The Cost of Statistical Security in Proofs for Repeated Squaring

Cody Freitag
Cornell Tech, New York, NY, USA

Ilan Komargodski
The Hebrew University, Jerusalem, Israel
NTT Research, Sunnyvale, CA, USA

Abstract

In recent years, the number of applications of the repeated squaring assumption has been growing rapidly. The assumption states that, given a group element x, an integer T, and an RSA modulus N, it is hard to compute $x^{2^T} \bmod N$ – or even decide whether $y \stackrel{?}{=} x^{2^T} \bmod N$ – in parallel time less than the trivial approach of simply computing T squares. This rise has been driven by efficient proof systems for repeated squaring, opening the door to more efficient constructions of verifiable delay functions, various secure computation primitives, and proof systems for more general languages.

In this work, we study the complexity of *statistically sound* proofs for the repeated squaring relation. Technically, we consider proofs where the prover sends at most $k \geq 0$ elements and the (probabilistic) verifier performs generic group operations over the group $\mathbb{Z}_N^\star$. As our main contribution, we show that for any (one-round) proof with a randomized verifier (i.e., an MA proof) the verifier either runs in parallel time $\Omega(T/(k+1))$ with high probability, or is able to factor N given the proof provided by the prover. This shows that either the prover essentially sends p, q such that $N = p \cdot q$ (which is infeasible or undesirable in most applications), or a variant of Pietrzak's proof of repeated squaring (ITCS 2019) has optimal verifier complexity $O(T/(k+1))$. In particular, it is impossible to obtain a statistically sound one-round proof of repeated squaring with efficiency on par with the computationally-sound protocol of Wesolowski (EUROCRYPT 2019), with a generic group verifier.

We further extend our one-round lower bound to a natural class of recursive interactive proofs for repeated squaring. For r-round recursive proofs where the prover is allowed to send k group elements per round, we show that the verifier either runs in parallel time $\Omega(T/(k+1)^r)$ with high probability, or is able to factor N given the proof transcript.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Cryptographic Proofs, Repeated Squaring, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ITC.2023.4

Related Version *Full Version*: `https://eprint.iacr.org/2022/766`

Funding *Cody Freitag*: Cody Freitag's work was partially done during an internship at NTT Research. He is also supported in part by the National Science Foundation Graduate Research Fellowship under Grant No. DGE–2139899, DARPA Award HR00110C0086, and AFOSR Award FA9550-18-1-0267. *Ilan Komargodski*: Ilan Komargodski is the incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University, supported in part by an Alon Young Faculty Fellowship, by a JPM Faculty Research Award, by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643).

1 Introduction

The repeated squaring (RS) assumption (first introduced by Rivest, Shamir, and Wagner [33]) states that for an RSA modulus N, a group element x, and a time bound T, it is hard to compute $x^{2^T} \bmod N$ – or even decide whether $y \stackrel{?}{=} x^{2^T} \bmod N$ – in parallel time less than the trivial approach of simply computing T squares. Under this assumption, the RS function is a candidate *sequential* function, meaning it cannot be sped up using parallel processors. This gives the ability to tune the time bound T so that computing the RS function requires a specified amount of wall-clock time, e.g. you can set T so that computing $x^{2^T} \bmod N$ takes at least 1 hour. This property, combined with the algebraic structure of the repeated squaring function, has led to many exciting applications.

Originally, Rivest, Shamir, and Wagner [33] used the RS function to construct time-lock puzzles. Time-lock puzzles provide a mechanism to send a message "to the future", by allowing a sender to quickly generate a puzzle with an underlying message that remains hidden for a specified amount of wall-clock time. These are possible because repeated squaring in RSA group has a natural trapdoor that allows the puzzle generator to evaluate the function quickly. Namely, given the factorization of N, one can reduce 2^T mod the order of the group to compute x^{2^T} efficiently. In this sense, time-lock puzzles effectively give a "fine-grained" variant of standard cryptographic commitments, where the hiding property only holds for some fixed amount of time specified by T.

More recently, Pietrzak [30] and Wesolowski [41] showed how to construct efficient (non-interactive) proofs for the RS relation. (In both works, this was obtained by first designing an interactive protocol for RS and then making it non-interactive via the Fiat-Shamir transform [12].) Such proofs for the RS relation have been the main driving force behind various applications of the repeated squaring function. For instance, they are used to construct *verifiable delay functions* (VDFs), first proposed by Boneh, Bonneau, Bünz, and Fisch [5], where the output of the function serves as a unique "proof-of-sequential-work" that can be efficiently verified with an associated non-interactive proof. Among other applications, Boneh et al [5] propose VDFs as a way to generate randomness for a trusted lottery or to construct resource-efficient blockchains (which has since been adopted by Chia [1]).

Since the initial works of [6, 30, 41], there have been many new proposed applications for proofs of repeated squaring: [7] construct accumulators, [11] construct randomness beacons, [10] construct polynomial commitments for succinct arguments, and [4] build off of [10] to construct time and space-efficient arguments. Furthermore, there has been much focus on understanding the efficiency and security of such proofs (see e.g. [30, 41, 6, 11, 34, 4, 19]) as well as the security of the sequentiality assumption underlying RS (see e.g. [36, 23, 37, 35]).

Proofs of repeated squaring. In this work, we are interested in proofs for the repeated squaring language, defined with respect to a multiplicative group of integers modulo N:

$$\mathcal{RS}_N = \left\{ (x, y, T) \mid y = x^{2^T} \bmod N \right\},$$

where x and y are two group elements and T is an integer.[1] A proof system for this language consists of a prover P and a (probabilistic) verifier V. The goal of the verifier is to decide, given an instance (x, y, T), whether it is in $\mathcal{RS}_N$ or not, and the prover sends the verifier a

[1] More generally, our results hold for the repeated squaring relation in any multiplicative group of unknown order. We focus on RSA groups for this introduction for simplicity of presentation.

proof string π to help in this task. We emphasize that we are mostly interested in the setting of non-interactive proofs in this work as they are most useful for applications, but later we also generalize the above and consider interactive protocols.

For a proof system to be meaningful, it must satisfy completeness and soundness. Completeness stipulates that an honestly generated proof will convince the verifier whenever $(x, y, T) \in \mathcal{RS}_N$, and soundness requires that, whenever $(x, y, T) \notin \mathcal{RS}_N$, there is no cheating proof π that will convince a verifier with noticeable probability. As stated, this is a statistical notion of soundness, asking whether there exist cheating proofs at all. We also consider proof systems that are only computationally sound – commonly known as *arguments* – where there may exist such cheating proofs, but we assume they are hard to find. Furthermore, computationally sound proofs come in many forms depending on the kind of assumptions they rely on.

There are several known proofs for $\mathcal{RS}_N$ (see Appendix B for an overview). For all of them, at least one of the following hold:

(A) The proof is only computationally sound. This is undesirable or even insufficient for several important applications; see below.

(B) The prover leaks the factorization to the verifier. This only allows N to be used once and is infeasible unless the prover can factor N.

(C) There is a tradeoff between proof size, $|\pi|$, and (parallel) running time of V, Time_V, where

$$|\pi| \cdot \mathsf{Time}_V \geq T.$$

That is, inefficiency is somewhat necessary: any improvement in communication complexity must necessarily cause an increase in computational complexity and vice versa.

All three of the above properties are undesirable for various reasons as we discussed above. So, in this work, we aim to understand whether having one of the above drawbacks is necessary. We do this by studying the cost of statistical soundness in $\mathcal{RS}_N$:

Can we construct a (statistically sound) non-interactive proof system for $\mathcal{RS}_N$ with low communication, an efficient verifier, and that doesn't leak the factorization of N?

On statistical soundness and tradeoffs between efficiency and security. Purely based on security and ignoring efficiency, it is clear that proofs with statistical soundness are strictly better than ones with only computational soundness. So perhaps in high-stakes applications (e.g. for blockchains where lots of money is at stake), having soundness rely on newer and untested mathematical/ computational assumptions may not be worth it. It's worth emphasizing, however, that the computationally sound, non-interactive proofs for $\mathcal{RS}_N$ rely not only on well-formulated computational assumptions, but potentially also on assumptions regarding the setup used to generate the RSA modulus N.

Wesolowski's argument [41], for instance, is completely broken if the prover knows the factorization of N. This is not the case for Pietrzak's non-interactive proof [30], but this already suffers an $O(\log T)$ multiplicative communication overhead in efficiency. Still, Pietrzak's protocol is potentially broken if N is not a product of safe primes. To fix this, Block et al. [4] give a non-interactive proof that works for any group and hence value of N, but results in an additional $O(\lambda)$ blowup in efficiency over Pietrzak's proof.[2] This protocol

[2] We mention that Hoffmann et al. [19] give a similar result to [4] that works in any group with improved efficiency by considering repeated qth powers for structured $q \gg 2$. Still, their protocol inherently cannot be made more efficient than the protocol of [30].

still relies on a random oracle to securely instantiate the FS heuristic though. Technically, the FS heuristic can be instantiated for [4] assuming LWE [3], but only at a further cost in efficiency for both the prover and the verifier with its own additional trusted setup. So it seems, no matter which computationally sound proof you choose, there is a complex combination of both computational and setup assumptions, and if one piece fails, the security of the entire system may be completely compromised. However, if the underlying proof is statistically sound, then this problem does not exist as it is *impossible* to generate accepting proofs for false statements (this is true even if $P = NP$ and factoring is easy).

Even if one is extremely confident in their computational assumptions, there are protocols based on proofs for $\mathcal{RS}_N$ that are completely broken if the wrong underlying protocol is used. Specifically, in the recent work of Freitag, Komargodski, Pass, and Sirkin [13], they use proofs for $\mathcal{RS}_N$ on top of a time-lock puzzle based on the RS function in order to construct publicly verifiable and non-malleable time-lock puzzles (on their way to building fair multi-party coin-flipping and auction protocols without trusted setup). In the context of time-lock puzzles, the party who generates the puzzle needs to know the factorization of N; this is actually a *feature* of time-lock puzzles, not a bug. However, this implies that they need the corresponding proof in their construction to be sound even if some party may know the factorization of N, so as pointed out above, Wesolowki's protocol will not suffice. They instead rely on Pietrzak's protocol, which is still plausibly secure when the factorization of N is leaked. But again, even though Wesolowski and Pietrzak's protocols are both "computationally secure", you cannot simply default to using the more practically efficient protocol of Wesolowski.

