2022 S.T. Yau High School Science Award

Research Report

The Team:

Student: Darren Li
Shanghai Guanghua Academy
Shanghai, China

Supervising Teacher: Yves Gallot
École Supérieure d'Electricité, SUPELEC (Retired)
Toulouse, France

Title of Research Report:

An Efficient Modular Exponentiation Proof Scheme

Date: 2022/09/01

For Yves Gallot's research, please visit:
https://www.researchgate.net/profile/Yves-Gallot/research

An Efficient Modular Exponentiation Proof Scheme

Darren Li

Supervisor: Yves Gallot

Abstract

We present an efficient proof scheme for any instance of left-to-right modular exponentiation, used in the Fermat probable prime test. Specifically, we show that for any (a, n, r, m) the claim $a^n \equiv r \pmod{m}$ can be proven and verified with an overhead negligible compared to the computational cost of the exponentiation. Our work generalizes the Gerbicz-Pietrzak double check scheme, greatly improving the efficiency of general probabilistic primality tests in distributed searches for primes such as PrimeGrid.

Keywords: Distributed computing, Primality testing, Proof schemes, Forking arguments

Contents

1 Introduction

In distributed computing, the task of efficiently discerning correct results from incorrect results, whether malicious or due to sheer chance, is notoriously difficult; it is necessary to suspect all results as possibly incorrect, and these suspicions can only be settled with peer verification. In most cases, such verification would take the form of an entire recomputation, doubling the amount of necessary computational power, all for a result that is most likely correct.

The majority of ongoing organized searches for primes of record-breaking sizes, such as PrimeGrid [7], are distributed. A significant portion of computing power is wasted in the double check process, where a single primality test is done twice to ensure the correctness of each result.

For a candidate prime m, the bottleneck of the Fermat probable prime test lies in evaluating $a^{m-1} \pmod{m}$ or a nearby power, requiring $\tilde{O}(\log^2 m)$ exact computations, where a single precision error will render the rest of the calculation incorrect. To prevent these errors that could potentially categorize a prime as composite, verification of all results are necessary.

With the development of the Pietrzak verifiable delay function [5], it became possible to instead use *certificates* - proofs of the result of modular exponentiation, thereby proving compositeness or probable primality - that could be created with minimal overhead and verifiable much faster than recomputing the test. However, both methods rely on the fact that for certain candidates, the desired answer can be achieved through only repeated squaring, which is not true in the general case.

We present an optimization of the verification process by creating a certificate for the Fermat probable prime test of *any* candidate prime, halving the total CPU time necessary to conduct a Fermat probable prime test on a distributed system. Such a certification scheme is particularly useful to accelerate the current search for Generalized Fermat primes, a form introduced for its unprecedentedly fast algorithms and unique GPU implementations [3] yet one of the few so far unamenable to the Gerbicz double check scheme.

1.1 Previous work

The Gerbicz double check scheme, initially derived as an error check by Robert Gerbicz and modified to a certification scheme by Pavel Atnashev [6], is used when the desired result can be derived from a^{2^n}. This is the case for (probable) prime tests for candidates of the form $k \cdot 2^n - 1$ and for Proth tests for primes of the form $k \cdot 2^n + 1$, where k is relatively small.

The Gerbicz double check scheme uses a list of "checkpoints" $C_1, C_2, \ldots$, where $C_i = a^{2^{iB}}$ for some constant B. It then exploits how we always have $C_{i+1} = C_i^{2^B}$ for all i. During verification, the Gerbicz double check scheme takes all required equivalences, takes each to a random exponent, and checks that the product of all left hand sides equal the product of all right hand sides.

When implemented in a method similar to the Pietrzak verifiable delay function, the Gerbicz double check scheme creates a certificate of $O(\log((\log n)/B))$ residues and takes B squarings to verify. The Gerbicz-Pietrzak double check scheme has been successfully implemented in the PrimeGrid distributed computing project since 2020. Although the Pietrzak verifiable delay function was originally proved to have unconditional soundness only when the modulus is the product of two safe primes, assuming the low order assumption, conditional soundness holds for all multiplicative groups [4].

