{V}_1 accepts transcript T_1 , where T is the same as T' but with the query list Q discarded. Hence $S \geq S_1$. \square

3.3 The Transcript Graph

Recall that $V = [P + N]$ indexes the combined set elements of the public key and transcript, with $[P]$ corresponding to the public key elements and $[P + 1, P + N]$ corresponding to the transcript elements.

Consider running the verifier \mathcal{V} . Any accepting Ver query by \mathcal{V} corresponds to an edge between nodes in V ; call this edge set of accepting queries E . Then

$G_T = (V, E)$ forms an undirected graph. G_T is the *transcript graph* of T . We note that verification may be randomized, yielding different transcript graphs each time. However, we will always assume a normal form protocol with deterministic verification, meaning that G_T is uniquely determined by the protocol transcript.

We say that a transcript graph is *valid* if there is no path between any two distinct public key elements. In other words, each public key element lies in a different connected component. Otherwise, a transcript graph is *invalid*.

3.4 Respecting Verifiers

Definition 3.4. A respecting verifier for a normal-form protocol is one that always rejects transcripts with invalid transcript graphs.

Lemma 3.5. If there is a public coin normal form ID protocol Π_1 in the group action model, then there is also a public coin normal form ID protocol Π_2 with a respecting verifier, such that $t_2 = 0, C_2 \geq C_1 - N_1/t_1, S_2 \leq S_1, N_2 = N_1, P_2 \leq P_1, R_2 = R_1$.

Proof. The intuition is that we use the provided protocol transcripts to compute the discrete logarithms between public key elements, and then use this information to represent certain public key elements in terms of others. This lets us remove such public key elements. If the next protocol run would have likely connected two public key elements together, then the previous runs would have also likely connected them anyway, meaning one of the elements would not have been in the public key in the first place.

In more detail, given $\Pi_1 = (\text{Gen}_1, \mathcal{P}_1, \mathcal{V}_1)$ for a public coin normal form ID protocol, we construct $\Pi_2 = (\text{Gen}_2, \mathcal{P}_2, \mathcal{V}_2)$ as follows.

$\text{Gen}_2()$: First run $(\text{pk}_1, \text{sk}_1) \xleftarrow{\$} \text{Gen}_1()$. Now run $\mathcal{P}_1(\text{pk}_1, \text{sk}_1) \iff \mathcal{V}_1(\text{pk}_1)$ for t_1 independent trails, collecting transcripts T_1, \dots, T_{t_1} . It then computes the transcript graphs $G_{T_1}, \dots, G_{T_{t_1}}$. Then for $i = 1, \dots, P_1$, it does the following:

- If the i -th public key set element $\langle x_i \rangle$ is connected to any previous public key set element $\langle x_j \rangle$ at position $j < i$ through any path of edges in $\cup_{\ell \in [t_1]} G_{T_\ell}$, take the minimal such j . Then use the queries in T_ℓ to determine the group element g such that $x_i = g * x_j$. Delete $\langle x_i \rangle$ from the public key, and replace it with the pair (j, g) . If there is no such path, then leave $\langle x_i \rangle$ as is.

Note that since j is minimal, in particular x_j is not connected to any x_ℓ for $\ell < j$. So if $\langle x_i \rangle$ is replaced with (j, g) , it must mean that $\langle x_j \rangle$ has not been deleted.

Then $\text{pk}_2 = \text{pk}_1$, except with all the deleted set elements replaced by the appropriate (j, g) . $\text{sk}_2 = \text{sk}_1$ ⁴.

⁴ Technically, we assumed sk was the random coins of Gen , and so our sk_2 should also include the random coins used to generate the T_i . However, this information will not be needed in the actual protocol, so we can think of sk_2 as being just sk_1 .

\mathcal{P}_2 : \mathcal{P}_2 runs \mathcal{P}_1 , except that any time \mathcal{P}_2 would need a deleted $\langle x_i \rangle$ from the public key, \mathcal{P}_2 re-computes it as $\langle x_i \rangle = \text{Act}(g, \langle x_j \rangle)$ for the appropriate (j, g) .

\mathcal{V}_2 : \mathcal{V}_2 runs \mathcal{V}_1 , except that any time \mathcal{V}_1 would needs a deleted $\langle x_i \rangle$ from the public key, \mathcal{V}_2 re-computes it as $\langle x_i \rangle = \text{Act}(g, \langle x_j \rangle)$ for the appropriate (j, g) . Moreover, at the end of the protocol \mathcal{V}_2 computes the transcript graph G_T , defined over the non-deleted elements in pk_2 , and automatically rejects if G_T is invalid.