We highlight two important takeaways from the example of [13]:

- If using computationally sound rather than statistically sound proofs, protocol designers need to be very careful about the specific assumptions that the soundness of this proof relies on. This crucially includes the interplay between the setup assumptions and mathematical/ computational assumptions that are needed in the case of [13].
- In settings where security is required (or simply desired) even when the factorization of N may be known, current protocols start with a statistically sound interactive proof, and then compile it to a non-interactive argument using the FS heuristic. As such, we see it as an important goal to characterize the efficiency of general, statistically sound, interactive proofs for $\mathcal{RS}_N$, which first requires understanding the setting of statistically sound, non-interactive proofs.

Still, statistically secure protocols tend to be much less efficient than their computationally secure counterparts. As such, our overall goal is to try to formally characterize the exact tradeoffs between efficiency and security for proofs of $\mathcal{RS}_N$, which has led to many exciting practical and theoretical applications in recent years.

The complexity of RS (without proofs). Even ignoring the potential help from a prover, the complexity of the RS function – or deciding the $\mathcal{RS}_N$ language – was not well understood until the very recent works of [36, 23], even in generic models. Specifically, Rotem and Segev [36] show that computing the RS function or deciding $\mathcal{RS}_N$ in less than T parallel time implies a factoring algorithm for N, at least when restricted to generic-ring algorithms. Katz, Loss, and Xu [23] show a similar result for computing the RS function in the strong algebraic group model.[3]

[3] These are incomparable models. The generic-ring model allows for multiplication/ division/ addition/ subtraction/ equality queries, but require that queries are independent of the group elements representations. The strong algebraic group model only allows multiplication/ division queries, but allows these queries to be made in a way that depends on the group elements explicit bit representations.

1.1 Our Results

We make progress towards resolving the above-mentioned questions. Within a certain restricted model (the generic-group model relative to a hidden order group; see below), we prove results on the tradeoffs between the communication complexity and the verifier's complexity in a large class of proof systems for $\mathcal{RS}$. In particular, for the class of proof systems that we consider, any improvement over known ones would lead to a non-trivial factoring algorithm. Thus, assuming that factoring is hard, any improvement must either be outside of the restricted model or relax soundness to computational.

A bound for MA proof systems. We consider proof systems where the prover sends the verifier a single possibly long message, and then the verifier decides whether to accept or not by running a probabilistic polynomial time computation. This corresponds to the class MA (which generalizes NP by allowing the verifier to be probabilistic).

We briefly mention two statistically sound proofs for $\mathcal{RS}$. First, the prover can just send the factorization (p, q) where $N = p \cdot q$. The verifier can check that $N = p \cdot q$, compute the order of the group $\varphi(N)$, and then efficiently check that y equals $x^{2^T \bmod \varphi(N)} \bmod N$. The second is a sumcheck-style proof [27] that is a generalization of Pietrzak's protocol [30] due to [11]. Here, the prover sends k evenly spaced "midpoints" between x and T, which results in $k+1$ statements corresponding to $T/(k+1)$ squares. The verifier uses random exponents to combines these statements into a new statement $(x', y', T/(k+1))$ that it can check itself in time $T/(k+1)$.

We show that the above two protocols are essentially the best possible among all generic-group proofs. Specifically, we show that *either* we can factor composite numbers (matching the first protocol), or otherwise in any MA proof that includes $k \geq 0$ group elements, the verifier must run in parallel time at least $\Omega(T/(k+1))$ (matching the second protocol). Additionally, if neither of these hold, then the protocol must not be statistically sound – there must exist proofs for false statements, even if they may be computationally hard to find.

We prove our result by presenting an algorithm that uses any "too-good-to-be-true" generic-group MA proof to solve factoring in the plain model. To this end, we use Maurer's [28] generic-group algorithms abstraction and extend it to capture MA proofs. In our model, we restrict the verifier to be a generic-group algorithm (in Maurer's sense) that makes a bounded number of group multiplication and division queries[4], and we say that it accepts if it outputs the group's identity 1. Notice, for example, that this allows the verifier to compute two element g, h and accept if they are equal by outputting $g \cdot h^{-1}$. Furthermore, the verifier can perform ANDs of equality checks and accept if many pairs $(g_1, h_1), \ldots, (g_n, h_n)$ are equal (allowing parallel repetition). This can be done by sampling random exponents $r_1, \ldots, r_n \in [2^\lambda]$ and outputting $\prod_{i=1}^{n} (g_i \cdot h_i^{-1})^{r_i}$, a la the sumcheck-style technique used in [30]. Finally, we note that all efficient proofs specifically designed for $\mathcal{RS}_N$ fall into this generic model.

The prover, on the other hand, may still be an unbounded (not necessarily generic) algorithm whose proof consists of a bit string and a sequence of group elements. Note that *not* restricting the prover to be generic only makes our result applicable to larger classes of constructions, thereby making it stronger. Refer to Section 2 for the precise model

[4] Such algorithms are sometimes referred to as straight-line programs.

definition. We emphasize that even in this simplified one-round setting, it turns out to be highly non-trivial to prove our result in a way that captures the behaviors of arbitrary provers and verifiers; see Section 1.3 for an overview.

▶ **Theorem 1** (Simplified and Informal; see Theorem 4)**.** *For any generic-group MA proof system for $\mathcal{RS}_N$, if the prover sends $k \geq 0$ group elements and a string* st*, the verifier either runs in parallel time $\Omega(T/(k+1))$, or is able to factor N given* st*.*

In fact, we prove in Corollary 9 that the above holds for any hidden order group. In addition to RSA groups, this notably includes class groups of unknown order, which was suggested in the context of repeated squaring by Wesolowski [41] (see [9] for a more general survey on the use of class groups in cryptography). In the general case, we show that either the verifier runs in parallel time $\Omega(T/(k+1))$, or is able to compute (a non-zero multiple of) the order of the group given the string st output by the prover. However, by a variant of the Miller-Rabin primality test [29, 31], it is well known that this implies a factoring algorithm for N when working over the multiplicative group $\mathbb{Z}_N^\star$.

We note that if the prover is efficient, we can compute st ourselves. So, the existence of a verifier with $o(T/(k+1))$ parallel runtime implies a standard model factoring algorithm.

A bound for recursive interactive proofs. We extend our lower bound for MA proofs to a certain natural class of general (multi-round) interactive proofs (IPs). Specifically, we consider a class of *recursive IPs*, where in every round of communication, the prover attempts to prove a new instance of $\mathcal{RS}_N$, although with a different starting point x, a different endpoint y, and a different delay parameter T. This class of IPs captures many sumcheck-style proofs for $\mathcal{RS}_N$; see Appendix B for an overview. In particular, for a bound on the round complexity r and a communication bound k, the adaptation of Pietrzak's [30] protocol results with a recursive IP with total communication $k \cdot r$ and verifier running time $O(T/(k+1)^r)$. Here, we obtain an optimal tradeoff between the message complexity, the round complexity, and the verifier's parallel running time, at least when restricted to generic group verifiers.

▶ **Theorem 2** (Simplified and Informal)**.** *For any generic-group r-round recursive interactive proof system for $\mathcal{RS}_N$, if the prover sends k group elements per round and results in a transcript* tr*, the verifier either runs in parallel time $\Omega(T/(k+1)^r)$, or is able to factor N given* tr*.*

Future Directions and Open Problems

Our work leaves many exciting open problems. We mention some of them next:

1. We prove our result in the generic-group model where we only allow multiplication and division queries. It would be interesting to extend this to handle general equality queries or addition/ subtraction queries in the the generic-ring model [2, 21, 36].
2. Can we get a similar result to Theorem 2 for general (public-coin) IPs rather than just for "recursive" IPs?
3. In general, for what other languages can we say that sumcheck-style (e.g. see [8] and references therein) proofs are optimal (at least among a reasonable but restricted class of verifiers)?

Paper Organization

In Section 1.2, we give an overview of related work, and then in Section 1.3, we give a detailed overview of the main techniques in this work. In Section 2, we define the generic group model we use in this work in the context of proofs. Then in Section 3, we give our main result for MA proofs. We provide standard notation and preliminaries in Appendix A and a detailed overview of existing non-interactive proofs for the repeated squaring relation in Appendix B.

Due to space constraints, we refer the reader to the full version of the paper for more details regarding our result on recursive interactive proofs and for all proofs.

1.2 Related Work

Complexity of interactive proofs. Goldreich and Håstad [15] initiated the investigation of interactive proofs with bounded communication. They showed that if a language L has an interactive proof in which the total communication is bounded by $c(n)$ bits then $L \in \mathsf{BPTime}(2^{c(n)} \cdot \mathsf{poly}(\mathsf{n}))$. Further relations between the communication complexity of interactive proof for a language and its complement were shown by Goldreich, Vadhan, and Widgerson [16].

The IP=PSPACE result [27, 38] says that languages that can be verified in polynomial time are exactly those proofs that can be generated with polynomial space. In this interactive proof system, the honest prover runs in super-polynomial time (even for log-space languages); this is true even for the scaled down version which captures polynomially recognizable languages. Nevertheless, the "easy" side of this result says that every language with an interactive proof of c bits is decidable with c space [27, 38]. Therefore, languages that require a lot of space to decide cannot have super efficient interactive proof systems.

Computationally sound proof systems can recognize any language in NP while using only poly-logarithmic message complexity (assuming collision resistant hash functions) [24].

In the statistical setting, the first interactive proofs with an *efficient* prover were given by Goldwasser, Kalai, and Rothblumn [17]. They designed an interactive proof system where the honest prover is efficient and run in polynomial time. In their proof system the language is given by a log-space uniform Boolean circuit with depth d and input length n. Their verifier runs in time $n \cdot \mathsf{poly}(d, \log n)$, the communication complexity is $\mathsf{poly}(d, \log n)$, and the prover runs in time $\mathsf{poly}(n)$. This protocol is very useful for low-depth computations.