1.2 Our contribution

For our purposes, the Gerbicz-Pietrzak construction is not applicable due to inhomogeneous relations between checkpoints. We present a double check scheme similar to that of Gerbicz, and we extend it to a certification scheme with a divide-and-conquer structure similar to the Pietrzak verifiable delay function, completing a practical and sound proof scheme for modular exponentiation in general.

The new construction can be further generalized into the verification of an individual intervals of steps of the left-to-right modular exponentiation process, allowing for further division of the Fermat probable prime test process, to the point where it becomes feasible to distribute steps across multiple computers for world-record level probable prime tests. We have incorporated our described certificate construction to new versions of the Genefer source code.

2 Double check process

We first outline the core idea of our certification scheme, the double check process. Taking inspiration from Gerbicz, we similarly save checkpoints through the process of left-to-right modular exponentiation. Suppose a is the base, n is the exponent, and m is the modulus of the left-to-right modular exponentiation process that we wish to certify. Let

$$n = n_0 2^0 + n_1 2^1 + \cdots + n_{L-1} 2^{L-1}$$

be the binary expansion of n, i.e. $n_i \in \{0, 1\}$. Left-to-right modular exponentiation is the calculation of the sequence $u_i = a^{\lfloor \frac{n}{2^i} \rfloor}$, where

$$u_i = \begin{cases} 1 & L \leq i \\ u_{i+1}^2 \cdot a^{n_i} & \text{otherwise} \end{cases}$$

Our answer is $a^n \equiv u_0 \bmod m$. This computation requires L squarings and at most L multiplications by a, the latter of which is cheaper than a full multiplication.

u_i must satisfy that

$$u_i = a^{\left\lfloor \frac{n}{2^i} \right\rfloor} = a^{\left\lfloor \left\lfloor \frac{n}{2^i} \right\rfloor / 2^j \right\rfloor 2^j + \left(\left\lfloor \frac{n}{2^i} \right\rfloor \bmod 2^j \right)} = a^{\left\lfloor \frac{n}{2^{i+j}} \right\rfloor 2^j} a^{\left\lfloor \frac{n}{2^i} \right\rfloor \bmod 2^j} = u_{i+j}^{2^j} a^{\left\lfloor \frac{n}{2^i} \right\rfloor \bmod 2^j}$$

Saving $u_0, u_B, u_{2B}, \ldots$ for some constant B, a double check can then be expressed as

$$\begin{aligned} \prod_l u_{lB}^{w_l} &\stackrel{?}{=} \prod_l u_{(l+1)B}^{w_l 2^B} a^{\left(\left\lfloor \frac{n}{2^{Bl}} \right\rfloor \bmod 2^B \right) w_l} \\ &= \left(\prod_l u_{(l+1)B}^{w_l} \right)^{2^B} x^{\sum_l \left(\left\lfloor \frac{n}{2^{Bl}} \right\rfloor \bmod 2^B \right) w_l} \end{aligned}$$

where w_i is randomly generated during the double check, indicating the weight of the i-th checkpoint. This method reduces the cost of the double check from L multiplications to B multiplications.

3 Interactive proof scheme

A direct implementation of the above method, as a proof scheme, requires a prohibitive amount of bandwidth between the prover and the verifier, as such a certificate will need to contain $u_0, u_B, u_{2B}, \ldots$ for a total of L/B residues, amounting to several gigabytes in cutting-edge primality tests. To ensure the practicality of our method, we describe a certificate construction, better understood as a special assignment to the weights described above, that reduces the size of the certificate to $\log(L/B)$ residues.

