Certifying Giant Nonprimes

Charlotte Hoffmann[1], Pavel Hubáček[2], Chethan Kamath[3], and Krzysztof Pietrzak[1]

[1] Institute of Science and Technology Austria, Austria
{krzysztof.pietrzak,charlotte.hoffmann}@ist.ac.at
[2] Institute of Mathematics, Czech Academy of Sciences, Prague, Czech Republic
[3] Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
hubacek@iuuk.mff.cuni.cz
[4] Tel Aviv University, Israel
ckamath@protonmail.com

Abstract. GIMPS and PrimeGrid are large-scale distributed projects dedicated to searching giant prime numbers, usually of special forms like Mersenne and Proth primes. The numbers in the current search-space are millions of digits large and the participating volunteers need to run resource-consuming primality tests. Once a candidate prime N has been found, the only way for another party to independently verify the primality of N used to be by repeating the expensive primality test. To avoid the need for second recomputation of each primality test, these projects have recently adopted certifying mechanisms that enable efficient verification of performed tests. However, the mechanisms presently in place only detect benign errors and there is no guarantee against adversarial behavior: a malicious volunteer can mislead the project to reject a giant prime as being non-prime.

In this paper, we propose a practical, cryptographically-sound mechanism for certifying the non-primality of Proth numbers. That is, a volunteer can – parallel to running the primality test for N – generate an efficiently verifiable proof at a little extra cost certifying that N is not prime. The interactive protocol has statistical soundness and can be made non-interactive using the Fiat-Shamir heuristic.

Our approach is based on a cryptographic primitive called Proof of Exponentiation (PoE) which, for a group $\mathbb{G}$, certifies that a tuple $(x, y, T) \in \mathbb{G}^2 \times \mathbb{N}$ satisfies $x^{2^T} = y$ (Pietrzak, ITCS 2019 and Wesolowski, J. Cryptol. 2020). In particular, we show how to adapt Pietrzak's PoE at a moderate additional cost to make it a cryptographically-sound certificate of non-primality.

1 Introduction

The search for giant primes has long focussed on primes of special forms due to the availability of faster, custom primality tests.Two of the most well-known examples are

Mersenne numbers of the form $M_n = 2^n - 1$, for some $n \in \mathbb{N}$, which can be tested using the Lucas-Lehmer or the Lucas-Lehmer-Reisel test [25,23,35]; and

Proth numbers of the form $P_{k,n} = k2^n + 1$, for some $n \in \mathbb{N}$ and odd $k \in \mathbb{N}$, which can be tested using Proth's theorem [33].

To harness the computational resources required for finding giant primes, there are massive distributed projects like GIMPS (Great Internet Mersenne Prime Search) and PrimeGrid dedicated to the search for giant primes of special forms, including the ones above. A volunteer in such a distributed project can download an open-source software that locally carries out primality tests on candidate numbers, at the end of which, a candidate is either rejected as a composite number or confirmed as a new prime. The largest-known prime as of now is a Mersenne prime $(2^{82,589,933} - 1)$ with 24,862,048 decimal digits found by GIMPS [16].

Testing primality of giant numbers. The search for large primes is a time-consuming process: the GIMPS website warns that a single primality test could take up to a month. The reason for this is that these tests – whenever the prime candidate has no small prime factors[5] – require the computation of a very long sequence modulo an extremely large number. For example, Proth's theorem [33] states that $P_{k,n} = k2^n + 1$ is prime if and only if, for a quadratic non-residue x modulo $P_{k,n}$, it holds that

$$x^{k2^{n-1}} \equiv -1 \bmod P_{k,n}. \tag{1}$$

To date, the largest-known Proth prime is $10223 \cdot 2^{31,172,165} + 1$ [31]. Since n is of the order of magnitude 10^7 and the square-and-multiply algorithm is the fastest way currently known to carry out exponentiation, the test roughly requires 10^7 squarings modulo a 10^7-digit modulus. Unfortunately, performing this test does not yield an immediate witness that certifies the correctness of the result – in particular, if $P_{k,n}$ is composite, the test does not find a divisor of $P_{k,n}$.[6] Until very recently, the standard way for another party to independently validate the test result was by recomputing the result of Equation (1). In 2020, Pavel Atnashev demonstrated that a cryptographic primitive called *Proof of*

[5] GIMPS first tests by trial division whether a candidate number has any prime divisors of size up to a bound between 2^{66} and 2^{81}. Only when this is not the case is that they run a more expensive specialized primality test: details can be found on this page.

[6] Note that some primality tests, like, e.g., Miller-Rabin [27,34], can be modified to (sometimes) yield factors in case the number being tested is not a prime.

Exponentiation (PoE) might be applicable in the context of these specialized primality tests to avoid the costly second recomputation.[7]

PoEs and efficient verification of primality tests. For a group $\mathbb{G}$, a PoE [28,39] is an interactive protocol for the language

$$L_{\mathbb{G}} := \left\{(x, y, T) \in \mathbb{G} \times \mathbb{G} \times \mathbb{N} : x^{2^T} = y \text{ over } \mathbb{G}\right\}. \tag{2}$$

In case the PoE is public-coin, it can be transformed into a non-interactive PoE using the Fiat-Shamir heuristic [14]. A PoE enables efficient verification of costly iterated exponentiation even without the knowledge of the order of the underlying group. Since the primality test using Proth's theorem amounts to iterated exponentiation, it seems immediate that one would attempt to exploit PoEs also towards efficient verifiability in the context of primality tests for giant numbers. The idea is for the volunteer to use the (non-interactive) PoE to compute – alongside the result of the test – a proof that helps any other party verify the result. For this approach to be feasible,

1. computing the proof should not require much more additional resource (relative to the iterated exponentiation induced by the specialized primality test), and
2. the cost of verifying a proof should be significantly lower than that of recomputing the exponentiation.

Recently, this approach has been deployed in both GIMPS [17] and PrimeGrid [32], where (non-interactive) Pietrzak's PoE [28] is used to certify (both primality and non-primality) of Mersenne and Proth numbers when used along with Lucas-Lehmer-Riesel test and Proth's theorem, respectively. In fact, one of the recently-found Proth primes, $68633 \cdot 2^{2715609} + 1$, has been certified so.

However, PoEs were constructed for groups whose order is hard to compute like, e.g., RSA group [36] or class group [11]. In such groups, the only known way to compute x^{2^T} is via T sequential squaring. On the other hand, if one party knows the group order, they can not only speed up this exponentiation[8] but also (in many groups) construct *false* Pietrzak PoEs that lead a verifier to accept proofs for false statements. In the context of primality testing the underlying group is $\mathbb{Z}_{P_{k,n}}$, so the group order is known whenever $P_{k,n}$ is prime. While this does not speed up the computation of the primality test (since the modulus is larger than the exponent), it removes the soundness guarantee of the protocol. As we discuss next, a malicious prover can falsely convince a verifier that *any* Proth prime is composite using Pietrzak's protocol in those groups.

[7] More details can be found in this thread of mersenneforum.org. An implementation due to Atnashev is available on GitHub. The idea of using PoEs for certifying giant primes has been discussed also by Mihai Preda in another thread in the same forum already in August 2019

[8] If the order of the group is known, then x^{2^T} can be computed efficiently using the shortcut: $y = x^e$ (over $\mathbb{G}$) for $e = 2^T \bmod \text{ord}(\mathbb{G})$.

1.1 Our Contribution

The statistical security guarantee of Pietrzak's PoE applies only to groups *without* low-order elements.[9] In groups with low-order elements, one additionally requires the *low-order assumption* [8] to hold, i.e., it must be computationally hard to find a group element of low order. Boneh, Bünz, and Fisch [8] described an attack on the soundness of Pietrzak's PoE when implemented in groups where low-order elements are easy to find (see Section 1.2). This presents an issue with its usage in the GIMPS and PrimeGrid projects since there are no guarantees on the structure of the group in these applications. In fact, if $P_{k,n}$ is prime, the order of the group is $P_{k,n} - 1 = k2^n$ so low-order elements (e.g., of order 2) do exist and can be found without much effort. We show in Appendix A how a malicious volunteer can exploit the attack from [8] to generate a proof that "certifies" an arbitrary Proth *prime* as composite with constant probability. Indeed, people at GIMPS and PrimeGrid were aware of this [3] and Pietrzak's PoE is currently employed in these projects more-or-less as a checksum to catch benign errors (e.g., hardware errors). When a volunteer is malicious and deliberately tries to mislead the project, there are no guarantees. This could force the volunteer network to waste additional computation and possibly postpone the discovery of another giant prime by years.