Security. If \mathcal{V}_2 did not check the validity of G_T , then the interaction between \mathcal{P}_2 and \mathcal{V}_2 is identical to that of $\mathcal{P}_1, \mathcal{V}_1$, since each just re-computes the correct $\langle x_i \rangle$ as needed. Moreover, notice that computing pk_2 from pk_1 can be done by an adversary for $\mathcal{P}_1, \mathcal{V}_1$ using the t_1 transcripts provided to it in the passive security game. Adding a reject condition in \mathcal{V}_2 only decreases the adversary's success probability.

Correctness. In order to establish the correctness of the protocol, we just need to bound the probability G_T is invalid. Fix some $(\text{pk}_1, \text{sk}_1)$. For any transcript graph G_T , let G'_T be the induced graph with nodes in $[P_1]$, where there is an edge between two nodes in $[P_1]$ if and only if there is a path between those nodes in G_T . Let n_i be the number of connected components in $G'_i := \cup_{j \leq i} G'_{T_j}$, and $e_i = \mathbb{E}[n_i]$ be the expectation of n_i . Note that $n_0 = P_1$, $n_i \geq 0$ for all i , and $n_{i+1} \leq n_i$. Therefore, these (in)equalities hold in expectation.

Moreover, $i \mapsto e_i$ is convex, meaning $e_i - e_{i+1} \leq e_{i-1} - e_i$ for all i . To see this, let n'_i be the number of connected components in $G'_{T_{i-1}} \cup G'_{T_{i+1}}$. The difference relative to n_i is that we swap out G'_{T_i} for $G'_{T_{i+1}}$. Let $e'_i = \mathbb{E}[n'_i]$. Since G'_{T_i} and $G'_{T_{i+1}}$ come from the same distribution, we must have $e'_i = e_i$. Now let $r_i := n_{i-1} - n_i$ and $r'_i := n_{i-1} - n'_i$. This means $G'_{T_{i+1}}$ connects r_{i+1} pairs of the connected components of G'_i together, and r'_i pairs of connected components of G'_{i-1} . For every connection $G'_{T_{i+1}}$ makes between connected components of G'_i , there are corresponding connected components of G'_{i-1} that it also connects, since the connected components of G'_{i-1} is just a refinement of the connected components of G'_i . Thus $r'_i \geq r_{i+1}$, meaning $\mathbb{E}[r_i] = \mathbb{E}[r'_i] \geq \mathbb{E}[r_{i+1}]$. Hence $e_i - e_{i+1} \leq e_{i-1} - e_i$.

By the triangle inequality, this means $|e_{t_1+1} - e_{t_1}| \leq P_1/t_1$. In particular, $\Pr[n_{t_1+1} < n_{t_1}] < P_1/t_1$. But notice that $n_{t_1+1} = n_{t_1}$ corresponds to the transcript graph of $\mathcal{P}_2 \iff \mathcal{V}_2$ being valid. This is because pk_2 has exactly n_{t_1} public key elements remaining, one for each connected component in $\cup_{j \in [t_1]} G'_{T_j}$. Then any edge between remaining public key elements in pk_2 would have reduced the number of connected components, implying $n_{t_1+1} < n_{t_1}$.

Therefore, except with probability P_1/t_1 , the transcript graph for $\mathcal{P}_2 \iff \mathcal{V}_2$ is valid. This means \mathcal{V}_2 accepts with probability at least $C_2 \geq C_1 - P_1/t_1$. \square

3.5 Guessing Provers

A guessing prover has the following structure:

- The prover initially guesses a random partition W of V , such that each set in the partition contains exactly one public key element. In other words, for each transcript element in V , the prover chooses a random public key element to associate the transcript element to. The number of possible W is P^N .
- Recall by Lemma 2.2 that we can always assume the prover only queries Act on input set elements, and immediately outputs the result of the query as an set output. Consider such a query $\langle y \rangle = \text{Act}(g, \langle x \rangle)$. The prover guarantees that for any such query, $\langle y \rangle$ is in the same element of W as is $\langle x \rangle$.
- Let W' be the partition corresponding to the connected components of the final transcript graph G_T . Then if W' is not a refinement of W , the prover aborts and sends \perp for its last message (which the verifier would presumably reject if it were respecting).
- The prover never makes any queries to Eq .