Reingold, Rothblum, and Rothblum [32] showed a different protocol which suits polynomial time and bounded-polynomial space computations. They give a constant round protocol for polynomial time and space $S = S(n)$ languages such that: the honest prover runs in polynomial time, the verifier is almost linear time, and the communication complexity is $O(S \cdot n^{\delta})$ for $\delta \in (0, 1)$. Applied on the repeated squaring language, (where $S = \mathsf{poly} \log n$) this protocol's communication roughly matches Pietrzak's [30] when adapted to run in constant rounds (in which case it also requires the transmission of n^{δ} group elements).

Generic models. The problem we consider can be placed in a long line of research on proving efficiency trade-offs for various primitives, in some restricted class of constructions usually termed "black-box" or "generic". Generic or black-box constructions have the benefit of being applicable to every instantiation of the underlying structure, irrespectively of the exact details of its description. For specific instances, this usually allows for cleaner and more efficient constructions. The interactive proofs for $\mathcal{RS}$ of Pietrzak [30] and Wesolowski [41] are generic.

Our work is the first to study the complexity of proofs for $\mathcal{RS}$ from a foundational perspective. The most relevant previous works study the ("generic") complexity of related cryptographic primitives or assumptions. Rotem and Segev [36] and Katz et al. [23] showed that any generic algorithm for repeated squaring which is faster-than-trivial can be used to solve factoring. The result of [36] rules out generic constructions in the generic-ring model introduced by Aggarwal and Maurer [2] (see also Jager and Schwenk [21]). The result of [23] rules out constructions in the strong algebraic group model (extending [14]) wherein the adversary may use the concrete representation of group elements to make its group queries. In another work, Rotem, Segev, and Shahaf [37] showed that hidden order groups are necessary for achieving "delay" functions, at least generically. The result of [37] rules out generic-group constructions in Maurer's model [28] (same as our proof).

On class groups. It is worth noting that class groups are an alternative candidate for a group of hidden order. In contrast to RSA groups, they only require a trusted setup consisting of an honestly generated random string. Since this setup is simpler and easy to generate, it is presumably less likely that someone may know a trapdoor (the order of the group) for class groups. However, while they can be used to construct VDFs, it is not known how to use them to get TLPs. See [9] for a general survey of the use of class groups in cryptography.

1.3 Technical Overview

Throughout this overview, we use $\lambda \in \mathbb{N}$ to refer to the security parameter and let N denote the RSA modulus, where N is a product of two random λ-bit primes. We use $\mathbb{Z}_N^\star$ to denote the multiplicative group of integers mod N. We consider interactive proof systems for the repeated squaring relation $\mathcal{RS}_N$, which we represent via the function $f_{N,T}(x) = x^{2^T} \bmod N$ for any time bound $T \in \mathbb{N}$. As a warm up, we will start by considering single-round, NP-style, proof systems where the verifier is a *deterministic*, generic group algorithm. We will later show how to deal with randomized verifiers, and additionally extend to the class of *recursive* interactive proofs.

Overview of generic group proof systems. A (non-interactive) proof system consists of two parties, the prover P and the verifier V. On input a group element $x \in \mathbb{Z}_N^\star$, P's goal is to convince V that another group element y is equal to $f_{N,T}(x) = x^{2^T} \bmod N$. P is allowed to send V up to k group elements $\pi_1, \ldots, \pi_k \in \mathbb{Z}_N^\star$ as well as a bit string $\mathsf{st} \in \{0,1\}^*$. Throughout the overview, we will always assume that P sends exactly k group elements as part of its proof. V processes this information and outputs 1 to accept that $y = x^{2^T} \bmod N$ or rejects otherwise. We require that the proof system satisfies the standard notions of completeness and soundness. Completeness says that if $y = x^{2^T} \bmod N$, then an honest prover P causes V to accept. We parameterize soundness by a parameter δ, which says that if $y \neq x^{2^T} \bmod N$, then no (potentially unbounded) cheating prover $P^\star$ can cause V to accept with probability more than δ.

We restrict the above model by requiring that V is a (straight-line) *generic group* verifier, whereas we still allow the prover to be unbounded and behave arbitrarily. Specifically, V takes as input the modulus N, the time bound T, the prover's string st as explicit inputs. However, V only has implicit access to the input group element x, the purported output y, and the proof elements $\pi_1, \ldots, \pi_k$ sent by P. Intuitively, this means that V is allowed to multiply and divide these elements arbitrarily, as long as it does so in a way that independent of their representation. We formalize this following Maurer's generic group model [28], which we outline in Section 2.

At the end of the day, we leverage the fact that V uses its explicit inputs[5] to effectively generate various exponents α, β, $\gamma_1, \ldots, \gamma_k$ such that its output is given by the group element corresponding to

$$V(N, T, \mathsf{st}, x, y, \pi_1, \ldots, \pi_k) = x^{\alpha} \cdot y^{\beta} \cdot \prod_{i=1}^{k} \pi_i^{\gamma_i} = g.$$

Furthermore, we can always run V with dummy elements $x, y, \pi_1, \ldots, \pi_k$ and compute the exponents $(\alpha, \beta, \gamma_1, \ldots, \gamma_k)$ by observing its group operations. We say that V accepts if the output group element g is equal to the multiplicative identity $1 \in \mathbb{Z}_N^{\star}$, and V rejects otherwise. While this convention may seem restrictive, as V doesn't even know whether it is accepting or rejecting, we claim that this is still very expressive as V can compute two different group elements g, h and then output $g \cdot h^{-1}$, which is 1 if and only if $g = h$. Most natural protocol for repeated squaring including [30, 41] fall into this category. Furthermore, the verifier can perform ANDs of equality checks and accept if many pairs $(g_1, h_1), \ldots, (g_n, h_n)$ are equal (allowing parallel repetition). This can be done by sampling random exponents $r_1, \ldots, r_n \in [2^{\lambda}]$ and outputting $\prod_{i=1}^{n} (g_i \cdot h_i^{-1})^{r_i}$, a la the sumcheck-style technique used in [30].

The complexity of deterministic (NP) proofs. As a warm-up, suppose that the verifier V is deterministic. This means that for every set of explicit inputs N, T, st that V receives, it generates the same exponents $(\alpha, \beta, \gamma_1, \ldots, \gamma_k)$. Given this knowledge, we want to characterize all possible strategies a cheating prover may use. So, say a cheating prover $P^{\star}$ wants to fool V on any $y = x^d \neq x^{2^T} \bmod N$. Effectively, $P^{\star}$ can only set each group element π_i to be equal to x^{z_i} for some value z_i.[6] Then, it follows that V accepts if

$$x^{\alpha} \cdot x^{d \cdot \beta} \cdot \prod_{i=1}^{k} x^{z_i \cdot \gamma_i} = 1.$$

However, since the base x is shared by all of the group elements, the above holds if

$$\alpha + d \cdot \beta + \sum_{i=1}^{k} z_i \cdot \gamma_i = 0 \bmod \mathsf{Carm}(N),$$

where $\mathsf{Carm}(N)$ is Carmichael totient function, which is defined as the minimal value c such that $g^c = 1 \in \mathbb{Z}_N^{\star}$ for all $g \in \mathbb{Z}_N^{\star}$.[7] But, as long as $\vec{\gamma} = (\gamma_1, \ldots, \gamma_k) \neq \vec{0} \bmod \mathsf{Carm}(N)$, it follows that $P^{\star}$ can simply solve for a solution to $z_1, \ldots, z_k$ in the equation above to generate a proof that will falsely convince V that $x^d = x^{2^T}$.[8]

[5] If we allowed V to also use the representation of the input group elements, this would correspond to the strong algebraic group model of [23].

[6] Note that this is not true in general since $\mathbb{Z}_N^{\star}$ is not cyclic and hence there are group elements not represented as x^c for some $c \in \mathbb{Z}$. However, we assume this in the overview for simplicity as it captures the main idea of the proof.

[7] We note that we can simply choose x to be a group element whose order attains the maximal value $\mathsf{Carm}(N)$. This is what allows us to switch to working over the exponent without loss of generality.

[8] We note that this style of attack works for Wesolowski's (computationally sound) proof of repeated squaring [41], which is an AM protocol. The adaptive root assumption essentially states that it is computationally infeasible to perform such an attack, leveraging the randomness sampled by the verifier before the prover sends its message.

Still, it may be the case that V simply ignores the proof elements $\pi_1, \ldots, \pi_k$ by setting $\gamma_1, \ldots, \gamma_k = 0$. In this case, we leverage the completeness of the proof system to conclude that either V is inefficient and runs in parallel time T, or V must be able to factor N. If $y = x^{2^T} \bmod N$ and $\gamma_1, \ldots, \gamma_k = 0$, then we know, by the above equation, that V accepts if

$$\alpha + 2^T \cdot \beta = 0 \bmod \mathsf{Carm}(N).$$

We consider two different cases, either (1) $\alpha + 2^T \cdot \beta = 0 \in \mathbb{Z}$ or (2) $\alpha + 2^T \cdot \beta = c \cdot \mathsf{Carm}(N)$ for some $c \neq 0 \in \mathbb{Z}$.

In case (2), this actually immediately implies a probabilistic factoring algorithm for N via a well known adaptation of the Miller-Rabin primality test (formally stated in Lemma 6). Since we can compute α and β, given the code of V and the prover's string st, and hence $\alpha + 2^T \cdot \beta = c \cdot \mathsf{Carm}(N)$, this implies a factoring algorithm in the standard model given st. If the prover P is efficient, then we can compute st by ourselves, so it implies a factoring algorithm for any N, without any auxiliary advice. We emphasize, however, that it may be the case that the explicit string st sent by P helps V to compute some value $\alpha = 2^T \bmod \mathsf{Carm}(N)$. For example, P could have just set st to be a representation of $\mathsf{Carm}(N)$, and V simply set $\alpha = 2^T \bmod \mathsf{Carm}(N)$ and $\beta = -1$. This is why the factoring algorithm must receive the proof string st as input in general.