Are Cryptographically-Sound Certificates Possible? In our work, we explore whether any cryptographic guarantee for *practical* proofs is possible in the above scenarios. Whilst it is theoretically possible to use existing results to certify non-primality, these measures, as we discuss in Section 1.1, turn out to be too expensive. As a first step towards practical proofs, we show how to achieve soundness for proving *non-primality* of Proth numbers. That is, we construct an interactive protocol for the language

$$L := \big\{(k,n) \in \mathbb{N}^2 : k \text{ is odd and } P_{k,n} = k2^n + 1 \text{ is not a prime}\big\}. \tag{3}$$

While ideally, one would want to certify both primality and non-primality, the latter is much more important for projects like GIMPS and PrimeGrid: they worry about missing out on primes rather than false claims stating that a composite is a prime. Primes are very sparse[10], so double checking claims of primality is not a problem, but performing *each* primality check twice to catch benign errors or a malicious volunteer is almost twice as expensive as using a sound non-primality test as suggested in this work.

[9] Recall that the order of an element $g \in \mathbb{G}$, denoted $\mathrm{ord}(g)$, is the least integer such that $g^{\mathrm{ord}(g)} = 1$. An example of a group without low-order elements is *signed quadratic residues* QR_N^+, where N is sampled as the product of safe primes [15,21]. This is also the algebraic setting used in Pietrzak's VDF [28].

[10] For $N \in \mathbb{N}$, let $\pi(N)$ denote the number of primes less than N. By prime number theorem, asymptotically $\pi(N)$ approaches $N/\log(N)$. For the case of Proth numbers, however, even the question of whether there are infinitely many of them is open [9].

Our interactive protocol has statistical soundness: if the candidate number to be tested is indeed a Proth prime, then even a computationally unbounded malicious prover (a malicious volunteer) will not be able to convince the verifier (say a server run by the project) that it is composite.

Theorem 1 (Informal). *There is a practical public-coin statistically-sound interactive proof for the non-primality of Proth numbers.*

We provide an overview of our interactive protocol in the next section. Since it is public-coin, our interactive protocol can be made non-interactive via the Fiat-Shamir transform [14]. In general, the Fiat-Shamir transform only works for constant round protocols, which is not the case for our protocol, so showing that Fiat-Shamir works in our case needs a proof.

Corollary 1 (Informal). *In the random-oracle model, there is a practical statistically-sound non-interactive proof for the non-primality of Proth numbers.*

Concrete Efficiency We defer exact details about the complexity of our protocol to Section 4.3. Here, we provide concrete (worst-case) numbers for our non-interactive proof using the largest Proth prime known to date as the candidate: $10223 \cdot 2^{31172165} + 1$ [31]. For $k = 10223$, $n = 31172165$ and security parameter $\lambda = 80$:

- the prover (additionally) stores 5584 group elements (which is around 20GB) and performs 13188 multiplications;
- the verifier performs 10046 multiplications; and
- the proof size is 26 elements of size 31172179, i.e., around 102 MB.

Note that recomputing the result of the primality test would take $n = 31172165$ multiplications, so our protocol reduces the number of multiplications by a factor of $\lfloor 31172165/(13188 + 10046) \rfloor = 1341$. Note that this takes the order of hours rather than days. In Section 4.4, we show that the additional cost of our protocol compared to the one that is being used now (which is not cryptographically sound) is moderate: In the above example, the prover performs 2021 and the verifier 4046 multiplications more than in the current implementation.

Applicability of Existing Statistically-Sound PoEs. The issue with low-order elements when using Pietrzak's PoE out-of-the-box can be resolved using alternative PoEs that are statistically sound in *arbitrary* groups.[11] Indeed, such PoEs were recently proposed [6,20]. [6] can be regarded as a parallel-repeated variant of Pietrzak's protocol but, to achieve statistical soundness, the number of repetitions is as large as the security parameter. This leads to significant overhead in terms of both proof-size and computation. For example, to compute the PoE of [6] for the Proth prime from Section 1.1, the prover needs to perform

[11] One could also use SNARGs [22,26] for this purpose but, being a general-purpose primitive, the resulting schemes would not be practically efficient.

893312 multiplications and it takes the verifier 318800 multiplications to verify the proof consisting of 2080 group elements (i.e., 8160 MB). This means that our protocol reduces the number of multiplications of [6] by a factor of 52 and the proof size by a factor of 80. The overall approach in [20] is similar to that in [6], but it improves on the complexity of [6] whenever it is possible to choose the exponent to be a large q of a special form. In the primality testing application, we do not have the freedom to choose the exponent and, for the case of $q = 2$, the complexity of [20] is comparable to that of [6].

1.2 Technical Overview

Our starting point is Pietrzak's PoE (`PPoE`, Figure 2), which is a statistically-sound $\log(T)$-round interactive protocol for the language $L_{\mathbb{G}}$ from Equation (2). We start with its overview (which is adapted from [20]). The protocol in [28] is recursive in the parameter T and involves $\log(T)$ rounds of interaction. To prove a (true) statement (x, y, T), the (honest) prover $\mathcal{P}$, in the first round, sends the "midpoint" $z := x^{2^{T/2}}$ to the verifier $\mathcal{V}$. This results in two *intermediate* statements $(x, z, T/2)$ and $(z, y, T/2)$. Next, $\mathcal{V}$ sends a random challenge r to $\mathcal{P}$, and they merge these two intermediate statements into a *new* statement $(x^r \cdot z, z^r \cdot y, T/2)$. The above steps constitute the "halving" sub-protocol, which is repeated $\log(T)$ times, halving the parameter T each time, until $\mathcal{P}$ and $\mathcal{V}$ arrive at a (base) statement for $T = 1$. At this point, $\mathcal{V}$ can efficiently check the correctness on its own by performing a single squaring.

Problem with low-order elements. The soundness argument in groups without low-order elements proceeds in a round-by-round manner as follows: when starting with a false statement, it is guaranteed that at least one of the two intermediate statements is false and one can argue that the new statement is false with high probability (over the choice of r). In groups that have easy-to-compute low-order elements, the above argument fails and we recall the attack described in [8]. In the following discussion, by a "μ-false" statement, we refer to a statement that is off a true statement by a factor of $\mu \in \mathbb{G}$. Suppose that $\mu \in \mathbb{G}$ has order 2 and let (x, y, T) be a true statement. For the μ-false statement $(x, y\mu, T)$, the cheating prover simply sends μz as its first message and the claim is that the new statement at the end of the first halving sub-protocol is true with probability 1/2. To see this, note that the new statement is $(x^r \cdot \mu z, (\mu z)^r \cdot y, T/2)$ and whenever r is odd, it reduces to the true statement $(\mu \cdot x^r \cdot z, \mu \cdot z^r \cdot y, T/2)$ (since μ vanishes when exponentiated to an even power $T/2$). Thus, the verifier eventually accepts. Therefore, applying `PPoE` out-of-the-box as a certificate of non-primality for a Proth number $P_{k,n}$ is not sound since the group $\mathbb{Z}^*_{P_{k,n}}$ might have easy-to-find elements of low order. We show in Appendix A that this is indeed the case and it is not hard to generate `PPoE` proofs that "certify" a prime $P_{k,n}$ as composite.

Working around low-order elements. The way low-order elements are dealt with in [6,20] is via parallel repetition and/or by working with exponents q of a

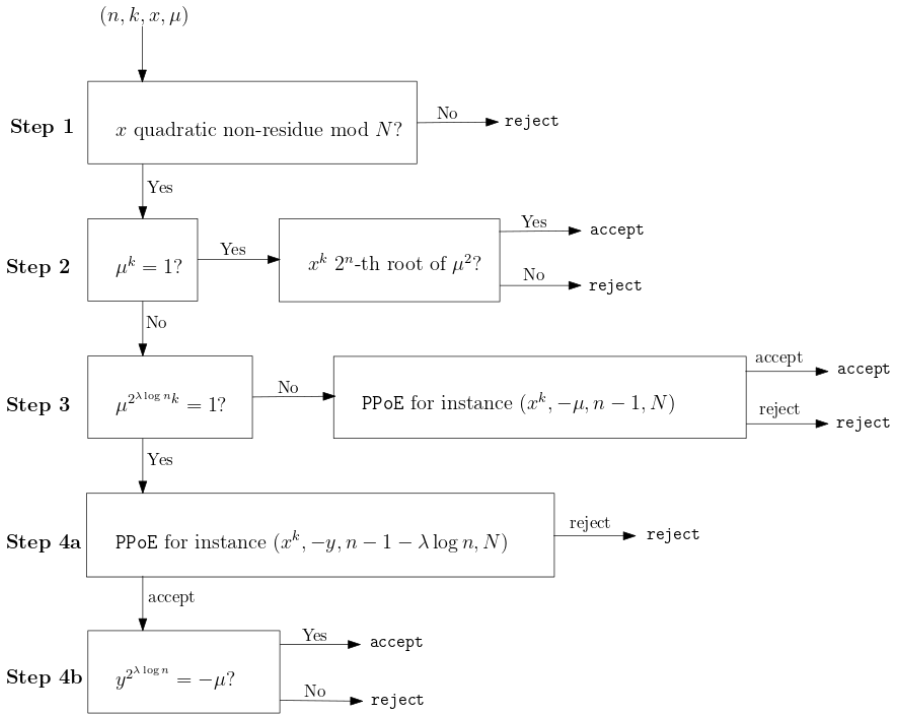

Fig. 1. Overview of the protocol in Figure 3. All computations are done in the group $\mathbb{Z}_N^*$, where $N = k2^n + 1$.

particular form. As explained in Section 1.1, we cannot exploit either of these techniques because of efficiency reasons and the restriction on the exponent placed by the primality test. Nevertheless, our interactive protocol, described in Figures 1 and 3, builds on some of the ideas in [28,20] to get around the issue of low-order elements for the specific exponentiation considered in Proth's test (Equation (1)). Below, we give an overview of how this is accomplished – we refer the readers to Section 4 for a more detailed overview.