Lemma 3.6. *If there is a public coin normal form ID protocol Π_2 with a respecting verifier in the group action model and $t_2 = 0$, then there is a public coin normal form ID protocol Π_3 with a respecting verifier and guessing prover such that $t_3 = 0, C_3 \geq C_2 \times P_2^{-N_2}, S_3 \leq S_2, N_3 = N_2, P_3 = P_2, R_3 = R_2$. In particular, conditioned on \mathcal{P}_3 not sending \perp , its correctness probability is at least C_2 .*

Proof. Recall that we assume \mathcal{P} is given the random coins used during setup. In particular, \mathcal{P} is able to compute the discrete logs between public key elements. This means it always knows the discrete logs between any group elements, and can therefore answer any Eq query by itself without making the query.

\mathcal{P}_3 simply runs \mathcal{P}_2 , except that it processes each query. Suppose \mathcal{P}_2 computes $\langle y \rangle = \text{Act}(g, \langle x_1 \rangle)$ for public key element $\langle x_1 \rangle$, while \mathcal{P}_3 needs to compute $\langle y \rangle = \text{Act}(g', \langle x_2 \rangle)$ for some other public key element $\langle x_2 \rangle$. Since \mathcal{P}_3 can compute the discrete log h such that $x_1 = h * x_2$, we can simply set $g' = gh$. Thus, \mathcal{P}_3 perfectly simulates the messages of \mathcal{P}_2 , until the last message. Importantly, all the previous messages are independent of W .

Whenever the prover convinces the verifier, since the verifier is respecting, the transcript graph is valid and must therefore have each public key element in a different connecting component. Let W' be the associated partition of the public key elements. Since W' is independent of W , we must have that $W' = W$ with probability $P_2^{-N_2}$. In particular, W' is a refinement of W with probability at least $P_2^{-N_2}$. Hence, the overall correctness probability is at least $C_3 \geq C_2 \times P_2^{-N_2}$. \square

3.6 Finishing the Proof of Theorem 3.1

We are now ready to finish the proof of Theorem 3.1, by showing the following

Lemma 3.7. *If there is a public coin normal form ID protocol Π_2 with a respecting verifier in the group action model, then for any polynomial poly , $S_2 \geq (C_2 - 1/\text{poly}) \times P_2^{-N_2}$*

Proof. We first invoke Lemma 3.6 to arrive at a protocol Π_3 with soundness error $S_3 \leq S_2$, and where the guessing prover \mathcal{P}_3 has correctness C_2 conditioned on it not sending \perp in the last message, for an overall correctness probability $C_3 \geq C_2 \times P_2^{-N_2}$.

We create a family of malicious provers $\mathcal{A}^{(i)}$, which are only given pk_3 , and attempt to simulate \mathcal{P}_3 . Let aux_3 be the non-set element part of pk_3 . $\mathcal{A}^{(i)}$ samples random coins for Gen_3 , conditioned on Gen_3 outputting aux_3 . By Lemma 2.2, the part of Gen_3 that outputs aux_3 maps bits to bits, and so makes no oracle queries at all. Therefore, sampling the random coins can be done without making any queries. Let $\text{sk}_3^{(1)}$ be the obtained public key.

In the case of $\mathcal{A}^{(1)}$, we now simply run $\mathcal{P}_3(\text{pk}_3, \text{sk}_3^{(1)})$. Let $q^{(1)}$ be the probability of convincing the verifier, conditioned on the final message of \mathcal{P}_3 not being \perp . When ignoring the set elements, $\text{sk}_3^{(1)}$ is identically distributed to sk_3 . Therefore, $\mathcal{P}_3(\text{pk}_3, \text{sk}_3^{(1)})$ is identically distributed to $\mathcal{P}_3(\text{pk}_3, \text{sk}_3)$, unless (1) the $\mathcal{P}_3(\text{pk}_3, \text{sk}_3^{(1)})$ does not send \perp , and also (2) there is a query in $(g, i, j, b) \in Q$ where $b \neq \text{Ver}(g, \langle x_i \rangle, \langle x_j \rangle)$. We note that if i, j are in the same part of the partition W , then this is guaranteed to never happen, since all elements within a partition element are generated as in the honest \mathcal{P}_3 . Also, recall that the verifier is respecting, meaning for i, j in different parts, it rejects if ever $b = 1$.

Therefore, the only “bad” case is when i, j are in different parts of the partition W , $\mathcal{A}^{(1)}$ generates $(g, i, j, b = 0)$, but actually $\text{Ver}(g, \langle x_i \rangle, \langle x_j \rangle) = 1$, meaning $g * x_i = x_j$. But observe that, in this case, the actual Ver query reveals the discrete log between two public key elements, which presumably should be hard. We will use this bad event to create a different adversary with a better success probability.