We split case (1) into two further subcases, either (1A) $\beta = 0$ or (1B) $\beta \neq 0$. In case (1B) where $\beta \neq 0$, this implies that

$$2^T \leq 2^T \cdot |\beta| \leq |\alpha|.$$

But that implies that V must run in parallel time T to compute x^α since $|\alpha| \geq 2^T$.

In case (1A) where $\beta = 0$ and $\alpha + 2^T \cdot \beta = 0$, it must also be the case that $\alpha = 0$. However, we've already assumed that $\gamma_1, \ldots, \gamma_k = 0$, so this means that V just always outputs 1 and accepts! So clearly, (P, V) cannot be a valid proof system as V accepts any $y \neq x^{2^T} \bmod N$ with probability 1 in this case.

In summary, if (P, V) is a sound proof system where V is a *deterministic* generic group verifier, then either:

1. V must run in parallel time at least T, or
2. there is a standard model factoring algorithm for N given the code of V and the string st output by P.

Stated another way, if V runs in parallel time less than T, then V must be able to factor N (with the help of the prover via st).

Extending to randomized verifiers. The high level outline of the lower bound for randomized verifiers is actually very similar to the case of deterministic verifiers. However, allowing the verifier to use randomness to determine its exponents introduces many highly non-trivial challenges. The key distinction between deterministic and randomized verifiers is that randomized verifiers are allowed to choose their exponents as a function of their randomness, so the attack where a cheating prover simply solves a single equation to fool the verifier no longer works. Instead, the cheating prover needs to satisfy a random equation with better than δ probability in order to violate soundness. Still, we will show how we can use the verifier's exponents to factor, or argue that the verifier must have parallel running time greater than $T/(k+1)$ with high probability.

Throughout, we will consider a fixed set of explicit inputs N, T, and st received by the verifier. Then, for any random string $\rho \in \{0,1\}^\lambda$ sampled by the verifier, we use $\mathsf{coef}(\rho)$ to denote the exponents that V uses to compute its output. So, if

$$V(N, T, \mathsf{st}, x, y, \pi_1, \ldots, \pi_k; \rho) = x^\alpha \cdot y^\beta \cdot \prod_{i=1}^{k} \pi_i^{\gamma_i},$$

then we say that $\mathsf{coef}(\rho) = (\alpha, \beta, \gamma_1, \ldots, \gamma_k)$. We note that we refer to these exponents as "coefficients" as they will correspond to coefficients in a system of equations over the exponent, hence the notation $\mathsf{coef}(\rho)$.

Our main strategy is to sample many different values $\rho_1, \ldots, \rho_n$ such that $||\mathsf{coef}(\rho_i)||_{\max} \ll 2^{T/(k+1)}$ for each $i \in [n]$, where $||\cdot||_{\max}$ indicates the maximum absolute value in the coefficient vector. If this isn't possible, then that means that the verifier must run in parallel time at least $T/(k+1)$, and we are done. Otherwise, it remains to show that we can either use these coefficients to factor or show that (P, V) is not a valid proof system. For each randomness value ρ_i, let $\mathsf{coef}(\rho_i) = (\alpha_i, \beta_i, \gamma_{i,1}, \ldots, \gamma_{i,k})$ denote the corresponding coefficient vector for ρ_i. We combine all of these coefficients together in the following way. Let $\Gamma \in \mathbb{Z}^{n \times k}$ be the matrix consisting of all of the $\gamma_{i,j}$ values, and let $\vec{\alpha}, \vec{\beta} \in \mathbb{Z}^n$ be vectors of the α_i and β_i values. A key property we will leverage is that the system of equations $\Gamma \cdot \vec{z} = -\vec{\alpha} - d \cdot \vec{\beta} \bmod \mathsf{Carm}(N)$ has a solution for $d = 2^T$ by completeness, but does not have a solution for any $d \neq 2^T \bmod \mathsf{Carm}(N)$ by soundness (with high probability), which we explain next.

For simplicity, we will assume throughout this overview that the proof elements π_j potentially output by the prover are all equal to x^{z_j} for some $z_j \in \mathbb{Z}$. Then, for $y = x^{2^T}$ and all $i \in [n]$, completeness tells us that there must be a solution for $z_1, \ldots, z_k$ to the equation

$$\alpha_i + 2^T \cdot \beta_i + \sum_{j=1}^{k} \gamma_{i,j} \cdot z_j = 0 \bmod \mathsf{Carm}(N).$$

Since the prover's proof must work for all randomness values by completeness, we know that the prover's vector $\vec{z} = (z_1, \ldots, z_k)^\top$ actually satisfies

$$\Gamma \cdot \vec{z} = -\vec{\alpha} - 2^T \cdot \vec{\beta} \bmod \mathsf{Carm}(N).$$

However, for any $d \neq 2^T \bmod \mathsf{Carm}(N)$ corresponding to $x^d \neq x^{2^T}$, we use soundness to show that

$$\nexists \vec{z},\ \Gamma \cdot \vec{z} = -\vec{\alpha} - d \cdot \vec{\beta} \bmod \mathsf{Carm}(N),$$

as long as we sample enough vectors n. At a very high level, this follows since each newly sampled coefficient vector must restrict the space of solutions in a non-trivial way, since otherwise the same solution will work with good probability for many different choices of exponents. So we set n large enough such that, with high probability, the space of possible solutions for any $d \neq 2^T \bmod \mathsf{Carm}(N)$ is empty. The details of this argument are given in the full version of the paper.

Next, we prove a key technical lemma that allows us to relate whether or not a system of equations mod $\mathsf{Carm}(N)$ has a solution. Specifically, we show that there exists an efficiently computable matrix M that satisfies the following two properties:

1. If there exists a solution $\vec{z}$ such that $\Gamma \cdot \vec{z} = -\vec{\alpha} - d \cdot \vec{\beta} \bmod \mathsf{Carm}(N)$, then $M \cdot (-\vec{\alpha} - d \cdot \vec{\beta}) = \vec{0} \bmod \mathsf{Carm}(N)$.
2. If $M \cdot (-\vec{\alpha} - d \cdot \vec{\beta}) = \vec{0}$ over $\mathbb{Z}$, then there exists a solution $\vec{z}$ such that $\Gamma \cdot \vec{z} = (-\vec{\alpha} - d \cdot \vec{\beta})$ over $\mathbb{Z}$ (and hence $\bmod \mathsf{Carm}(N)$).

Furthermore, we show that $||M \cdot \vec{v}||_{\max} < 2^T$ when $||\vec{v}||_{\max}, ||\Gamma||_{\max} << 2^{T/(k+1)}$. When working over a field, such a result is well known by simply converting Γ into reduced row echelon form and the linear function M is closely related to the determinant of Γ. However, working over the integers mod $\mathsf{Carm}(N)$, this becomes much messier to work with. At a very high level, we show the lemma by first converting Γ to its Hermite normal form H, which is the integer counterpart to reduced row echelon form. We then augment the matrix H with the column $(-\vec{\alpha} - d \cdot \vec{\beta})$ and apply linear operations to zero out the last column to construct the matrix M. However, working over the integers, we must be careful to make sure that the values don't blow up in order to get our desired bound on $||M \cdot \vec{v}||_{\max}$. The full details for the proof of this technical lemma are provided in the full version of the paper.

Armed with our key technical lemma and the observations above, we are ready to complete the logic of our result, which follows the same high level structure as the deterministic case. Given M, we compute $\vec{v} = M \cdot (-\vec{\alpha} - 2^T \cdot \vec{\beta})$. By completeness, we know that there exists a vector $\vec{z}$ such that $\Gamma \cdot \vec{z} = (-\vec{\alpha} - d \cdot \vec{\beta}) \bmod \mathsf{Carm}(N)$, so by the technical lemma, we know that $\vec{v} = \vec{0} \bmod \mathsf{Carm}(N)$. We consider two different cases, either (1) $\vec{v} = \vec{0}$ over $\mathbb{Z}$ or (2) there exists an index i such that $\vec{v}_i = c \cdot \mathsf{Carm}(N)$ for $c \in \mathbb{Z}$. In case (2), we can factor given $\vec{v}_i$ using the variant of the Miller-Rabin primality test, so we are done.

For case (1), we use the fact that M is linear, so

$$\vec{v} = M \cdot (-\vec{\alpha} - 2^T \cdot \vec{\beta}) = -M \cdot \vec{\alpha} - 2^T \cdot M \cdot \vec{\beta} \bmod \mathsf{Carm}(N).$$

We consider two further subcases, either (1A) $M \cdot \vec{\beta} = \vec{0}$ over $\mathbb{Z}$ or (1B) there exists an index i such that $M_i \cdot \vec{\beta} \neq 0$. In case (1B), this implies that

$$2^T \leq 2^T \cdot |M_i \cdot \vec{\beta}| \leq |M_i \cdot \vec{\alpha}|,$$

but we show in our key technical lemma that $|M_i \cdot \vec{\alpha}| < 2^T$. So case (1B) cannot happen.

In case (1A) where $M \cdot \vec{\beta} = \vec{0}$, this actually implies that $M \cdot \vec{\alpha} = \vec{0}$ since we have already assumed that $\vec{v} = M \cdot (-\vec{\alpha} - 2^T \cdot \vec{\beta}) = \vec{0}$. But, this implies that $M \cdot (-\vec{\alpha} - d \cdot \vec{\beta}) = \vec{0}$ over $\mathbb{Z}$ for any $d \neq 2^T \bmod \mathsf{Carm}(N)$! So, by our key technical lemma, we conclude that there exists a solution over $\mathbb{Z}$, and hence mod $\mathsf{Carm}(N)$ for some $d \neq 2^T \bmod \mathsf{Carm}(N)$. However, we argued above that this cannot be the case by soundness (with high probability).

Combining the above, we've ruled out the possibility of case (1), so case (2) must hold, which implies we can factor with high probability. So, in summary, if (P, V) is a sound proof system where V is now a *randomized* generic group verifier and P sends at most k group elements in its proof, then either:

1. V runs in parallel time at least $T/(k+1)$ with high probability, or
2. there is a standard model factoring algorithm for N given the code of V and the string st output by P.