For a prime $N := P_{k,n}$, suppose that $x \in \mathbb{Z}_N^*$ is a quadratic non-residue.[12] Suppose that a malicious prover $\mathcal{P}^*$ tries to convince the verifier $\mathcal{V}$ that

$$x^{k2^{n-1}} \equiv -\mu \bmod N, \quad 1 \neq \mu \in \mathbb{Z}_N^*. \tag{4}$$

Since N is a prime and the result must be -1 by Proth's theorem, the statement $(x, -\mu, k2^{n-1})$ corresponding to Equation (4) is μ-false. Our protocol exploits the fact that $\mathcal{V}$ does not care about the exact value of $x^{k2^{n-1}}$ and it rejects as long as the correct result is not equal to -1. This observation greatly simplifies

[12] In the actual protocol, we explain how the verifier can check if the Jacobi symbol of x is -1 (Step 1 in Figure 3).

the task for $\mathcal{V}$. As we show, it is sufficient to perform a few efficient checks on the order of μ, depending on which $\mathcal{V}$ can choose a sound method for verification. For ease of exposition, we restrict this overview to the case where k itself is prime.

- Our starting point is the case where $\mathrm{ord}(\mu)$ is "large", by which we mean $\mathrm{ord}(\mu) \nmid k2^{\lambda \log(n)}$ (Step 3). We show in this case that it is possible to use PPoE out of the box to prove the statement $(x^k, -\mu, n-1)$, which is equivalent to the statement in Equation (4). Key to proving this is the following observation on the fine-grained nature of soundness of PPoE: the "falseness" of a statement in each round of the sub-protocol cannot decrease by too much. More precisely, if the cheating prover starts with an α-false statement then the new statement is β-false for some β whose order cannot be much smaller than α's. Therefore, if the cheating prover starts off with a statement that is sufficiently false, which turns out to be when $\mathrm{ord}(\alpha) = k2^{\lambda \log(n)}$, then the statement in the final round remains false with overwhelming probability and is rejected by the PPoE verifier. We formalise this observation in Lemma 1 and point out that, while a similar lemma was proved in [20], there are some crucial differences: see Remark 3.
- Next is the case where $\mathrm{ord}(\mu)$ is "small and odd" (Step 2), i.e., in this overview $\mathrm{ord}(\mu) = k$. In this case, $\mathcal{V}$ can verify the statement in Equation (4) *without* any help from $\mathcal{P}^*$ as follows: find the inverse of 2^n modulo k and raise μ^2 to that element.[13] By Equation (4), this yields the same element as x^k if the prover is honest. If N is prime, this will yield a different element than x^k as we show in Section 4.2. This quick verification is only possible since k and 2^n are coprime, so it can only be used in this case.
- Finally, consider the case where the order is "small and even", by which we mean $\mathrm{ord}(\mu) \mid 2^{\lambda \log(n)}$ (Step 4). Here, we are in a situation where PPoE does not guarantee soundness (since the statement is not "false enough"). However, as in [20], it is possible to reduce the task of checking Equation (4) to that of verifying, using PPoE, the "smaller" statement obtained by taking the $2^{\lambda \log(n)}$-th root of Equation (4). To be precise, $\mathcal{P}$ and $\mathcal{V}$ verify the statement $(x^k, y, n - 1 - \lambda \log(n))$ using PPoE and, if convinced, $\mathcal{V}$ then checks whether $y^{2^{\lambda \log(n)}} = -\mu$, by itself, using a final exponentiation. This final exponentiation forces a malicious prover $\mathcal{P}^*$ to cheat with an element of high enough order during the PPoE. To see this, assume for example that $\mathcal{P}^*$ sends the honest result $y = x^{2^{n-1-\lambda \log(n)}}$. Then, $\mathcal{V}$'s final exponentiation leads to outright rejection since

$$y^{2^{\lambda \log(n)}} = (x^{k2^{n-1-\lambda \log(n)}})^{2^{\lambda \log(n)}} = x^{k2^{n-1}} \neq -\mu.$$

  On the other hand, $\mathcal{P}^*$ cheating with an element of such high order during the PPoE makes the verifier reject this PPoE with overwhelming probability (as in the first case).

[13] Note that if μ has order k, then $-\mu$ has order $2k$ (which is not coprime to 2^{n-1}), which is the reason we have to square the statement in Equation (4) before computing the inverse of the exponent.

We refer the reader to Sections 3 and 4 for the formal analysis.

1.3 Related Work

General-purpose primality testing. Pratt showed that primality testing (of arbitrary integers) lies in the class **NP**, via the eponymous Pratt certificates [30] (an alternative certificate of primality is the Atkin-Goldwasser-Kilian-Morain certificate [2,18]). Coupled with the fact that non-primality has succinct certificates in the form of factorization (which can be efficiently checked by integer multiplication) placed primality testing in **NP**∩**co-NP**. Probabilistic tests like Solovay-Strassen [38], Miller-Rabin [27,34] and Baillie–PSW [4,29] soon followed, which placed primality testing in classes like **BPP**, **RP** or **ZPP**.[14] Finally, Agrawal, Kayal, and Saxena [1] settled the question by showing that primality testing is in **P**. We refer the readers to [1] for a more detailed exposition on (general-purpose) primality testing.

Giant prime numbers and custom primality tests. In addition to Mersenne numbers M_n and Proth numbers $P_{k,n}$, numbers of special form that have been targetted in the search for giant primes include Fermat numbers $F_n := 2^{2^n} + 1$ (which are a special case of Proth numbers), generalised Fermat numbers $F_{a,b,n} := a^{2^n} + b^{2^n}$ and Woodall numbers $W_n := n \cdot 2^n - 1$. We refer the readers to PrimePages for a more comprehensive list. These numbers of special forms are amenable to custom primality tests that run faster than general-purpose primality tests. For example, the Lucas-Lehmer (LL) test [25,23] is a determinstic primality test for M_n that runs in time $O(n \cdot \mu(n))$, where $\mu(n)$ denotes the complexity of multiplying two n-bit integers.[15] In comparison, for M_n, the complexity of deterministic AKS primality test is $\tilde{O}(n^6)$ and the complexity of probabilistic Miller-Rabin test is $O(\lambda n \cdot \mu(n))$ (for a statistical error of $2^{-\lambda}$). GIMPS relies on the Lucas-Lehmer-Riesel [35] test, which is a generalization of the Lucas-Lehmer test for numbers of the form $k2^n - 1$. PrimeGrid performs a variety of primality tests including Proth's theorem [33] for Proth numbers. They were first to realize that (Pietrzak's) PoE can be used to certify the results of Proth's primality test [32]. They also noticed that low-order elements can affect the soundness of the protocol and, therefore, included some checks on the order of the result [3]. For Proth number $P_{k,n}$, *given* a quadratic non-residue modulo $P_{k,n}$, the complexity of Proth's test [33] is $O(\log(k) \cdot n\mu(n))$; otherwise it is a Las Vegas test (since we currently know how to generate a quadratic non-residue only in expected polynomial-time). An alternative is to use the deterministic Brillhart-Lehmer-Selfridge test [10].

More Related Work on PoE. PoE was introduced in the context of another cryptographic primitive called Verifiable Delay Function (VDF) [7]. The VDFs

[14] In fact, Miller's test [27] runs in strict polynomial time assuming the Generalised Riemann Hypothesis.