Concretely, let $\mathcal{A}^{(2)}$, generates $\text{sk}_3^{(1)}$, but then simulates for itself the interaction $\mathcal{V}_3(\text{pk}) \iff \mathcal{A}^{(1)}(\text{pk}_3, \text{sk}_3^{(1)})$ (choosing its own messages for \mathcal{V}_3), but conditioned on the final transcript graph G_T yielding a partition W' that is a refinement of W . Note that since \mathcal{P}_3 never makes queries to Eq and the transcript graph G_T does not contain set elements, determining whether the simulation has W' being a refinement of W can be computed without making any oracle queries at all (by Lemma 2.2). So even though this event is exponentially unlikely, conditioning on this event can be done with only a polynomial number of queries (namely the number of queries in the protocol). Let $p^{(1)}$ be the probability a discrete log is revealed. By our conditioning on \mathcal{P}_3 not sending \perp , we have that $(C_3 - q^{(1)}) \leq p^{(1)}$.

Then $\mathcal{A}^{(2)}$ chooses $\text{sk}_3^{(2)}$ from the same distribution as $\text{sk}_3^{(1)}$, *except* that if any discrete logs $g * x_i = x_j$ are revealed in the first step, it *also* conditions on Gen producing public key elements with these discrete log. As before, this conditional sampling can be done without making any queries. Now $\mathcal{A}^{(2)}$ simply runs $\mathcal{P}_3(\text{pk}_3, \text{sk}_3^{(2)})$. Let $q^{(1)}$ be the probability of convincing the verifier, conditioned on the final output of \mathcal{P}_3 not being \perp . Now by similar arguments as before, $\mathcal{P}_3(\text{pk}_3, \text{sk}_3^{(2)})$ is identically distributed to $\mathcal{P}_3(\text{pk}_3, \text{sk}_3)$, unless a “bad” case occurs, where Q contains $(g, i, j, b = 0)$ such that $\text{Ver}(g, \langle x_i \rangle, \langle x_j \rangle) = 1$.

Except here, the “bad” case must also reveal a “new” discrete log, meaning $g * x_i = x_j$ could not be derived from any discrete logs revealed in the first step. This is because if $g * x_i = x_j$ could be derived from the discrete logs in the first step, our conditioning on the discrete logs in the first step would have ensured that Q contained the correct value of b . Let $p^{(2)}$ be the probability that a new discrete log is revealed. By our conditioning, we have that $(C_3 - q^{(2)}) \leq p^{(2)}$.

We similarly define $\mathcal{A}^{(3)}, \mathcal{A}^{(4)}, \dots$. We have that $(C_3 - q^{(i)}) \leq p^{(i)}$.

Now, note that there can only be at most $P_3 - 1$ “new” discrete logs revealed across the various steps. This means that, for any u , $\sum_{i=1}^u p^{(i)} \leq P_3 - 1$. This in particular means that, for any u , there must be an $i \in [u]$ such that $p^{(i)} < P_3/u$. So for any desired polynomial error poly , there will be some $i \leq \text{poly} \times P_3$ such that $p^{(i)} < 1/\text{poly}$, in which case $q^{(i)} > C_3 - 1/\text{poly}$. In other words, $\mathcal{A}^{(i)}$, conditioned on not outputting \perp in the final message, convinced the verifier with probability at least $C_3 - 1/\text{poly}$. Then, since $\mathcal{A}^{(i)}$ outputs something other than \perp with probability at least $P_3^{-N_3}$, the overall soundness error of $\mathcal{A}^{(i)}$ is at least $S_3 \geq (C_3 - 1/\text{poly}) \times P_3^{-N_3}$.

It remains to show that $\mathcal{A}^{(i)}$ makes a polynomial number of queries. Indeed, the sampling of the various $\text{sk}_3^{(j)}$ requires no queries, and then $\mathcal{A}^{(i)}$ runs i executions of the protocol, each incurring a polynomial number of queries. Since i itself is polynomial, the total query count is polynomial. \square

4 Extensions

Here, we discuss a few possible different models for black box group actions, extending our model from Sect. 3.

4.1 Direct Sampling

We now consider a model which captures the following feature of isogeny-based group actions: the ability to directly sample into the set elements, without having to act on existing elements. Our model is identical to the model from Sect. 2, except that it provides an additional random oracle for sampling elements:

- $\text{Eq}(\langle x \rangle, \langle y \rangle)$ takes as input two handles for set elements x, y , and returns 1 if $x = y$ and 0 otherwise.
- $\text{Act}(g, \langle x \rangle)$ takes as input a group element g and a handle $\langle x \rangle$ to a set element, and returns a handle $\langle y \rangle$ for the set element $y = g * x$.
- $\text{Samp}(s)$ takes as input a string $s \in \{0, 1\}^\lambda$ and outputs $\langle L(s) \rangle$ where $L : \{0, 1\}^\lambda \rightarrow X$ is a uniform random function.