An alternative way to view this result is as follows. If V runs in parallel time less than $T/(k+1)$ with good probability, then either it must "know" a factorization of N to be able to reduce its exponents mod $\mathsf{Carm}(N)$, or there must be a cheating strategy that falsely convinces V on such randomness values. Hence, if you want both statistical security and an efficient verifier V, it must be the case that V can factor N.

Recursive interactive proofs. We next discuss how our result for one-round, MA-style, proofs extends to the class of recursive interactive proofs. First, we define what we mean by a r-round recursive interactive proof for the function $f_{N,T}(x) = x^{2^T} \bmod N$. In each round i, there is an input statement (x, y, T) claiming that $y = x^{2^T} \bmod N$. P starts the round by sending a string $\mathsf{st} \in \{0,1\}^*$ and up to k group elements $\pi_1, \ldots, \pi_k$. V then responds with a random string $\rho \leftarrow \{0,1\}^\lambda$. If i is the last round, V uses its randomness ρ and the message from P to decide whether or not $y = x^{2^T} \bmod N$. Otherwise, P and V both use a generic group algorithm A_i to compute a new statement (x', y', T') given the prover's message and the verifier's random coins, and they start a new independent (recursive) proof for this statement with one fewer round.

The overall running time of V is simply the running time of A_i in each round i, plus its final running time to compute its output at the end of the protocol. In addition to standard notions of completeness and soundness, we require that if (x, y, T) is valid at the beginning of the round, then (x', y', T') is also valid for the start of the next round. However, if (x, y, T) starts as invalid, so $y \neq x^{2^T} \bmod N$, then we require that (x', y', T') is invalid with probability at least $1 - \delta$.

Due to the recursive nature of this interactive proof, we are able to reduce to the one-round case to show that in each round T' cannot shrink too much relative to T, assuming A_i (and hence V) runs in low parallel time. If there exists a round i such that T' is much smaller than T, then we could construct a proof system $(\widehat{P}, \widehat{V})$ for $y = x^{2^T} \bmod N$ as follows. The prover $\widehat{P}$ sends whatever P would have sent in round i. Then, $\widehat{V}$ runs A_i to compute (x', y', T') and outputs $(x')^{2^{T'}} \cdot (y')^{-1}$. It follows that $\widehat{V}$ runs in time corresponding to the running time of A_i plus T', which is dominated by T'. By our result for one-round proofs, this means that T' must be at least $T/(k+1)$ with high probability, otherwise we can construct a factoring algorithm given the proof string st from P in round i. Hence, after $r-1$ rounds, the final time bound T' must be at least $T/(k+1)^{r-1}$ and V must run in parallel time at least $T/(k+1)^r$ to be a valid proof system.

In summary, if (P, V) is a *recursive*, generic group, r-round interactive proof for $f_{N,T}(x) = x^{2^T} \bmod N$, where the prover sends at most k group elements per round, then either:

1. V runs in parallel time at least $T/(k+1)^r$ with high probability, or
2. there is a standard model factoring algorithm for N given the code of V and the transcript generated by an honest prover P.

2 Generic Group Proof Systems

We next give the details for the generic group model we use in this work. Then we define proof systems where the verifier is restricted to generic group operations.

2.1 The Generic Group Model

In this work, we use Maurer's generic group model abstraction [28], following the related works of Aggarwal and Maurer [2] and Rotem and Segev [36]. We note that this is not the same as Shoup's random representation model [39]. See the work of Zhandry [42] for a detailed comparison between these two models.

Informally, a generic group algorithm is one that can perform arbitrary group operations as long as the operations performed are independent of the representation of the group elements. At a high level, we model this by giving the algorithm indirect access to its input group elements via pointers into a table, and each new multiplication or division adds a new element to the table and returns the corresponding pointer.

Formally, we consider the multiplicative group $\mathbb{Z}_N^\star$ in this work, where N is an RSA modulus in $\mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$ for some security parameter $\lambda \in \mathbb{N}$. A generic group algorithm A receives N as input as an explicit bit string and also receives access to a table Table via an oracle $\mathcal{O}$ that stores the group elements computed so far. Initially, Table contains the identity $v_0 = 1 \in \mathbb{Z}_N^\star$ at index 0, and all of the group elements $x_1, \ldots, x_k \in \mathbb{Z}_N^\star$ provided as input to A in indices $1, \ldots, k$. A can make queries to the oracle $\mathcal{O}$ via the following syntax:

- Mutliplication: On input $(i_1, i_2, j, \times)$, the oracle $\mathcal{O}$ checks that the values v_{i_1} and v_{i_2} at indices i_1 and i_2 in Table are non-empty and not $\perp$. If so, $\mathcal{O}$ computes $v_{i_1} \circ v_{i_2}$ and stores the result at index j in Table. Otherwise, $\mathcal{O}$ stores $\perp$ at index j.
- Division: On input $(i_1, i_2, j, \div)$, the oracle $\mathcal{O}$ additionally checks v_{i_2} is invertible. If so, $\mathcal{O}$ computes $v_{i_2}^{-1}$ and stores $v_i \times v_{i_2}^{-1}$ at index j in Table, if applicable. Otherwise, $\mathcal{O}$ stores $\perp$ at index j.

We note that Maurer's generic group model usually includes equality queries, which we do not handle in this work. An algorithm A that does not issue any equality queries is known as a *straight-line* algorithm, so for this reason, we state our formal results for straight-line generic group algorithms to avoid confusion. We note that generic-ring algorithms are defined similarly as above, but they also include addition and subtraction queries with essentially the same syntax.

For a group element g computed by A, we use $\widehat{g}$ to denote the pointer to the corresponding element g in the table Table. We abuse notation slightly and whenever we write that A receives a group element g as input, we mean that it receives a pointer $\widehat{g}$ to the element in the corresponding table Table.

We allow generic group algorithms to receive and output both "explicit" values, represented by bit strings, and "implicit" values indicating group elements, represented by pointers into Table. We can think of all of the explicit values as helping the generic algorithm decide how to invoke the oracle $\mathcal{O}$ to perform generic operations.

A randomized generic group algorithm also receives as input a string $\rho \in \{0,1\}^\lambda$ (we assume λ bits of randomness for simplicity, however this could be extended arbitrarily). For any input inp, We denote $A(1^\lambda, N, \mathsf{inp}; \rho)$ the randomized generic group algorithm with random tape ρ.

Measuring complexity. Let A be a generic group algorithm. We denote by $\mathsf{Time}_A(1^\lambda, N, \mathsf{inp}; \rho)$ the total running time of A on the given inputs with random tape ρ, where each oracle query costs a single unit of time. Additionally, we allow A to be a parallel algorithm. Following Rotem and Segev [36], we model parallel generic group algorithms A by allowing A to issue oracle queries in "rounds". In each round, A can issue any number of oracle queries to $\mathcal{O}$ in a single time step via multiple processors. We use $\mathsf{Width}_A(1^\lambda, N, \mathsf{inp}; \rho)$ to denote the maximum number of processors used by A at any time step and $\mathsf{ParTime}_A(1^\lambda, N, \vec{x}; \rho)$ to denote the number of sequential time steps that it takes for A to compute its output. Whenever we omit input/ randomness parameters from Time_A, Width_A, or $\mathsf{ParTime}_A$, we mean the worst case running time over an arbitrary choice of input parameters.

The behavior of generic group algorithms. Let $\lambda \in \mathbb{N}$ and $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$. Let A be a straight-line generic group algorithm such that $A(1^\lambda, N, \mathsf{st}, \vec{x}; \rho)$ takes as input an explicit string $\mathsf{st} \in \{0,1\}^*$ and group elements $\vec{x} = x_1, \ldots, x_k \in \mathbb{Z}_N^\star$ and outputs a group element g. As A is only allowed to perform generic operations, it follows that A's output is of the form $\prod_{i=1}^k x_i^{\gamma_i}$ for $\gamma_1, \ldots, \gamma_k \in \mathbb{Z}$. Furthermore, by running A, we can compute these coefficients by providing arbitrary pointers as input to A in place of $\vec{x}$. We use the notation

$\mathsf{coef}_{V,\lambda,N,\mathsf{st}}(\rho) = (\gamma_1, \ldots, \gamma_k)^\top$ to denote the coefficient vector of V on input ρ for security parameter λ, modulus N, and explicit string st. We note that the main distinction between our model and the strong algebraic group model of [23] is that they allow the coefficient vector to additionally depend on the bit representations of the input group elements.

Relating parallel running time to degree. Its easy to see that a straight-line generic group algorithm that computes $A(1^\lambda, N, \mathsf{st}, \vec{x}; \rho) = \prod_{i=1}^{k} x_i^{\gamma_i}$, where $\gamma_1, \ldots, \gamma_k$ are given by $\vec{\gamma} = \mathsf{coef}_{A,\lambda,N,\mathsf{st}}(\rho)$, must run in depth at least $\log ||\vec{\gamma}||_{\max}$. This can be shown by induction for $||\vec{\gamma}||_{\max}$ equal to 2^i for $i \geq 0$. If $||\vec{\gamma}||_{\max} = 2^0 = 1$, then it may be the case that A just immediately outputs a group element in 0 steps, satisfying the base case. Suppose that $||\vec{\gamma}||_{\max} = 2^i$. After $i-1$ steps, the maximum exponent in absolute value of any group element in Table is 2^{i-1} by assumption. So, in the next time step, A can issue a multiplication query multiplying two such elements together. However, this will result in an element with depth at most 2^i, as required. It follows that

$$\mathsf{ParTime}_A(1^\lambda, N, \mathsf{st}, \vec{x}; \rho) \geq \log ||\mathsf{coef}_{A,\lambda,N,\mathsf{st}}(\rho)||_{\max}$$

for all $\lambda \in \mathbb{N}$, $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$, string $\mathsf{st} \in \{0,1\}^*$, input elements $\vec{x}$, and random string $\rho \in \{0,1\}^\lambda$.

We additionally note that, even if we only require A to compute a high degree function with high probability and with pre-processing over a random input, then the same lower bound holds by the work of Rotem and Segev [36].