[15] Since these numbers have a succinct representation, the complexity of these tests is, strictly-speaking, exponential in the size of the input (which is n for M_n).

of Pietrzak [28] and Wesolowski [39], both, implicitly involve the construction of a PoE: an overview and comparison of these PoE protocols can be found in [8]. The soundness of these PoEs relies on new hardness assumptions called *low-order assumption* and, the stronger, *adaptive root assumption*, respectively [8], i.e., strictly-speaking, these are *arguments* of exponentiation. Pietrzak's PoE [28] is, however, statistically-sound in groups with the syntactic guarantee that *no* elements of low order exist, e.g., a subgroup of quadratic residues of RSA group. In addition, there are two more statistically-sound PoE constructions currently known: [6] and [20]. [6] can be seen as a elaborate parallel repetition of [28] that is statistically sound in *any* group. However, this repetition increases the complexity of the protocol by a multiplicative factor λ, where λ is a statistical security parameter. [20] improves on the construction in [6] and reduces the complexity by almost one order of magnitude *whenever* one can freely choose the exponent in the exponentiation. Finally, PoEs have recently been used as a crucial building block in constructing space-efficient general-purpose succinct non-interactive arguments of knowledge (SNARKs) [12,6], thus establishing a converse relationship with SNARGs (since SNARGs trivially imply PoEs). Recently, Rotem studied the problem of batching PoEs in his work on batching VDFs [37].

2 Preliminaries

Interactive protocols. Let Σ be an alphabet. For $\ell \in \mathbb{N}$, an interactive protocol consists of a pair $(\mathcal{P}, \mathcal{V})$ of interactive Turing machines called prover and verifier, respectively. In an ℓ-round (i.e., $(2\ell-1)$-message) interactive protocol, $\mathcal{P}$ and $\mathcal{V}$ run on a common input x and proceed as follows: in each round $i \in [1, \ell]$, first $\mathcal{P}$ sends a message $\alpha_i \in \Sigma^a$ to $\mathcal{V}$ and then $\mathcal{V}$ sends a message $\beta_i \in \Sigma^b$ to $\mathcal{P}$, where Σ is a finite alphabet. At the end of the interaction, $\mathcal{V}$ runs a (deterministic) Turing machine on input $\{x, (\beta_1, \ldots, \beta_\ell), (\alpha_1, \ldots, \alpha_\ell)\}$. The interactive protocol is *public-coin* if β_i is a uniformly distributed random string in Σ^b.

Interactive proofs. The notion of an interactive proof for a language L is due to Goldwasser, Micali and Rackoff [19].

Definition 1. *An interactive protocol $(\mathcal{P}, \mathcal{V})$ is an* ϵ-sound interactive proof system *for L if:*

- ***Completeness:*** *For every $x \in L$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on common input x, then $\mathcal{V}$ accepts with probability* 1.
- ***Soundness:*** *For every $x \notin L$ and every cheating prover strategy $\widetilde{\mathcal{P}}$, the verifier $\mathcal{V}$ accepts when interacting with $\widetilde{\mathcal{P}}$ with probability at most $\epsilon(|x|)$, where $\epsilon = \epsilon(n)$ is called the* soundness error *of the proof system.*

In particular, the interactive protocol is a statistically-sound proof *if the soundness holds against computationally-unbounded cheating prover strategies and the soundness error is negligible (in the input-length).*

Non-interactive proofs in the random-oracle model. A non-interactive protocol involves the prover sending a single message to the verifier. We are interested in non-interactive proofs in the random-oracle model instead of the more standard non-interactive arguments in the common reference string (CRS) model. Therefore, we now have to consider oracle interactive Turing machines $\mathcal{P}^{(\cdot)}$ and $\mathcal{V}^{(\cdot)}$ for the prover and verifier.

Definition 2. *A pair of oracle machines $(\mathcal{P}^{(\cdot)}, \mathcal{V}^{(\cdot)})$ is an* ϵ-sound non-interactive proof system *for a language L in the random-oracle model if the following properties hold:*

- ***Completeness:*** *For every $x \in L$,*

$$\Pr_{O \leftarrow \mathcal{O}} \left[\mathcal{V}^O(x, \mathcal{P}^O(x)) = 1 \right] = 1,$$

  where the probability is over the random choice of the oracle $O \in \mathcal{O}$.
- ***Soundness:*** *For every (computationally-unbounded) cheating prover strategy $\widetilde{\mathcal{P}}$,*

$$\Pr_{\substack{O \leftarrow \mathcal{O} \\ (x, \widetilde{\pi}) \leftarrow \widetilde{\mathcal{P}}^O}} \left[\mathcal{V}^O(x, \widetilde{\pi}) = 1 \wedge x \notin L \right] \leq \epsilon(|x|).$$

Remark 1 (On the complexity of the honest prover and verifier). In standard definitions, the prover is either unbounded and deterministic, or given additional information (e.g., the witness when the language is in **NP**), whereas the verifier is polynomially-bounded and randomised. We prefer a more fine-grained definition where the (deterministic) prover is stronger than the (randomised) verifier, but both parties can be polynomially-bounded.

Remark 2. The definition of interactive PoEs and non-interactive PoEs in the random-oracle model can be recovered by restricting Definitions 1 and 2 to the language $L_{\mathbb{G}}$ from Equation (2).

3 Pietrzak's PoE in Groups of Known Order

In this section, we recall Pietrzak's PoE (`PPoE`) [28] and some of its properties. The protocol is presented in Figure 2. By inspection of the protocol we see that it has perfect completeness. Pietrzak proved the following complexity results in [28]:

Proposition 1 ([28, Section 6.2]). *On instance $(x, y, T, \mathbb{G})$ `PPoE` has the following efficiency properties:*

1. *$\mathcal{V}$ performs $3\lambda \log T$ multiplications in $\mathbb{G}$.*
2. *$\mathcal{P}$ performs $2\sqrt{T}$ multiplications in $\mathbb{G}$ and stores $\sqrt{T}$ group elements to compute the proof.*
3. *The size of the proof is $\log T$ elements of $\mathbb{G}$.*

Instance: $(x, y, T, \mathbb{G})$, where $x, y \in \mathbb{G}$ and $T \in \mathbb{N}$ is even

Parameters: statistical security parameter λ

Statement: $x^{2^T} = y$ in $\mathbb{G}$

Protocol:

1. For $T = 1$:
   - If $x^2 = y$, output `accept`.
   - Else, output `reject`.
2. For $T > 1$:
   (a) $\mathcal{P}$ sends $v = x^{2^{T/2}}$ to $\mathcal{V}$.
   (b) If $v \notin \mathbb{G}$, $\mathcal{V}$ outputs `reject`. Otherwise, $\mathcal{V}$ samples $r \leftarrow \{0, 1, \ldots, 2^\lambda - 1\}$ uniformly at random and sends it to $\mathcal{P}$.
   (c) $\mathcal{P}$ and $\mathcal{V}$ compute $x' := x^r v$ and $y' := v^r y$ in $\mathbb{G}$.
   (d) If $T/2$ is even, $\mathcal{P}$ and $\mathcal{V}$ run the protocol on instance $(x', y', T/2, \mathbb{G})$. If $T/2$ is odd, $\mathcal{P}$ and $\mathcal{V}$ run the protocol on instance $(x', y'^2, (T+1)/2, \mathbb{G})$.

Fig. 2. `PPoE`.

Furthermore, Pietrzak proved that the PoE is statistically sound in groups without low-order elements, in particular safe prime RSA groups. Boneh et al. later proved computational soundness in groups where it is hard to find low-order elements (the low-order assumption) [8]. Ideally, we would like to use `PPoE` in a group of known order, where an adversary can find low-order elements in polynomial time. However, Boneh et al. showed that this is not sound by presenting an attack with low-order elements in [8]. In the following section, we analyze in what way these low-order elements affect the soundness of `PPoE`.

3.1 (Non-)Soundness

We analyze the soundness of `PPoE` in groups of known order. Assume that the correct result of an exponentiation is $x^{2^T} = y \mod N$ but $\tilde{\mathcal{P}}$ claims that for some $\alpha \neq 1 \mod N$ it is $x^{2^T} = y\alpha \mod N$. We sometimes call α the "bad" element and say that the second statement is α-*false*. Note that the prover's statement is of this form without loss of generality because every element has an inverse in a group. This means that if the prover claims that the result is some group element β, we can always find a group element α such that $\beta = y\alpha$. Soundess of `PPoE` only depends on the order of this bad element α. If its order only has small prime divisors with small exponents, the probability that repeated exponentiation of this element with a random exponent decreases its order to one, and thus the verifier's check for $T = 1$ passes, is not negligible.

The following lemma bounds the probability that the order of the bad element "drops" by a factor p^ℓ in one round of `PPoE`. It will be the main tool in proving soundness of our non-primality certificate later on.