As before, each query incurs unit cost, and all computation outside of queries is zero cost. Algorithms are not allowed any computation on handles, except to pass them to other algorithms or send as inputs to the oracles Eq, Act . The only handles an algorithm can query to Eq, Act are those provided explicitly as input, or provided as output of prior queries to Act or Samp . Note that we do not explicitly provide an $\langle x_0 \rangle$ as it is redundant, given Samp .

We call this model the *extended* black box group action model. We now prove the following:

Theorem 4.1. *If a public coin ID protocol Π in the extended black box group action model has completeness C , then for any polynomial t , the protocol is (t, S) -unsound, for $S \geq (C - P/t - 1/\text{poly}) \times (P+1)^{-N}$, where poly is any polynomial. In particular, if $S \leq 2^{-\lambda}$, $C \geq 0.99$ and $t \geq 2.05P$, then $N \geq (\lambda-1)/\log_2(P+1)$.*

Note that the quantitative theorem statement is almost identical to that of Theorem 3.1, except that P^{-N} is replaced with $(P+1)^{-N}$. This slightly weaker bound is inconsequential for security.

Proof. The proof follows a very similar outline to the proof of Theorem 3.1, with a couple of key changes.

Normal Form Protocols. We first define a normal form protocol similar to Definition 3.2, but with some changes:

- Verification is deterministic conditioned on the transcript. This is identical to Definition 3.2.
- Verification only queries **Ver**, **Samp** and not **Act**, **Eq**. This is identical to Definition 3.2, except that we allow for **Samp** queries.
- The final message from the prover contains a list Q , where each entry in Q has either the form (g, i, j, b) or s . Here, (g, i, j, b) represents an **Act** query as in Definition 3.2. The new part are terms of the form s , which correspond to a query **Samp**(s).
- The verifier first makes queries corresponding to those in Q . These are the only queries it makes. If any of the query responses are inconsistent with Q , the verifier immediately aborts and rejects.

Assuming all queries are consistent, the verifier is allowed arbitrary subsequent deterministic computation to decide whether to accept or reject, but it can make no additional queries.

By an essentially identical proof to that of Lemma 3.3, we can conclude the following:

Lemma 4.2. *If there is a public coin ID protocol Π in the group action model, then there is also a normal form ID protocol Π_1 such that $t_1 = t, C_1 = C, S_1 = S, N_1 = N, P_1 = P, R_1 = R + 2$.*

The Transcript Graph. We define the transcript graph similarly to Sect. 3, except that we also include the results of any verifier queries to **Samp** as nodes in the graph. We connect nodes in this graph via accepting **Ver** queries as before.

We say that a transcript graph is *valid* if there is no path between any two public key elements, and also no path between a public key element and an **Samp** element. We include paths of length zero in our notion of paths, so every node has a path to itself. In other words, each public key element lies in a different connected component, and those connected components are distinct from any connected component containing an **Samp** element. Otherwise, a transcript graph is *invalid*.

Respecting Verifiers. As in Sect. 3, a respecting verifier for a normal-form protocol is one that rejects invalid transcript graphs, except we use our updated notion of invalid transcripts. We now state an updated version of Lemma 3.3 to work with extended group actions, which follows from an essentially identical argument.

Lemma 4.3. *If there is a public coin normal form ID protocol Π_1 in the extended group action model, then there is also a public coin normal form ID protocol Π_2 with a respecting verifier, such that $t_2 = 0, C_2 \geq C_1 - N_1/t_1, S_2 \leq S_1, N_2 = N_1, P_2 \leq P_1, R_2 = R_1$.*

Guessing Provers. A guessing prover has the following structure:

- The prover initially guesses a random partition W of V into $P + 1$ sets, P of which contain exactly one public key element, and the final set containing none. The difference from Sect. 3 is that we allow for this extra set containing no public key elements. The number of possible W is $(P + 1)^N$, slightly more than in Sect. 3 owing to the additional set.
- The prover only queries **Act** on input set elements or the result of a **Samp** query. It then immediately outputs the result of the **Act** query as an set output. Moreover, for any such query $\langle y \rangle = \text{Act}(g, \langle x \rangle)$, the prover guarantees that $\langle y \rangle$ and $\langle x \rangle$ are in the same element of W . This is the same as Sect. 3, except we allow the prover to derive its outputs also from **Samp** queries.
- Let W' be the partition corresponding to the connected components of the final transcript graph G_T . Then if W' is not a refinement of W , the prover aborts and sends \perp for its last message (which the verifier would presumably reject if it were respecting).
- The prover never makes any queries to **Eq**.