2.2 Proof Systems in the Generic Group Model

A proof system consists of two algorithms: the prover P and the verifier V. For a language L, P and V interact on common input x over potentially many rounds until V either accepts or rejects. In order to be non-trivial, the prover P must have some additional capabilities compared to the verifier V. For classical proof systems, the prover P is an unbounded algorithm while V is polynomially bounded. The two main properties of a proof system are completeness and soundness. Completeness stipulates that P convinces V on $x \in L$, and δ-soundness stipulates no cheating prover $P^\star$ can convince V on $x \notin L$ with probability better than δ.

We consider *generic group* proof systems for languages defined by a function f defined over a group $\mathbb{Z}_N^\star$ for $\lambda \in \mathbb{N}$ and $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$. For such proof systems, we restrict V to be a generic group algorithm that makes a bounded number of group multiplication and division queries, whereas P may still be an unbounded (not necessarily generic) algorithm that sends a bit string and group elements to V. So, for a function f, P and V receive an input a security parameter 1^λ, the group description N, an input group element x, and the output of the function $f(x)$ as common input. P sends a bit string $\mathsf{st} \in \{0,1\}^*$ and sequence of group elements $\pi_1, \ldots, \pi_k$ to V, which V receives access to via pointers into a table as a generic group algorithm. V then performs generic computations and outputs a pointer to a group element $\widehat{g}$ and "accepts" if the corresponding group element $g = 1$.

▶ **Definition 3** (Generic Group Proof Systems). *Let $\delta\colon \mathbb{N} \to [0,1]$ and $k\colon \mathbb{N} \to \mathbb{N}$. For any $\lambda \in \mathbb{N}$, $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$, let $f\colon \mathbb{Z}_N^\star \to \mathbb{Z}_N^\star$ be a function. We say that the pair (P, V) is a k-element generic group proof system for f with δ-soundness if V is a generic group algorithm, and for all $\lambda \in \mathbb{N}$, $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$, and $k = k(\lambda)$, the following hold:*

- *Completeness: For all* $x \in \mathbb{Z}_N^\star$*, let* $\mathsf{st}, \pi_1, \ldots, \pi_k$ *be the output of* $P(1^\lambda, N, x, f(x))$*, then it holds that*

$$V(1^\lambda, N, \mathsf{st}, x, f(x), \pi_1, \ldots, \pi_k) = 1.$$

- *Soundness: For all* $x \in \mathbb{Z}_N^\star$*,* $y \neq f(x)$*, and algorithms* $P^\star$ *such that* $P^\star(1^\lambda, N, x, y)$ *outputs a string* st *and group elements* $z_1, \ldots, z_k$*, it holds that*

$$\Pr_{\rho \leftarrow \{0,1\}^\lambda}\left[V(1^\lambda, N, \mathsf{st}, x, y, z_1, \ldots, z_k) = 1\right] \leq \delta(\lambda).$$

If the verifier V *is a straight-line algorithm, we say that* (P, V) *is a straight-line generic group proof system.*

3 One Round Proofs

In this section, we provide our main theorem. Let $\lambda \in \mathbb{N}$ and $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$. We show that if there is a k-element generic group proof system with a straight-line verifier that runs in parallel time less than $T/2(k+1)$ with probability ϵ, then there is a $\mathsf{poly}(1/\epsilon) \cdot \mathsf{Time}_V$ algorithm that factors N. We define some useful notation for the theorem first, and then provide a high level outline of the proof structure.

For each randomness ρ, let $\mathsf{coef}_{V,\lambda,N,\mathsf{st}}(\rho) = (\gamma_1, \ldots, \gamma_k, \alpha, \beta)^\top$ be the coefficients such that $V(1^\lambda, N, \mathsf{st}, x, y, z_1, \ldots, z_k)$ outputs $x^\alpha \cdot y^\beta \cdot \prod_{i=1}^k z_i^{\gamma_i}$. As V is a generic group algorithm, we can compute $\mathsf{coef}_{V,\lambda,N,\mathsf{st}}(\rho)$ by simply running $V(1^\lambda, N, \mathsf{st}, x, y, z_1, \ldots, z_k)$ for generic elements $x, y, z_1, \ldots, z_k$ and keep track of the operations of V. For notational convenience, when $V, \lambda, N, \mathsf{st}$ are clear from context, we simply write $\mathsf{coef}(\rho)$. We also define $\mathsf{dcoef}_{V,\lambda,N,\mathsf{st}}(\rho, d)$ to denote the vector $(\gamma_1, \ldots, \gamma_k, \alpha + d \cdot \beta)^\top$, where $(\gamma_1, \ldots, \gamma_k, \alpha, \beta)$ are given by $\mathsf{coef}(\rho)$, which will be useful in our analysis.

▶ **Theorem 4.** *Let* $\lambda \geq 2, T \in \mathbb{N}$*,* $k \colon \mathbb{N} \to \mathbb{N}$*,* $\delta, \epsilon \colon \mathbb{N} \to [0,1]$*,* $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$*, and* (P, V) *be a* k*-element straight-line generic-group proof system for the function* $f_{N,T}(x) = x^{2^T} \bmod N$ *with soundness error* δ*.*

Let $x \in \mathbb{Z}_N^\star$ *and* $(\mathsf{st}, \pi_1, \ldots, \pi_{k(\lambda)}) \in \mathrm{Supp}\left(P(1^\lambda, N, T, x, f_{N,T}(x))\right)$*. If*

$$\Pr_\rho\left[\mathsf{ParTime}_V(1^\lambda, N, T, \mathsf{st}) < \frac{T}{k(\lambda)+1} - \log(k(\lambda))\right] \geq \max(2\delta(\lambda), \epsilon(\lambda)),$$

then there exists a standard model probabilistic $\mathsf{poly}(\lambda, k(\lambda), T, 1/\epsilon(\lambda)) \cdot \mathsf{Time}_V(1^\lambda, N, \mathsf{st})$ *time algorithm* A *such that*

$$\Pr\left[p, q \leftarrow A\left(1^\lambda, N, k, T, \mathsf{st}, 1/\epsilon(\lambda)\right) : N = p \cdot q\right] \geq 1 - 2^{-\lambda}.$$

We refer the reader to the full version for the proof of the theorem.

References

1 Chia network. `https://chia.net/`. Accessed: 2022-10-05.

2 Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. *IEEE Trans. Inf. Theory*, 62(11):6251–6259, 2016.

3 Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D. Rothblum. Ppad is as hard as lwe and iterated squaring. In *TCC*, 2022.

4 Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In *Advances in Cryptology - CRYPTO*, pages 123–152, 2021.

5 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Advances in Cryptology - CRYPTO*, pages 757–788, 2018.

6 Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. *IACR Cryptol. ePrint Arch.*, page 712, 2018.

7 Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 561–586. Springer, 2019.

8 Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In *CRYPTO (1)*, volume 12825 of *Lecture Notes in Computer Science*, pages 742–773. Springer, 2021.

9 Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2001.

10 Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In *Advances in Cryptology - EUROCRYPT*, pages 677–706, 2020.

11 Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *Advances in Cryptology - EUROCRYPT*, pages 125–154, 2020.

12 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO*, pages 186–194, 1986.

13 Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In *Theory of Cryptography - 19th International Conference, TCC*, pages 447–479, 2021.

14 Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Advances in Cryptology - CRYPTO*, pages 33–62, 2018.

15 Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998.

16 Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Comput. Complex.*, 11(1-2):1–53, 2002.

17 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

18 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. URL: `http://www.jstor.org/stable/2282952`.

19 Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Practical statistically-sound proofs of exponentiation in any group. In *CRYPTO (2)*, 2022.

20 Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In *STOC*, pages 750–760. ACM, 2021.

21 Tibor Jager and Jörg Schwenk. On the analysis of cryptographic assumptions in the generic ring model. *J. Cryptol.*, 26(2):225–245, 2013.

22 Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 708–721, 2021.

23 Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography – TCC*, pages 390–413, 2020.

24 Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC*, pages 723–732, 1992.

25 Swastik Kopparty and Abhishek Bhrushundi. Lecture 3: Finding integer solutions to systems of linear equations, fall 2014. URL: `https://sites.math.rutgers.edu/~sk1233/courses/ANT-F14/lec3.pdf`.

26 Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to ppad-hardness and vdfs. In *Advances in Cryptology - CRYPTO*, pages 632–651, 2020.

27 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

28 Ueli M. Maurer. Abstract models of computation in cryptography. In *IMACC*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

29 Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976.

30 Krzysztof Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS*, pages 60:1–60:15, 2019.

31 Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

32 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM J. Comput.*, 50(3), 2021.

33 Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology. Laboratory for Computer Science, 1996.

34 Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In *TCC (3)*, volume 13044 of *Lecture Notes in Computer Science*, pages 382–414. Springer, 2021.

35 Lior Rotem. Revisiting the uber assumption in the algebraic group model: Fine-grained bounds in hidden-order groups and improved reductions in bilinear groups. In *ITC*, volume 230 of *LIPIcs*, pages 13:1–13:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

36 Lior Rotem and Gil Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In *Advances in Cryptology - CRYPTO*, pages 481–509, 2020.

37 Lior Rotem, Gil Segev, and Ido Shahaf. Generic-group delay functions require hidden-order groups. In *Advances in Cryptology - EUROCRYPT*, pages 155–180, 2020.

38 Adi Shamir. Ip=pspace. In *31st Annual Symposium on Foundations of Computer Science, FOCS*, pages 11–15, 1990.

39 Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

40 Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2006.

41 Benjamin Wesolowski. Efficient verifiable delay functions. *J. Cryptol.*, 33(4):2113–2147, 2020.

42 Mark Zhandry. To label, or not to label (in generic groups). *IACR Cryptol. ePrint Arch.*, page 226, 2022.

A Preliminaries

For any $n \in \mathbb{N}$, we use $[n] = \{1, \ldots, n\}$ to denote the set from 1 to n. For a distribution X, we denote by $x \leftarrow X$ the process of sampling a value x from the distribtion X. For a set $\mathcal{X}$, we use $x \leftarrow \mathcal{X}$ to denote the process of sampling a value x from the uniform distribution over $\mathcal{X}$. For a bit string $\mathsf{st} \in \{0,1\}^*$, we use $|\mathsf{st}|$ to denote the length of st. Throughout, we use $\lambda \in \mathbb{N}$ to denote the security parameter.