Lemma 1. *Let $(x, y\alpha, T, \mathbb{G})$ be an α-false statement for some $\alpha \in \mathbb{G}$, $\mu \in \mathbb{G}$ an arbitrary group element, p^e any prime power that divides the order of α and let $r \leftarrow \{0, 1, \ldots, 2^\lambda - 1\}$ be sampled uniformly at random. Assume that the statement $(x^r\mu, \mu^r y\alpha, T/2, \mathbb{G})$ is β-false for some $\beta \in \mathbb{G}$. For any $\ell \leq e$, the probability that $p^{e-\ell+1}$ does not divide the order of β is at most $1/p^\ell$.*

Proof. If the statement $(x^r\mu, \mu^r y\alpha, T/2)$ is β-false, we have $\mu = \gamma x^{2^{T/2}}$ such that $\beta = \alpha\gamma^{r-2^{T/2}}$. We want to bound the following probability

$$\begin{aligned}\Pr_r[\beta^{p^{e-\ell}s} = \alpha^{p^{e-\ell}s}\gamma^{(r-2^{T/2})p^{e-\ell}s} = 1] &= \Pr_r[\gamma^{(r-2^{T/2})p^{e-\ell}s} = \alpha^{-p^{e-\ell}s}] \\ &\leq \frac{1}{\mathrm{ord}(\gamma^{p^{e-\ell}s})} + \frac{1}{2^\lambda} \\ &= \frac{\gcd(d, p^{e-\ell}s)}{d} + \frac{1}{2^\lambda}, \end{aligned} \tag{5}$$

where d denotes the order of γ and s is any positive integer not divisible by p. The inequality follows from the fact that the size of the randomness space is 2^λ. Now assume that the above event holds. Then we have $\gamma^{p^{e-\ell}sm} = \alpha^{-p^{e-\ell}s}$ for some integer m, hence

$$\mathrm{ord}(\alpha^{-p^{e-\ell}s}) = \mathrm{ord}(\gamma^{p^{e-\ell}sm}) = \frac{d}{\gcd(d, p^{e-\ell}sm)}$$

and equivalently

$$d = \mathrm{ord}(\alpha^{-p^{e-\ell}s})\gcd(d, p^{e-\ell}sm) \geq p^\ell \gcd(d, p^{e-\ell}sm).$$

Plugging into (5) we get

$$\Pr[\alpha^{p^{e-\ell}s}\gamma^{(r-2^{T/2})p^{e-\ell}s} = 1] \leq \frac{\gcd(d, p^{e-\ell}s)}{p^\ell \gcd(d, p^{e-\ell}sm)} + \frac{1}{2^\lambda} \leq \frac{1}{p^\ell} + \frac{1}{2^\lambda}.$$

Remark 3. A lemma of flavour similar to Lemma 1 was proven in [20, Lemma 1] for a parallel-repeated variant of `PPoE`. However, there are major differences between these: (i) the new statements in the protocol in [20] (and also [6]) are obtained in a slightly different way, using multiple random coins and (ii) [20, Lemma 1] was only proven for restricted choices of numbers p and e. Hence, Lemma 1 does not follow from [20, Lemma 1].

Corollary 2. *Let $(x, y\alpha, T, \mathbb{G})$ be an α-false statement for some $\alpha \in \mathbb{G}$ and 2^e any power of 2 that divides the order of α. The probability that $2^{e-\ell}$ does not divide the order of the bad element after one round of `PPoE` is at most $1/2^\ell$.*

Proof. By Lemma 1 we know that the probability that $2^{e-\ell+1}$ does not divide the order of the bad element of the instance $(x', y', T/2, \mathbb{G})$ is at most $1/2^\ell$. Now if T is odd, the new instance of the protocol is $(x', y'^2, (T+1)/2, \mathbb{G})$, so the bad element is squared once. This reduces its order by a factor of 2, which yields the claim.

4 Certifying Non-Primality of Proth Primes

In this section, we present the interactive protocol for verifying that a Proth number $N = k2^n + 1$ is *not* prime, i.e., that $x^{k2^{n-1}} = -\mu \mod N$ for some $\mu \neq 1$ and x a small prime number that is a quadratic non-residue modulo N. This means that from now on all group operations will be performed in the group $\mathbb{Z}_N^*$.

The protocol presented in Figure 3 consists of four steps in which $\mathcal{V}$ performs different checks on the order of the element μ and then chooses the best method for verification accordingly. An overview can be found in Figure 1.

In the first step, $\mathcal{V}$ checks if x has Jacobi symbol -1 modulo N since the primality test is only conclusive if x is a quadratic non-residue. To this end, $\mathcal{V}$ first computes $a := N \mod x$ and, if $a \neq 0$, checks if the Jacobi symbol $(\frac{x}{N}) = (\frac{a}{x})$ is -1. If $a = 0$ we know that x is a divisor of N and hence N is composite, so $\mathcal{V}$ can already accept in Step 1. If the Jacobi symbol is 1, it is unclear if x is a quadratic residue mod N so $\mathcal{V}$ rejects the proof. If the Jacobi symbol is -1, the protocol moves on to the next step.

In the second step, $\mathcal{V}$ checks if the element μ has small *odd* order dividing $k2^n$, i.e., order dividing k, by computing $\tilde{\mu} := \mu^k \mod N$. If $\tilde{\mu} \neq 1$, the order of μ does not divide k and $\mathcal{V}$ goes on to the next step. If $\tilde{\mu} = 1$, $\mathcal{V}$ can easily find the order d of μ by factoring the (small) integer k. Then $\mathcal{V}$ can verify the statement without any message from $\mathcal{P}$. In fact, $\mathcal{V}$ only verifies the statement $x^{k2^n} = \mu^2$ by computing the 2^n-th root of μ^2. Unfortunately we can not compute the 2^{n-1}-th root of $-\mu$ because $-\mu$ has order $2d$ and the inverse of 2^{n-1} modulo $2d$ does not exist. This additional squaring step eliminates potential "bad" elements of order 2, so this check only proves that $x^{k2^{n-1}} = -\mu \cdot \alpha \mod N$, for some element α of order 2 or $\alpha = 1$. Luckily, this is enough information for $\mathcal{V}$ since we only want to rule out the possibility that the result of the exponentiation is -1 and $\mu^{-1} \neq \alpha$ since μ has odd order.

If $\mathcal{V}$ gets to the third step, we know that the order of μ does not divide k. To make sure that it does not divide k times a small power of 2 either, $\mathcal{V}$ checks if $\mu^{k2^{\lambda \log n}} \neq 1 \mod N$. If this holds, we know that $\mathcal{V}$ can accept a `PPoE` for the statement $x^{k2^{n-1}} = -\mu \mod N$ because a malicious prover will only be successful in convincing $\mathcal{V}$ with negligible probability. If $\mu^{k2^{\lambda \log n}} = 1$, such a PoE is not sound, so $\mathcal{V}$ goes on to the next step.

If $\mathcal{V}$ gets to the last step, we know that the order of μ is too small to soundly accept a `PPoE`. However, we now know that the order of μ is even, so we can use the following trick: Instead of sending a `PPoE` for the statement $x^{k2^{n-1}} = -\mu \mod N$, $\mathcal{P}$ sends a `PPoE` for the statement $x^{k2^{n-1-\lambda \log n}} = y \mod N$ for some element $y \in \mathbb{Z}_N^*$. Then $\mathcal{V}$ checks if $y^{2^{\lambda \log n}} = -\mu$. If this holds and the PoE is correct, $\mathcal{V}$ outputs accept. Else, $\mathcal{V}$ outputs reject.

Remark 4. The complexity of Steps 3 and 4 of our protocol could be slightly improved with the following changes:

- Instead of $\mathcal{V}$ computing the exponentiation $\tilde{\mu}_1^{2^{\lambda \log n}}$ in Step 3, $\mathcal{P}$ could send a proof for the statement $\tilde{\mu}_1^{2^{\lambda \log n}} \neq 1$. This can be done in a sound manner

since again $\mathcal{V}$ only wants to rule out *one* result, so $\mathcal{P}$ and $\mathcal{V}$ can execute Steps 2-4 recursively. This reduces the work for $\mathcal{V}$ but increases the work for $\mathcal{P}$. However, the PoEs can be batched together similarly to the batching protocol in [20] so the proof size only grows by one group element.

– If $\mathcal{V}$ and $\mathcal{P}$ find out in Step 3 that $\tilde{\mu}_1^{2^{\lambda \log n}} = 1$, they also know the smallest integer i such that $\tilde{\mu}_1^{2^{\lambda \log n - i}} \neq 1$. This means that, in Step 4, $\mathcal{P}$ can send a PoE for the statement

$$(x^k)^{2^{n-1-\lambda \log n+i}} = y \mod N,$$

and $\mathcal{V}$ only needs to check if $y^{2^{\lambda \log n - i}} = -\mu \mod N$. This reduces the work for $\mathcal{V}$ by i multiplications.

For simplicity of the analysis and because the improvements are minor, we omit these changes and analyze the protocol as it is stated in Figure 3.