The following is proved via an almost identical proof to Lemma 3.6:

Lemma 4.4. *If there is a public coin normal form ID protocol Π_2 with a respecting verifier in the extended group action model and $t_2 = 0$, then there is a public coin normal form ID protocol Π_3 with a respecting verifier and guessing prover such that $t_3 = 0, C_3 \geq C_2 \times P_2^{-N_2}, S_3 \leq S_2, N_3 = N_2, P_3 = P_2, R_3 = R_2$. In particular, conditioned on \mathcal{P}_3 not sending \perp , its correctness probability is at least C_2 .*

Finishing the Proof. We now give an extension of Lemma 3.7, which finishes the proof of Theorem 4.1:

Lemma 4.5. *If there is a public coin normal form ID protocol Π_2 with a respecting verifier in the group action model, then for any polynomial poly , $S_2 \geq (C_2 - 1/\text{poly}) \times (P_2 + 1)^{-N_2}$*

The proof follows an almost identical argument to that of Lemma 3.7, leveraging Lemma 4.4. Putting Lemmas 4.2, 4.3, and 4.5 together gives Theorem 4.1. \square

4.2 Black Box Graph Actions

Here, we generalize the group structure of the black box group action to what we call a *graph* action. Instead of a group, there is a labelled directed graph $G = (X, E)$ whose nodes are the set X , satisfying the following properties:

- **Reversibility:** If there is an edge $(x, y) \in E$, then $(y, x) \in E$.
- **Composition:** If there is a path p from x to y , then the edge $(x, y) \in E$.
- **Unambiguous labels:** For any node x , all the outgoing edges from x have distinct labels. Likewise, all the incoming edges to x have distinct labels. There may be overlapping edges amongst between the incoming and outgoing edges.
- **Base labels:** There is a set S of labels, such that for every node $x \in X$ and every label $s \in S$, there is an incoming edge and an outgoing edge from x with label s .

In the case of a group action, the edge labels are group elements, and for all nodes x and group elements g , the edge $(x, g * x) \in E$ and has label g . Reversibility, composition, and unambiguous labels follow immediately from the basic properties of group actions.

Now the following oracles are provided to all parties:

- **Eq**($\langle x \rangle, \langle y \rangle$) takes as input two handles for set elements x, y , and returns 1 if $x = y$ and 0 otherwise.
- **Act**($\ell, \langle x \rangle$) takes as input a label ℓ and a handle $\langle x \rangle$ to a node. If there is an edge $(x, y) \in E$ with label ℓ , then output $(\ell', \langle y \rangle)$, where ℓ' is the label for $(y, x) \in E$. Otherwise output \perp .
- **Inv**($\ell, \langle x \rangle$) takes as input a label ℓ and a handle $\langle x \rangle$ to a node. If there is an edge $(y, x) \in E$ with label ℓ , then output $(\ell', \langle y \rangle)$, where ℓ' is the label for $(x, y) \in E$. Otherwise output \perp .
- **Comp**($\ell_1, \ell_2, \langle x \rangle$) takes as input labels ℓ_1, ℓ_2 and a handle $\langle x \rangle$ to a node. If there are edges $(x, y) \in E$ and $(y, z) \in E$ with labels ℓ_1, ℓ_2 respectively, then output ℓ_3 , the label for the edge (x, z) . Otherwise output \perp .

Like with the group action model, each query incurs unit cost, and all computation outside of queries is zero cost. Algorithms are not allowed any computation on handles, except to pass them to other algorithms or send as inputs to the oracles **Eq**, **Act**, **Inv**, **Comp**. A probabilistic polynomial time algorithm is a probabilistic algorithm in this model whose total cost is bounded by a polynomial. We can also consider extending the model to include an **Samp** which generates handles to random nodes.

By inspecting our proof of Theorem 3.1, we see that our lower bound also holds in the black box graph action model. The limitation of this model, however, is that for many graphs, there is trivially no security. Thus, while our impossibility for short signatures will apply for arbitrary graphs, in many cases the impossibility is uninteresting as there will be more trivial attacks.