A.1 Number Theory

In this work, we consider the multiplicative group of integers mod N, denoted by $\mathbb{Z}_N^\star$, where N is a product of two primes. Specifically, for any $\lambda \in \mathbb{N}$, we let $\mathsf{ModGen}(1^\lambda)$ denote the

algorithm that samples two random primes p, q in the interval $[2^\lambda, 2^{\lambda+1})$ and outputs $N = p \cdot q$. The group is given by $\mathbb{Z}_N^\star = \{x \in [1, N) : \gcd(x, N) = 1\}$, and multiplication in the group corresponds to multiplication over $\mathbb{Z}$ mod N. When it is clear from context we are working in the group $\mathbb{Z}_N^\star$, we will omit mod N when discussing multiplication of group elements.

The main language we consider in this work is the repeated squaring relation, $\mathcal{RS}_N$, defined as follows

$$\mathcal{RS}_N = \left\{(x, y, T) \mid y = x^{2^T} \bmod N\right\}.$$

For a particular value of N and T, we represent this relation by the function $f_{N,T}(x) = x^{2^T} \bmod N$. It is widely believed that $f_{N,T}$ cannot be computed and $\mathcal{RS}_N$ cannot be decided in depth less than T even with $\mathsf{poly}(\lambda, T)$ parallel processors. We focus on the proof complexity of this language in this work.

For any $a, b \in \mathbb{Z}$, we use $\gcd(a, b)$ and $\mathrm{lcm}(a, b)$ to denote the greatest common divisor and least common multiple of a and b, respectively. Specifically, $\gcd(a, b)$ is the maximal $c \in \mathbb{N}$ such that c divides a and b, and lcm is the minimal $c \in \mathbb{N}$ such that a and b both divide c. Let $a, b \in \mathbb{Z}$, then there always exist integers c, d such that $c \cdot a + d \cdot b = \gcd(a, b)$. c and d are known as Bezout coefficients for a and b. While Bezout coefficients may not be unique, we note that there always exist bezout coefficients such that $|c|, |d| \leq \max(|a|, |b|)$, and these are the coefficients given by the standard euclidean algorithm.

We denote by $\varphi(N) = |\mathbb{Z}_N^\star|$, known as the Euler totient function of N, and $\mathsf{Carm}(N) = \min\{a \in \mathbb{N} : \forall g \in \mathbb{Z}_N^\star, g^a = 1\}$, known as the Carmichael totient function. For $\lambda \in \mathbb{N}$ and $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$ such that $N = p \cdot q$, it holds that

$$\varphi(N) = (p-1) \cdot (q-1), \text{ and } \mathsf{Carm}(N) = \mathrm{lcm}(p-1, q-1).$$

For a specific element $g \in \mathbb{Z}_N^\star$, we define the order of g, $\mathsf{ord}(g)$, to be the minimum $c \in \mathbb{N}$ such that $g^c = 1 \in \mathbb{Z}_N^\star$.

In this work, we use the fact that for $N = p \cdot q$, $\mathbb{Z}_N^\star \cong \mathbb{Z}_p^\star \times \mathbb{Z}_q^\star$, where $\mathbb{Z}_p^\star$ and $\mathbb{Z}_q^\star$ are each cyclic groups of order $\varphi(p) = p - 1$ and $\varphi(q) = q - 1$, respectively. Let g_p and g_q be generators for the corresponding subgroups. Then, we can write any group element $h \in \mathbb{Z}_N^\star$ in the form $h = g_p^a \cdot g_q^b$ for some $a, b \in \mathbb{N}$. For convenience of notation, we will use $h|_p$ to denote the p "component" of h and $h|_q$ to denote the q component, so $a = h|_p$ and $b = h|_q$ above.

In order to translate between results mod a composite number Φ and its solutions mod its prime power divisors, we make use of the Chinese remainder theorem (CRT). We use the following version of CRT.

▶ **Lemma 5.** *Let $k \in \mathbb{N}$, $n_1, \ldots, n_k, a_1, \ldots, a_k \in \mathbb{N}$. Then, the set of equations*

$$x = a_i \bmod n_i$$

has a solution over $\mathbb{Z}$ if and only if for all $i, j \in [k]$, $a_i = a_j \bmod \gcd(n_i, n_j)$. Moreover, any two solutions x_1, x_2 satisfy $x_1 = x_2 \bmod \mathrm{lcm}(n_1, \ldots, n_k)$.

The following lemma, based on the Miller-Rabin primality test [29, 31], gives a probabilistic factoring algorithm given any non-zero multiple of $\mathsf{Carm}(N)$. For the proof of the lemma and further discussion, we refer the reader to Section 10.4 of Shoup [40].

▶ **Lemma 6** (Factoring Lemma)**.** *Let $\lambda \in \mathbb{N}$, $N \in \mathrm{Supp}\left(\mathsf{ModGen}(1^\lambda)\right)$, and $m = c \cdot \mathsf{Carm}(N)$ for $c \in \mathbb{Z}$ such that $c \neq 0$. For any $\delta \colon \mathbb{N} \to [0, 1]$, there exists a probabilistic algorithm A that runs in $\mathsf{poly}(\lambda, \log(1/\delta(\lambda)))$ time such that*

$$\Pr\left[p, q \leftarrow A(1^\lambda, N, m) : N = p \cdot q\right] \geq 1 - \delta(\lambda).$$

A.2 Linear Algebra

Let M be a matrix in $\mathbb{Z}^{m\times\ell}$. For $i\in[m]$, $j\in[\ell]$, we use M_i to denote the ith row and $M_{i,j}$ to denote the element in the ith row and jth column. We use $M^\top$ to denote the transpose of a matrix. We treat vectors $\vec{v}\in\mathbb{Z}^n$ as column vectors, so implicitly of the form $\vec{v}\in\mathbb{Z}^{n\times 1}$. To take the dot product of two vectors $\vec{v},\vec{u}$, we write $\vec{v}^\top\cdot\vec{u}$. If $v\in\mathbb{Z}^{m\times 1}$ is a vector, we simply write v_i to denote the ith component. We write $||M||_{\max}=\max_{i\in[m],j\in[\ell]}|M_{i,j}|$ to denote the largest element in absolute value in the matrix M. For a matrix $M^{(1)}\in\mathbb{Z}^{m\times\ell_1}$ and a matrix $M^{(2)}\in\mathbb{Z}^{m\times\ell_2}$, we write $M'=(M^{(1)}|M^{(2)})$ to denote the augmented matrix which appends $M^{(2)}$ to the right of $M^{(1)}$ to get the matrix $M'\in\mathbb{Z}^{m\times(\ell_1+\ell_2)}$.

For any composite Φ, let $\mathbb{Z}_\Phi$ be the ring of integers mod Φ. We say that a function $f\colon\mathbb{Z}_\Phi^n\to\mathbb{Z}_\Phi^n$ is linear if for any vectors $\vec{g},\vec{h}\in\mathbb{Z}_\Phi$ and $a,b\in\mathbb{Z}$, it satisfies $f(a\cdot\vec{g}+b\cdot\vec{h})=a\cdot f(\vec{g})+b\cdot f(\vec{h})$. We say that a function f is affine if there exists some matrix M such that $f(\vec{g})=M\cdot\vec{g}'$ where $\vec{g}'$ is equal to $\vec{g}$ appended by 1. In particular, this means that f is a linear function shifted by a constant.

Let $\mathsf{Perm}(n)$ denote the set of all permutations over $[n]$. For a permutation $\sigma\in\mathsf{Perm}(n)$, we write $\mathsf{sign}(\sigma)$ to denote the sign of σ, i.e. 1 if there are an even number transpositions from the identity to σ, and -1 otherwise. For a square matrix M, the determinant of M is given by $\det(M)=\sum_{\sigma\in\mathsf{Perm}(n)}\mathsf{sign}(\sigma)\cdot\prod_{i=1}^n M_{i,\sigma(i)}$. It follows by definition of the determinant that $\det(M)\le n!\cdot||M||_{\max}^n$. We say that an integer matrix $U\in\mathbb{Z}^{m\times m}$ is unimodular if $\det(U)\in\{+1,-1\}$.

Let $\vec{v}^{(1)},\ldots,\vec{v}^{(n)}\in\mathbb{Z}^m$ be a set of vectors. This determines a lattice

$$\mathcal{L}=\mathcal{L}\left(\vec{v}^{(1)},\ldots,\vec{v}^{(n)}\right)=\left\{\sum_{i=1}^m c_i\cdot\vec{v}^{(i)}:c_1,\ldots,c_m\in\mathbb{Z}\right\}$$

of points spanned by these vectors. For a lattice $\mathcal{L}$, we refer to a basis of the lattice as a set of vectors $\vec{b}^{(1)},\ldots,\vec{b}^{(m)}$, often written in matrix matrix $B=(\vec{b}^{(1)}|\ldots|\vec{b}^{(m)})$, that are linearly independent over $\mathbb{R}$ and $\mathcal{L}=\mathcal{L}(B)$. A lattice is unique up to multiplication of B by a unimodular matrix U, so when the basis is clear from context, we refer simply to the lattice $\mathcal{L}$. The determinant of a lattice $\det(\mathcal{L})$ is defined to be the volume of the parallelepiped formed by a set of basis vectors over $\mathbb{R}^m$.

We next define the Hermite normal form (HNF) of an integer matrix $M\in\mathbb{Z}^{m\times n}$. We use the notion of column-style HNF, defined via right multiplication by a unimodular matrix, in contrast to row-style HNF.

▶ **Definition 7** (Hermite Normal Form). *A matrix $H\in\mathbb{Z}^{m\times n}$ is in Hermite normal form if the following hold:*

1. *Lower triangular: For some $h\le n$, there exists a sequence $1\le i_1<i_2<\ldots<i_h\le n$ such that $H_{i,j}\neq 0\Rightarrow i>i_j$.*
2. *Row-reduced: For all $k\le j\le n$, $0\le H_{i_j,k}\le H_{i_j,j}$.*

We additionally use the fact that the HNF of a matrix $M\in\mathbb{Z}^{m\times n}$ has entries bounded by $||M||_{\max}^n$. See [25] for a proof of this claim.