Instance: (n, k, x, μ), where $n \in \mathbb{N}$, $0 < k < 2^{n-1}$ an odd integer, $\mu \in \mathbb{Z}_N^*$ with $\mu \neq 1$ and $x \in \mathbb{Z}_N^*$ a small prime number with Jacobi symbol -1 modulo $N := k2^n + 1$

Parameters: statistical security parameter λ

Statement: $x^{k2^{n-1}} = -\mu \mod N$

Protocol:

1. $\mathcal{V}$ computes $a := N \mod x$ and if $a \neq 0$ the Jacobi symbol $(\frac{a}{x})$.
   - If $a = 0$, output `accept`.
   - If $a \neq 0$ and $(\frac{a}{x}) = -1$, go to Step 2.
   - Else, output `reject`.
2. $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{\mu}_1 := \mu^k \mod N$
   - If $\tilde{\mu}_1 \neq 1$, go to Step 3.
   - Else, $\mathcal{V}$ computes $d := \mathrm{ord}(\mu)$ and $a := 2^{-n} \mod d$. If $x^k = \mu^{2a} \mod N$ output `accept`. Else, output `reject`.
3. $\mathcal{P}$ and $\mathcal{V}$ compute $\tilde{\mu}_2 := \tilde{\mu}_1^{2^{\lambda \log n}} \mod N$.
   - If $\tilde{\mu}_2 = 1$, go to Step 4.
   - Else, $\mathcal{P}$ sends $\texttt{PPoE}(x^k, -\mu, n-1, N)$. If the `PPoE` verifier accepts, output `accept`. Else, output `reject`.
4. (a) $\mathcal{P}$ sends a group element y and a `PPoE` $(x^k, y, n-1-\lambda \log n, N)$ for some $y \in \mathbb{Z}_N^*$. If the `PPoE` verifier rejects, output `reject`. Else, go to Step 4b.
   (b) $\mathcal{V}$ computes $\tilde{y} := y^{2^{\lambda \log n}} \mod N$. If $\tilde{y} = -\mu$ output `accept`. Else, output `reject`.

Fig. 3. The non-primality certificate.

4.1 Completeness

In this section, we show that $\mathcal{V}$ always outputs `accept` if $\mathcal{P}$ is honest.

Theorem 2. *The protocol in Figure 3 has perfect completeness.*

Proof. We show that if $\mathcal{P}$ is honest, $\mathcal{V}$ does not output `reject` in any step and outputs `accept` in one of the steps.

Step 1. Assume that x does not divide N since otherwise $\mathcal{V}$ accepts in the first step and completeness holds trivially. If $\mathcal{P}$ is honest, x has Jacobi symbol $(\frac{x}{N}) = -1$. Furthermore, since x is prime and does not divide N, we have

$$\left(\frac{x}{N}\right) = (-1)^{(x-1)k2^n/4}\left(\frac{N}{x}\right) = \left(\frac{N}{x}\right) = \left(\frac{N \mod x}{x}\right),$$

where the first equality follows from the law of quadratic reciprocity. Hence, $\mathcal{V}$ does not reject in this step and goes on to the next one.

Step 2. If $\tilde{\mu}_1 \neq 1$, $\mathcal{V}$ does not output anything in this step and goes on to the next one. Assume $\tilde{\mu}_1 = 1$ and let $a := 2^{-n} \mod d$, where d is the order of μ. Then we have

$$\mu^{2a} = (\mu^2)^{2^{-n}} = ((x^k)^{2^n})^{2^{-n}} = x^k \mod N$$

so $\mathcal{V}$ accepts if $\mathcal{P}$ is honest.

Step 3. If $\tilde{\mu_2} = 1$, $\mathcal{V}$ does not output anything in this step and goes on to the next one. If $\tilde{\mu_2} \neq 1$, completeness follows immediately from the completeness property of `PPoE`.

Step 4. If $\mathcal{P}$ is honest, the verifier does not reject in Step 4a by the completeness property of `PPoE`. In Step 4b, the verifier checks if

$$y^{2^{\lambda \log n}} = (x^{k2^{n-1-\lambda \log n}})^{2^{\lambda \log n}} = -\mu \mod N,$$

which holds if $\mathcal{P}$ is honest.

4.2 Soundness

For our purposes, it is sufficient to consider a relaxed definition of soundness. We only want to rule out the event that a malicious prover $\tilde{\mathcal{P}}$ can convince $\mathcal{V}$ that a Proth number is not prime even though it is. This means we do not need to care about a cheating prover that convinces $\mathcal{V}$ of a wrong result of the exponentiation $x^{k2^{n-1}} \mod N$ as long as the correct result is not -1.

Definition 3. *We call a non-primality certificate* sound *if the probability that $\mathcal{V}$ outputs `accept` on a statement (n, k, x, μ) for some $\mu \neq 1$ but $x^{k2^{n-1}} = -1 \mod N$ is negligible. We call that probability the* soundness error.

Theorem 3. *The protocol in Figure 3 has soundness error at most $2^{-\lambda+2} \log n$.*

Proof. We bound the probability that $\mathcal{V}$ falsely accepts an incorrect statement in each step individually.

Step 1. If $\mathcal{V}$ accepts in Step 1, x is a divisor of N so N must be composite. Assume that this does not hold and x has Jacobi symbol 1 modulo N, i.e., $(\frac{x}{N}) = 1$. Then,

$$\left(\frac{a}{x}\right) = \left(\frac{N}{x}\right) = (-1)^{(x-1)k2^n/4} \left(\frac{x}{N}\right) = 1,$$

so $\mathcal{V}$ rejects in Step 1.

Step 2. Recall that we consider a relaxed definition of soundness (see Definition 3). This means that we only need $\mathcal{V}$ to reject, when the correct result of the exponentiation is -1. We show that if this is the case and $\tilde{\mu}_1 = 1$, $\mathcal{V}$ always rejects in Step 2(b). Assume that N is prime. If $\mathcal{V}$ gets to Step 2(b), we know that d is a divisor of k and hence odd. This means that μ^2 has order d. $\mathcal{V}$ computes $a := 2^{-n} \mod d$ and checks if $x^k = (-\mu)^{2a} = \mu^{2a} \mod d$.

1. Since N is prime it holds that $(x^k)^{2^{n-1}} = -1 \mod N$ so the order of x is $2^n k$. This means that the order of x^k is 2^n.
2. On the other hand, we know that the order of μ^2 is d so the order of μ^{2a} is a divisor of d and hence odd.

1. and 2. together yield that $x^k \neq \mu^{2a} \mod N$ so the verifier rejects.

Step 3. If $\mathcal{V}$ gets to Step 3 and $\tilde{\mu}_2 \neq 1$, we know that the order of the bad element μ is divisible by $2^{\lambda \log T}$. This means that a malicious prover convinces $\mathcal{V}$ to falsely accept if the execution of the PoE reduces the order of the bad element on average by 2^λ per round. In particular, there must be at least one round where the order drops by at least 2^λ. By Corollary 2, this happens with probability at most $2^{-\lambda+2}$ for a fixed round. Applying a union bound, we conclude that, in this case, $\mathcal{V}$ accepts with probability at most $2^{-\lambda+2} \log n$.[16]

Step 4. If $\mathcal{V}$ gets to Step 4, a malicious prover needs to cheat in Step 4 (a) since otherwise the check in Step 4 (b) will not go through. This means that the prover needs to multiply the claim in Step 4 (a) by a bad element α. What can we say about the order of α? We know that it needs to pass the following check:

$$(y\alpha)^{2^{\lambda \log T}} = -\mu,$$

where $y^{2^{\lambda \log T}} = -1$. This means that α is of the following form:

$$\alpha^{2^{\lambda \log T}} = \mu.$$

It is well known that

$$\mathrm{ord}(\alpha^{2^i}) = \frac{\mathrm{ord}(\alpha)}{\gcd(2^i, \mathrm{ord}(\alpha))} = \mathrm{ord}(\mu).$$

[16] PrimeGrid has already implemented a check $\mu^{k \cdot 2^{64}} = 1$? [3]. Our analysis shows that an exponent of 64 is not sufficient for cryptographic soundness as this only gives $64/\log(n)$ bits of security "per round"; once we apply the Fiat-Shamir methodology to make the proof non-interactive, each round can be attacked individually.

Output in	Prover's complexity	Verifier's complexity	Proof size
Step 1	0	$\log n$	0
Step 2	$1.5 \log k$	$2.5 \log k + 2 \log n$	0
Step 3	$1.5 \log k + \lambda \log n + 2\sqrt{n}$	$1.5 \log k + (4\lambda + 1) \log n$	$\log n$
Step 4	$1.5 \log k + \lambda \log n + 2\sqrt{n}$	$1.5 \log k + (5\lambda + 1) \log n$	$\log n + 1$

Table 1. Complexity of the protocol in Figure 3 depending on the step in which it outputs the result. Prover's and Verifier's complexity are measured in the number of multiplications and proof-size in the number of group elements. We denote by λ the statistical security parameter.