In more detail, consider an adversary given a handle $\langle x \rangle$ to a node. The adversary can choose two arbitrary labels ℓ_1, ℓ_2 , and compute **Comp**($\ell_1, \ell_2, \langle x \rangle$),

resulting in a label ℓ_3 . Observe that ℓ_1, ℓ_2, ℓ_3 are given as *bit-strings*, as opposed to handles.

For a general graph structure, it may be that $\text{Comp}(\ell_1, \ell_2, \langle x \rangle) \neq \text{Comp}(\ell_1, \ell_2, \langle y \rangle)$ for different nodes x, y . Thus, ℓ_3 potentially tells us information about x . If the adversary can generate many such (ℓ_1, ℓ_2) pairs, then after a polynomial number of queries x may be uniquely determined by the list of $\ell_3 = \text{Comp}(\ell_1, \ell_2, \langle x \rangle)$ values. In such a case, the graph action trivially has no security: an adversary can de-reference $\langle x \rangle$ into x by making a polynomial number of queries to get a list of ℓ_3 values, and then brute force search for a node $x \in G$ with the given composition structure. This brute force search may require exponential computation, but since the query count is polynomial this would be considered an adversary in the black box graph action model.

Such a trivial insecurity does not contradict our lower bound, but would render it meaningless.

The obvious way out would be for the graph to have the property that $\text{Comp}(\ell_1, \ell_2, \langle x \rangle) = \text{Comp}(\ell_1, \ell_2, \langle y \rangle)$ for all x, y , or at least for equality to hold with overwhelming probability over random x, y . In other words, for any ℓ_1, ℓ_2 , there is a unique ℓ_3 such that $\text{Comp}(\ell_1, \ell_2, \langle x \rangle) = \ell_3$ for most x .

But in this case, if we define $\ell_1 \times \ell_2$ as the unique ℓ_3 , this gives us a group structure on the set of labels, and this group acts on the set X . Thus, it appears that to avoid trivial attacks, we actually imposed a group action structure, and thus reduce to the case of Sect. 3.

4.3 A Fully Idealized Graph Action

Here, we present a fully idealized graph action model, which allows for general graphs (not corresponding to group actions) without rendering the graph action model trivially insecure.

The idea is to protect edge labels behind handles, in addition to the nodes. This means that, even though $\text{Comp}(\ell_1, \ell_2, \langle x \rangle) \neq \text{Comp}(\ell_1, \ell_2, \langle y \rangle)$, the actual output of $\text{Comp}(\ell_1, \ell_2, \langle x \rangle)$ is a handle. Attempting to brute force search for x given the list of label handles is no longer possible without making exponentially many queries.

This model is incomparable to the previous graph action model and also the group action model: while it prevents the attacker from making use of the bit representation of edge labels, it also prevents the *construction* from making use of such labels. In much of the group action and isogeny literature, the protocols do not need the bit representation, and would work with such a fully idealized graph action model. But there are construction techniques that would make use of such a bit representation (see [38] for a discussion in the context of generic groups), and our fully idealized model would not allow for such techniques. Thus, while the model extends the graph structure, it limits constructions in other ways.

We now give the model. The graph $G = (X, E)$ is still defined in the same way, but we modify the oracles that are provided to the parties:

- $\text{Eq}(\langle x \rangle, \langle y \rangle)$ takes as input two handles for set elements x, y , and returns 1 if $x = y$ and 0 otherwise.

- $\text{Act}(\langle \ell \rangle, \langle x \rangle)$ takes as input a handle $\langle \ell \rangle$ to a label and a handle $\langle x \rangle$ to a node. If there is an edge $(x, y) \in E$ with label ℓ , then output $(\langle \ell' \rangle, \langle y \rangle)$, where ℓ' is the label for $(y, x) \in E$. Otherwise output \perp .
- $\text{Inv}(\langle \ell \rangle, \langle x \rangle)$ takes as input a handle $\langle \ell \rangle$ to a label and a handle $\langle x \rangle$ to a node. If there is an edge $(y, x) \in E$ with label ℓ , then output $(\langle \ell' \rangle, \langle y \rangle)$, where ℓ' is the label for $(x, y) \in E$. Otherwise output \perp .
- $\text{Comp}(\langle \ell_1 \rangle, \langle \ell_2 \rangle, \langle x \rangle)$ takes as input handles $\langle \ell_1 \rangle, \langle \ell_2 \rangle$ to labels and a handle $\langle x \rangle$ to a node. If there are edges $(x, y) \in E$ and $(y, z) \in E$ with labels ℓ_1, ℓ_2 respectively, then output $\langle \ell_3 \rangle$, the handle to the label for the edge (x, z) . Otherwise output \perp .