When working over a field $\mathbb{F}$, such as the integers mod a prime p or the rationals $\mathbb{Q}$, we can define standard notions like span and rank. The span of a set over vectors over an n dimensional vector space over a field $\mathbb{F}$ is defined as the set of all linear combinations of the vectors, with coefficients from the field $\mathbb{F}$. When clear from context, we use span in the context of integers to refer to the set of linear combinations with coefficients from $\mathbb{Z}$, as in the definition of a lattice. The rank of a matrix or vector space over a field $\mathbb{F}$ is the size of the minimal set of vectors that spans the space over $\mathbb{F}$.

A.3 Concentration Inequalities

Concentration inequalities allow us to bound the probability that certain random variables take values too far away from their mean. In this work, we use the following version of the well known Chernoff-Hoeffding bound [18].

▶ **Lemma 8** (Chernoff-Hoeffding Bound [18]). *Let $X = \sum_{i=1}^{m} X_i$ such that $X_i \in [0,1]$ are independent random variables. Let $\mu = \mathrm{E}[X]$. Then, for all t,*

$$\Pr[|X - \mu| > t] \leq 2e^{-2t^2/m}.$$

B Existing Proofs for RS

We give a brief overview of the currently known proof systems for $\mathcal{RS}_N$, focusing on the practical setting of non-interactive proofs. When discussing the proofs below, we use λ to denote the security parameter. Informally, we say that a verifier is efficient if it runs in time $\mathrm{poly}(\lambda, \log T)$, essentially independent of the time bound T.

- **The empty proof.** The prover can always do nothing and let the verifier check the relation $y = x^{2^T} \bmod N$ itself.
  This is a valid, albeit not very helpful, proof system that is perfectly complete and sound. In terms of efficiency, the verifier runs in time T to compute T squares, so nothing has been gained.
- **The factoring proof.** The prover can factor N to get primes p, q where $N = p \cdot q$ and send (p, q) to the verifier. The verifier can check that indeed $N = p \cdot q$, compute the order of the group $\varphi(N)$, and check if $y = x^{2^T \bmod \phi(N)} \bmod N$.
  This protocol is extremely efficient for the verifier, and is perfectly complete and sound. However, such a proof disallows N to be reused again since RS is not a sequential function whenever p, q are known. Furthermore, unless P generated N itself, it requires an inefficient prover.
- **Sumcheck-style proofs.** This is a general proof style that follows the structure of the sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [27]. The main idea is that the prover first splits the statement (x, y, T) into $k \geq 2$ sub-statements (x_i, y_i, T') for $i \in [k]$ for $T' < T$. Then, the verifier uses its randomness to merge these sub-statements into a single statement (x', y', T') which is hopefully easier to handle. Such protocols naturally lend themselves to recursive interactive proofs. We note that the proofs of [30, 11, 4, 19] as well as a generic proof for space-bounded computation [32] generally fall into this framework. We focus on Pietrzak's protocol [30] as it is the simplest and is specifically tailored for $\mathcal{RS}_N$.
  In the proof of [30], the prover sends a midpoint $\mu = x^{2^{T/2}}$, which induces two sub-statements $(x, \mu, T/2)$ and $(\mu, y, T/2)$. The verifier samples a random exponent $r \leftarrow [2^\lambda]$, and computes a new statement $(x', y', T/2)$ where $x' = x^r \cdot \mu$, $y' = \mu^r \cdot y$, and $T' = T/2$. In the non-interactive setting, the verifier can then simply check $(x')^{2^{T/2}} = y'$ itself.
  In terms of efficiency, this protocol only cuts down the running time of the verifier by a factor of 2. [11] show how to reduce this to an $T/k + 1$-time verifier for any $k \geq 0$ by having the prover sending k evenly spaced midpoints.
  It's easy to see that if (x, y, T) is valid, then so is (x', y', T'). Soundness follows since if (x, y, T) is invalid, then (x', y', T') becomes valid only with probability at most $O(1/s)$, where s is the size of the smallest subgroup of $\mathbb{Z}_N^\star$. If N is a product of safe primes, then $s = 2^\lambda$, and the protocol is statistically sound. Block et al. [4] show how to adapt this protocol, at the cost of $O(\lambda)$ multiplicative overhead in communication, to be statistically sound for *any* multiplicative group.

- **FS-style arguments.** We can get non-interactive proofs by applying the Fiat-Shamir (FS) heuristic [12] to the public-coin, interactive variants of the sumcheck-style proofs above. Again, we focus on the protocol of Pietrzak [30] for sake of comparison.
  The FS heuristic generates the verifier's randomness in each round by applying a (sufficiently random) hash function on the transcript of the protocol so far. Hence, the prover can generate all of its messages without needing to interact with the verifier, resulting in a non-interactive proof.
  In the case of [30], the prover generates an initial midpoint $\mu_1 = x^{2^{T/2}}$, then hashes μ_1 (along with the statement) to get a random value $r_1 \in [2^\lambda]$. The prover can then compute $(x', y', T/2)$ itself as above. At this point, P compute a second midpoint $\mu_2 = (x')^{2^{T/4}}$, generate randomness r_2 using the hash function, and continue this process r times until it generates a statement $(\hat{x}, \hat{y}, T')$ where $T' = T/2^r$ that the verifier can check directly. If $r = \log T$, then $T' = O(1)$, resulting in an efficient verifier. The prover needs to send r group elements in this protocol, so this requires $\Omega(\log T \cdot \lambda)$-bits of communication in total.
  In terms of security, we note that, even when modeling the hash function h as a random function, the resulting protocols are only computationally sound. An unbounded prover that can query the random oracle arbitrarily can generate cheating proofs for false statements. However, there is a recent line of work (see e.g. [26, 22, 3]) showing how to securely instantiate hash functions for different sumcheck-style protocols from more standard assumptions. Most relevant to us is the work of Bitansky et al. [3] that instantiates the FS-heuristic for the interactive proof of [4] for $\mathcal{RS}_N$ assuming only (polynomially hardness) LWE using the hash function of [20].
- **Wesolowski's argument.** Wesolowski [41] gave an extremely efficient non-interactive proof for $\mathcal{RS}_N$ where the prover sends a single group element and the verifier computes only $O(\lambda)$ squares. In this protocol, the verifier first samples a random λ-bit prime ℓ (or is sampled using a random function as in the FS-heuristic), and the prover sends an ℓth root of y, $\pi = x^{\lfloor 2^T/\ell \rfloor}$. The verifier then accepts iff $y = \pi^\ell \cdot x^c$ for $c = 2^T \bmod \ell$.
  The computational soundness of this protocol relies on a new "adaptive root assumption", which says that the prover cannot compute an ℓth root of a group element for a random prime ℓ. Aside from this assumption being relatively new, this protocol is *broken* if the prover knows the factorization of N. Namely, given the order of the group, the prover can break the adaptive root assumption. This means that the protocol additionally requires a strong assumption on the setup used to generate N as well. We note that this style of assumption is not required for the computational soundness for the FS-style arguments mentioned above.

C Extension to General Hidden Order Groups

Let $\mathbb{G}$ be any finite, abelian, multiplicative group. For any $\lambda \in \mathbb{N}$, we let $\mathsf{GroupGen}(1^\lambda)$ be an algorithm that outputs some group of size $[2^\lambda, 2^\lambda + 1)$ such that it is believe that it is hard to compute the order of a random group $\mathbb{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$. Any such group $\mathbb{G}$ must be finitely generated, so there exist elements $g_1, \ldots, g_s$ such that every $h \in \mathbb{Z}_N^\star$ is equal to $\prod_{i=1}^s g_i^{h|_i}$, where $h|_i \in \mathbb{Z}$ is the ith component of h. We use $\mathsf{ord}(\mathbb{G})$ to denote the size of the group, and $\mathsf{ord}(g)$ to denote the minimum c such that $g^c = 1$. Borrowing notation from $\mathbb{Z}_N^\star$, we use $\mathsf{Carm}(\mathbb{G})$ to denote the maximum value of $\mathsf{ord}(g)$ for any $g \in \mathbb{G}$. In particular, there must exist some $g \in \mathbb{G}$ such that $\mathsf{ord}(g) = \mathsf{Carm}(\mathbb{G})$.

The proof of Theorem 4 goes through by considering an arbitrary group $\mathbb{G}$. We refer to the full version of the paper for more details.

▶ **Corollary 9.** *Let* $\lambda \geq 2, T \in \mathbb{N}$, $k\colon \mathbb{N} \to \mathbb{N}$, $\delta, \epsilon\colon \mathbb{N} \to [0,1]$, $\mathbb{G} \in \mathrm{Supp}\left(\mathsf{GroupGen}(1^\lambda)\right)$, *and* (P,V) *be a* k*-element straight-line generic-group proof system for the function* $f_{\mathbb{G},T}(x) = x^{2^T} \in \mathbb{G}$ *with soundness error* δ. *For any* $(\mathsf{st}, \pi_1, \ldots, \pi_{k(\lambda)}) \in \mathrm{Supp}\left(P(1^\lambda, \mathbb{G}, T, x, f_{\mathbb{G},T}(x))\right)$. *If*

$$\Pr_{\rho}\left[\mathsf{ParTime}_V(1^\lambda, \mathbb{G}, T, \mathsf{st}) < \frac{T}{(k(\lambda)+1)} - \log(k(\lambda)+1)\right] \geq \max(2\delta(\lambda), \epsilon(\lambda)),$$

then there exists a standard model probabilistic $\mathsf{poly}(\lambda, k(\lambda), T, 1/\epsilon(\lambda)) \cdot \mathsf{Time}_V(1^\lambda, \mathbb{G}, \mathsf{st})$ *time algorithm* A *such that*

$$\Pr\left[c \leftarrow A\left(1^\lambda, \mathbb{G}, k, T, \mathsf{st}, 1/\epsilon(\lambda)\right) : \mathsf{ord}(\mathbb{G}) \text{ divides } c, c \neq 0\right] \geq 1 - 2^{-\lambda}.$$