(A proof can be found in any standard textbook on group theory, e.g., [13, Proposition 5]). Now the order of μ is even so we know that $\mathrm{ord}(\alpha) = 2^i \mathrm{ord}(\mu)$ for $i = \lambda \log T$. In particular, we have that $2^{\lambda \log T}$ is a divisor of the order of α. We can apply Corollary 2 and a union bound by the same argument as above and conclude that $\mathcal{V}$ accepts with probability at most $2^{-\lambda+2} \log n$ in this case.

All the cases together show that $\mathcal{V}$ outputs `accept` with probability at most $2^{-\lambda+2} \log n$ whenever N is prime.

Corollary 3. *The Fiat-Shamir transform of the protocol in Figure 3 yields a statistically sound non-interactive protocol in the random oracle model: The probability that $\mathcal{P}$ finds a non-primality certificate for a prime number with up to Q random oracle queries is at most $Q2^{-\lambda+2}$.*

Proof. As we have seen in the proof of Theorem 3, a cheating prover $\tilde{\mathcal{P}}$ can convince $\mathcal{V}$ to accept a proof of non-primality of a prime number only if $\tilde{\mathcal{P}}$ manages to decrease the order of the bad element by at least 2^λ in one of the rounds of a `PPoE`. By Corollary 2, this happens with probability at most $2^{-\lambda+2}$, where the probability depends only on the random coins. Assume that $\tilde{\mathcal{P}}$ makes up to Q queries to the random oracle. By the union bound, the probability that $\tilde{\mathcal{P}}$ finds a query that triggers the above event is at most $Q2^{-\lambda+2}$.

4.3 Efficiency

In this section, we analyze the complexity of the Fiat-Shamir transform of the protocol presented in Figure 3. Note that this complexity depends on the step in which the protocol returns the output. We summarize the results of this section in Table 1.

Prover's complexity. We compute the number of multiplications the prover has to perform additionally to finding a quadratic non-residue modulo N and computing the initial exponentiation.

Step 1. If the protocol returns the output in Step 1, $\mathcal{P}$ does not perform any additional computations.

Step 2. $\mathcal{P}$ checks if $\mu^k = 1$ via "square and multiply", which is approximately $1.5 \log k$ multiplications. If this holds, $\mathcal{P}$ does not perform any other computations.

Step 3. If the protocol runs until Step 3, $\mathcal{P}$ has checked if $\mu^k = 1$, which did not hold and now checks if $(\mu^k)^{2^{\lambda \log n}}$, which is $\lambda \log n$ additional multiplications. If this holds, $\mathcal{P}$ computes the proof of $\texttt{PPoE}(x^k, -\mu, n-1, N)$ which, by Proposition 1, can be done with $2\sqrt{n}$ multiplications and storage of $\sqrt{n}$ group elements.

Step 4. If the protocol runs until Step 4, $\mathcal{P}$ has checked if $\mu^k = 1$ and $(\mu^k)^{2^{\lambda \log n}}$, which did not hold. Now $\mathcal{P}$ computes the proof of $\texttt{PPoE}\ (x^k, y, n-1-\lambda \log n, N)$, which, by Proposition 1, can be done with $2\sqrt{n - \lambda \log n}$ multiplications and storage of $\sqrt{n - \lambda \log n}$ group elements.

Verifier's complexity.

Step 1. Computing $a := k2^n+1 \mod x$ takes approximately $\log n$ multiplications. Computing the Jacobi symbol $(\frac{a}{x})$ takes approximately $\log^2 x$ multiplications. Since x is a very small prime number in practice, we will ignore the $\log^2 x$ multiplications from now on.

Step 2. $\mathcal{V}$ checks if $\mu^k = 1$ via "square and multiply", which is approximately $1.5 \log k$ multiplications. If this holds, $\mathcal{V}$ computes $2^{-n} \mod d$, where d is the order of μ. This is another $\log n + \log k$ multiplications.

Step 3. If the protocol runs until Step 3, $\mathcal{V}$ has checked if $\mu^k = 1$, which did not hold and now checks if $(\mu^k)^{2^{\lambda \log n}}$, which is $\lambda \log n$ additional multiplications. If this holds, $\mathcal{V}$ verifies the proof of $\texttt{PPoE}(x^k, -\mu, n-1, N)$ which is $3\lambda \log n$ multiplications by Proposition 1.

Step 4. If the protocol runs until Step 4, $\mathcal{V}$ has checked if $\mu^k = 1$ and $(\mu^k)^{2^{\lambda \log n}}$, which did not hold. Now $\mathcal{V}$ verifies the proof of $\texttt{PPoE}\ (x^k, y, n-1-\lambda \log n, N)$, which is $3\lambda \log(n - \lambda \log n)$ multiplications (by Proposition 1) and then performs an exponentiation with exponent $2^{\lambda \log n}$, which is another $\lambda \log n$ multiplications.

Proof size.

Step 1. If $\mathcal{V}$ already accepts or rejects in Step 1, there is no proof needed.

Step 2. If $\mu^k = 1$, $\mathcal{V}$ can check the result themselves so there is no proof needed.

Step 3. If $\mathcal{P}$ sends a proof in this step, the proof size is equal to the size of the proof of $\texttt{PPoE}(x^k, -\mu, n-1, N)$, which is $\log(n-1)$ by Proposition 1.

Step 4. If $\mathcal{P}$ sends a proof in this step, it consists of a group element y and the proof of $\texttt{PPoE}\ (x^k, y, n-1-\lambda\log n, N)$, which is $\log(n-1-\lambda\log n)$ by Proposition 1.

Example. We give a numerical example of the complexity of the protocol when it outputs the result in Step 4 (the most expensive case) using the largest Proth prime known to date: $10223 \cdot 2^{31172165}+1$ [31]. For $k = 10223$ and $n = 31172165$ we have $\lceil \log k \rceil = 14$ and $\lceil \log n \rceil = 25$. If we choose the security parameter as $\lambda = 80$, we get that the prover stores $\lceil \sqrt{31172165} \rceil = 5584$ group elements, performs 13188 multiplications, the verifier performs 10046 multiplications and the proof size is 26 elements of size 31172179, i.e., around 102 MB. Note that recomputing the result of the primality test would take $n = 31172165$ multiplications in the same group, so our protocol reduces the number of multiplications by a multiplicative factor of $\lfloor 31172165/(13188+10046) \rfloor = 1341$.

Our protocol also achieves significant savings compared to [6]: To compute the PoE of [6], the prover needs to perform $2\lambda\sqrt{n} = 893312$ multiplications and the verifier does $2\lambda^2 \log n + 2\lambda = 318800$ multiplications to verify the proof consisting of $\lambda \log n = 2080$ group elements (i.e. 8160 MB). This means that our protocol reduces the number of multiplications of [6] by a factor of 52 and the proof size by a factor of 80.

4.4 Comparison with Pietrzak's PoE

We saw in Section 4.3 that the complexity of the protocol is the highest, when it outputs the result in Step 4. Even in this case the additional cost compared to the naive implementation of PPoE is moderate: Instead of performing $2\sqrt{n}$ multiplications, $\mathcal{P}$ needs $1.5\log k + \lambda \log n + 2\sqrt{n}$ multiplications to compute the proof. Using the numbers from the example in Section 4.3 this is 2021 extra multiplications on top of the 11168 multiplications that are performed in the naive implementation. Instead of performing $3\lambda \log n$ multiplications, $\mathcal{V}$ needs $1.5\log k + (5\lambda+1)\log n$ multiplications to verify the result. This is 4046 additional multiplications to the 6000 multiplications of the naive implementation in our example. The proof size grows by one group element from $\log n$ to $\log n + 1$. In our example, this corresponds to a proof size of 102 MB instead of 98 MB. If the protocol outputs the result in Step 1 or Step 2 it is even more efficient than PPoE. Recall that the implementation of PPoE in groups of known order does not have any soundness guarantees and for the groups that we are using, there are known attacks that break soundness. In contrast, we showed in Section 4.2 that our protocol is statistically sound in these groups. We conclude that our protocol yields a major soundness improvement at moderate additional costs.

5 Open Problems

In this work, we presented an efficient protocol that gives a certificate of non-primality for Proth numbers. While we believe that a certificate of non-primality

is more useful than a certificate of primality in the context of search for giant primes, constructing the latter is certainly an intriguing problem. Though, our techniques are not directly applicable to prove *primality* of Proth numbers because our protocol only rules out that the correct result is *one specific number* (namely -1). Conversely, when proving primality one has to rule out *all results except for one* (again -1). Constructing a cryptographic certificate of primality therefore remains an open problem.

Another open problem is to demonstrate the applicability of PoEs towards certifying (non-)primality of other types of numbers such as, for example, Mersenne numbers. The primality of Mersenne numbers is tested via Lucas Lehmer test amounting to computation of long modular recursive sequences. Equivalently, the test can be performed via exponentiation in a suitable extension ring and, thus, one could hope to employ PoEs also in the context of Mersenne number. However, there are some major differences to the case of Proth numbers. In particular, the order of the corresponding group is not necessarily efficiently computable even when the candidate is a prime, which is one of the issues preventing the use of our protocol.