The following is a straightforward extension of Theorem 3.1:

Theorem 4.6. *If a public coin ID protocol Π in the fully idealized black box graph action model has completeness C , then for any polynomial t , the protocol is (t, S) -unsound, for $S \geq (C - P/t - 1/\text{poly}) \times P^{-N}$, where poly is any polynomial. In particular, if $S \leq 2^{-\lambda}$, $C \geq 0.99$ and $t \geq 2.05P$, then $N \geq (\lambda - 1)/\log_2 P$.*

References

1. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 411–439. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_14
2. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 1993, pp. 62–73. ACM Press (Nov 1993). <https://doi.org/10.1145/168588.168596>
3. 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. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_9
4. Brassard, G., Yung, M.: One-way group actions. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 94–107. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_7
5. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive, Report 2022/975 (2022). <https://eprint.iacr.org/2022/975>
6. Chen, L., Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography, vol. 12. US Department of Commerce, National Institute of Standards and Technology (2016)
7. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006). <https://eprint.iacr.org/2006/291>
8. Cozzo, D., Smart, N.P.: Sashimi: cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 169–186. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_10

9. De Feo, L., Galbraith, S.D.: SeaSign: compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 759–789. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_26
10. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: compact post-quantum signatures from quaternions and isogenies. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 64–93. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_3
11. De Feo, L., Leroux, A., Wesolowski, B.: New algorithms for the deuring correspondence: SQISign twice as fast. Cryptology ePrint Archive, Report 2022/234 (2022). <https://eprint.iacr.org/2022/234>
12. Decru, T., Panny, L., Vercauteren, F.: Faster SeaSign signatures through improved rejection sampling. In: Ding, J., Steinwandt, R. (eds.) PQCrypto 2019. LNCS, vol. 11505, pp. 271–285. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25510-7_15
13. Dodis, Y., Kiltz, E., Pietrzak, K., Wichs, D.: Message authentication, revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 355–374. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_22
14. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the fiat-shamir transformation in the quantum random-oracle model. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 356–383. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_13
15. Döttling, N., Hartmann, D., Hofheinz, D., Kiltz, E., Schäge, S., Ursu, B.: On the impossibility of purely algebraic signatures. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13044, pp. 317–349. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90456-2_11
16. El Kaafarani, A., Katsumata, S., Pintore, F.: Lossy CSI-FiSh: efficient signature scheme with tight reduction to decisional CSIDH-512. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12111, pp. 157–186. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45388-6_6
17. Feo, L.D., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptol. **8**(3), 209–247 (2014)
18. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
19. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: 21st ACM STOC, pp. 44–61. ACM Press (May 1989). <https://doi.org/10.1145/73007.73012>
20. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
21. Ji, Z., Qiao, Y., Song, F., Yun, A.: General linear group action on tensors: a candidate for post-quantum cryptography. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 251–281. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_11
22. Kuperberg, G.: A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. SIAM J. Comput. **35**(1), 170–188 (2005)
23. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In: Proceedings of TQC, vol. 22, pp. 20–34 (2013)

24. Liu, Q., Zhandry, M.: Revisiting post-quantum fiat-shamir. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 326–355. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_12
25. Maino, L., Martindale, C.: An attack on SIDH with arbitrary starting curve. Cryptology ePrint Archive, Report 2022/1026 (2022). <https://eprint.iacr.org/2022/1026>
26. Maurer, U.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (2005). https://doi.org/10.1007/11586821_1
27. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
28. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
29. Peikert, C.: He gives C-Sieves on the CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 463–492. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_16
30. Pointcheval, D., Stern, J.: Provably secure blind signature schemes. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 252–265. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0034852>
31. Regev, O.: A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space (2004)
32. Reingold, O., Trevisan, L., Vadhan, S.: Notions of reducibility between cryptographic primitives. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 1–20. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_1
33. Rostovtsev, A., Stolbunov, A.: Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145 (2006). <https://eprint.iacr.org/2006/145>
34. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
35. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th FOCS, pp. 124–134. IEEE Computer Society Press (Nov 1994). <https://doi.org/10.1109/SFCS.1994.365700>
36. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18
37. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 755–784. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_25
38. Zhandry, M.: To label, or not to label (in generic groups). In: Dodis, Y., Shrimpton, T., (eds.) CRYPTO 2022, Part III. LNCS, vol. 13509, pp. 66–96. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-15982-4_3
39. Zhandry, M., Zhang, C.: The relationship between idealized models under computationally bounded adversaries. Cryptology ePrint Archive, Report 2021/240 (2021). <https://eprint.iacr.org/2021/240>