Finally, can our interactive protocol be made non-interactive under assumptions other than random oracles? Several recent works [24,5] have aimed to derandomise Pietrzak's protocol and its closely-related variant from [6] using more standard cryptographic assumptions. It would be interesting to explore whether these techniques are applicable here.

Acknowledgements. We are grateful to Pavel Atnashev for clarifying via e-mail several aspects of the primality tests implementated in the PrimeGrid project. Pavel Hubáček is supported by the Czech Academy of Sciences (RVO 67985840), the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X, and by the Charles University project UNCE/SCI/004. Chethan Kamath is supported by Azrieli International Postdoctoral Fellowship, ISF grants 484/18 and 1789/19, and ERC StG project SPP: Secrecy Preserving Proofs.

References

1. Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.
2. A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61(203):29–68, 1993.
3. Pavel Atnashev. Personal communication, 2022. February 2022.
4. Robert Baillie and Samuel S. Wagstaff. Lucas pseudoprimes. *Mathematics of Computation*, 35(152):1391–1417, 1980.
5. 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. TCC 2022, to appear, 2022.
6. Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021,*

Part IV, volume 12828 of *Lecture Notes in Computer Science*, pages 123–152, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

7. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.
8. Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.
9. B. Borsos, A. Kovács, and N. Tihanyi. Tight upper and lower bounds for the reciprocal sum of proth primes. *Ramanujan J*, 59:181–198, 2022.
10. John Brillhart, D. H. Lehmer, and J. L. Selfridge. New primality criteria and factorizations of $2^m \pm 1$. *Mathematics of Computation*, 29(130):620–647, 1975.
11. Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, June 1988.
12. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.
13. David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley and Sons, 3rd edition, 2003.
14. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.
15. Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, March 2000.
16. Great Internet Mersenne Prime Search GIMPS. GIMPS discovers largest known prime number: $2^{82,589,933} - 1$. https://www.mersenne.org/primes/press/M82589933.html, 2018. Accessed: 2022-05-18.
17. Great Internet Mersenne Prime Search GIMPS. Prime95 v30.3. https://www.mersenneforum.org/showthread.php?t=25823, 2020. Accessed: 2022-05-19.
18. Shafi Goldwasser and Joe Kilian. Almost all primes can be quickly certified. In *18th Annual ACM Symposium on Theory of Computing*, pages 316–329, Berkeley, CA, USA, May 28–30, 1986. ACM Press.
19. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
20. Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Practical statistically-sound proofs of exponentiation in any group. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 370–399, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.
21. Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 637–653, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

22. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 723–732, Victoria, BC, Canada, May 4–6, 1992. ACM Press.
23. D. H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bulletin of the American Mathematical Society*, 33(3):327 – 340, 1927.
24. Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 632–651, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.
25. Edouard Lucas. Théorie des fonctions numériques simplement périodiques. *American Journal of Mathematics*, 1(4):289–321, 1878.
26. Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.
27. Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.
28. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15, San Diego, CA, USA, January 10–12, 2019. LIPIcs.
29. Carl Pomerance, J. L. Selfridge, and Samuel S. Wagstaff. The pseudoprimes to $25 \cdot 10^9$. *Mathematics of Computation*, 35(151):1003–1026, 1980.
30. Vaughan R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975.
31. PrimeGrid. World record Colbert number discovered! http://www.primegrid.com/forum_thread.php?id=7116, 2016. Accessed: 2022-05-18.
32. PrimeGrid. Proposal: A new sierpiński problem. https://www.primegrid.com/forum_thread.php?id=9107, 2020. Accessed: 2022-05-19.
33. François Proth. Theoremes sur les nombres premiers. *Comptes rendus de l'Académie des Sciences de Paris*, 87(926), 1878.
34. Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.
35. Hans Riesel. Lucasian criteria for the primality of $N = h \cdot 2^n - 1$. *Mathematics of Computation*, 23(108):869–875, 1969.
36. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems (reprint). *Commun. ACM*, 26(1):96–99, 1983.
37. Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 382–414, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.
38. R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.
39. Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, October 2020.

Instance: $(x, y, T, \mathbb{G})$, where $x, y \in \mathbb{G}$, $T \in \mathbb{N}$ is even and $x^{2^T} = y$ in $\mathbb{G}$

Input to $\mathcal{P}^*$: $\alpha \in \mathbb{G}$

Parameters: statistical security parameter λ

Statement: $x^{2^T} = y\alpha$ in $\mathbb{G}$

Protocol:

1. For $T = 1$:
   - If $x^2 = y$, $\mathcal{V}$ outputs `accept`.
   - Else, $\mathcal{V}$ outputs `reject`.
2. For $T > 1$:
   (a) $\mathcal{P}^*$ sends $v = \alpha^{-1} x^{2^{T/2}}$ to $\mathcal{V}$.
   (b) If $v \notin \mathbb{G}$, $\mathcal{V}$ outputs `reject`. Otherwise, $\mathcal{V}$ samples $r \leftarrow \{0, 1, \ldots, 2^\lambda - 1\}$ uniformly at random and sends it to $\mathcal{P}^*$.
   (c) $\mathcal{P}^*$ and $\mathcal{V}$ compute $x' := x^r v$ and $y' := v^r y$ in $\mathbb{G}$.
   (d) If $T/2$ is even, $\mathcal{P}^*$ and $\mathcal{V}$ run the protocol on instance $(x', y', T/2, \mathbb{G})$ with input $\alpha^{2^{T/2}-r-1}$ to $\mathcal{P}^*$. If $T/2$ is odd, $\mathcal{P}^*$ and $\mathcal{V}$ run the protocol on instance $(x', y'^2, (T+1)/2, \mathbb{G})$ with input $\alpha^{2(2^{T/2}-r-1)}$ to $\mathcal{P}^*$.

Fig. 4. An attack with success probability at least $1 - (1 - 1/\operatorname{ord}(\alpha))^{\log T}$.

A Attacking Pietrzak's Protocol in Proth Number Groups

In this section we show how a malicious prover $\mathcal{P}^*$ can falsely convince the verifier $\mathcal{V}$ that a Proth *prime* is composite when using Pietrzak's PoE. This attack was first described in [8]. Let $N = k2^n + 1$ be prime and x be any quadratic non-residue modulo N. Since N is prime, it holds that $x^{k2^{n-1}} = -1 \mod N$. The easiest way for $\mathcal{P}^*$ to cheat is claiming that the result of this exponentiation is 1 instead of -1 and then multiplying the honest messages by -1 until the recombination step (Step 2d of `PPoE`) yields a correct instance. The probability that $\mathcal{V}$ accepts this false "proof" of non-primality is $1 - 1/2^{\log(n-1)} = 1 - (n-1)^{-1}$. To see this, consider the first round of the protocol. $\mathcal{P}^*$ multiplies the correct midpoint $v = (x^k)^{2^{(n-1)/2}}$ by -1 and sends the message $-v$ to $\mathcal{V}$. $\mathcal{V}$ samples a random coin r and they both compute $x' = -x^{kr}v$ and $y' = (-v)^r$ to create the new statement $x'^{2^{(n-1)/2}} = y'$. Plugging in the values for x', y' and v, we see that the new statement is correct whenever r is an odd integer:

$$\begin{aligned} x'^{2^{(n-1)/2}} &= y' \\ \Leftrightarrow (-x^{kr}v)^{2^{(n-1)/2}} &= (-v)^r \\ \Leftrightarrow v^{2^{(n-1)/2}} &= (-1)^r \\ \Leftrightarrow (x^k)^{2^{n-1}} &= (-1)^r. \end{aligned}$$

If r is even, the statement remains false and $\mathcal{P}^*$ does the same in the next round. $\mathcal{V}$ only outputs `reject` if all of the random coins are even which happens with probability $1/2^{\log(n-1)} = (n-1)^{-1}$ since $\log(n-1)$ is the number of rounds. Otherwise $\mathcal{V}$ outputs `accept` on a false statement. A generalization of this attack is shown in Figure 4. Instead of multiplying the correct statement by -1, $\mathcal{P}^*$ multiplies the correct statement by an arbitrary group element α and adapts its messages accordingly. The success probability can be lower bounded by $1 - (1 - 1/\operatorname{ord}(\alpha))^{\log(n-1)}$ which is the probability that in at least one round the bad element is raised to a multiple of the order of α. If $\operatorname{ord}(\alpha)$ is not a prime number, this bound is not tight since the order of the bad element can decrease during the execution of the rounds, making the success probability even higher. In the case where N is prime, the prover knows the group order $N-1 = k2^n$ and its factorization and can therefore construct elements of sufficiently low order.