In the following parts, we define that

$$\begin{aligned} S(x,y) &= a^{\sum_{i=x}^{x+y-1} n_i 2^{i-x}} \\ &= a^{\left\lfloor \frac{n}{2^x} \right\rfloor \bmod 2^y} \end{aligned}$$

Let $P(i,L)$ be the *claim* that $u_{iL} = u_{(i+1)L}^{2^L} S(iL, L)$. $P(i,L)$ is effectively the assertion that the step from u_{iL} to $u_{(i+1)L}$ has no errors. For a valid u sequence, all P claims are true.

3.1 Outline of our construction

We first describe an informal approach to our proof process; let us interpret claims to be multiplicative; for example, $P(a,L)^b P(c,L)^d$ is a shorthand for

$$u_{aL}^b u_{cL}^d = \left(u_{(a+1)L}^b u_{(c+1)L}^d\right)^{2^L} S(aL,L)^b S(cL,L)^d$$

Initially, the prover seeks to prove $P(0, 2^x)$ for some $2^x > L$. To prove $P(a, 2L)$, it is sufficient to prove $P(2a, L)$ and $P(2a+1, L)$. During the interactive proof, the prover and verifier iteratively reduces the size of a claim equivalent to $P(0, 2^x)$. Suppose that the prover currently has a claim $\mathbf{A} = \prod_{i=1}^{c-1} P(i, 2^L)^{w_i}$ for weights w_i. Expanding gives

$$\underbrace{\left[\prod u_{i2^L}^{w_i}\right]}_{\mathbf{A}_1} = \underbrace{\left[\prod u_{(i+1)2^L}^{w_i}\right]}_{\mathbf{A}_2}^{2^{(2^L)}} \cdot \underbrace{\left[\prod S(i2^L, 2^L)^{w_i}\right]}_{\text{Known.}} \qquad (\mathbf{A})$$

The verifier can check the current claim in 2^L squarings. To halve the number of squarings, we decompose $\mathbf{A}$ by the parity of i in $P(i, 2^L)$.

Define claim $\mathbf{B}$ as $\prod_{i=0}^{c-1} P(2i, 2^{L-1})^{w_i}$ and claim $\mathbf{C}$ as $\prod_{i=0}^{c-1} P(2i+1, 2^{L-1})^{w_i}$. Much like the Gerbicz process, the verifier randomly selects Q, after which the prover and verifier agrees on a reduction $\mathbf{A} \iff \mathbf{B} \wedge \mathbf{C} \iff \mathbf{A}' = \mathbf{B} \cdot \mathbf{C}^Q$. (We prove in Section 4 that, assuming the low order assumption in $\mathbb{Z}_p^\times$, for a security parameter λ, when Q is randomly sampled from from $\mathbb{N} \cup [1, 2^\lambda]$, $\mathbf{A}$ is equivalent to $\mathbf{B} \cdot \mathbf{C}^Q$ up to probability negligible in λ.)

Similarly defining $\mathbf{A}'_{1,2}$, $\mathbf{B}_{1,2}$, and $\mathbf{C}_{1,2}$, we now have:

$$\underbrace{\left[\prod u^{w_i}_{(2i+0)2^{L-1}}\right]}_{\mathbf{B}_1=\mathbf{A}_1} = \underbrace{\left[\prod u^{w_i}_{(2i+1)2^{L-1}}\right]}_{\mathbf{B}_2}^{2^{(2^{L-1})}} \cdot \left[\prod S((2i+0)2^{L-1}, 2^{L-1})^{w_i}\right] \tag{$\mathbf{B}$}$$

$$\underbrace{\left[\prod u^{w_i}_{(2i+1)2^{L-1}}\right]}_{\mathbf{C}_1} = \underbrace{\left[\prod u^{w_i}_{(2i+2)2^{L-1}}\right]}_{\mathbf{C}_2=\mathbf{A}_2}^{2^{(2^{L-1})}} \cdot \left[\prod S((2i+1)2^{L-1}, 2^{L-1})^{w_i}\right] \tag{$\mathbf{C}$}$$

Let $\mu = \mathbf{B}_2 = \mathbf{C}_1$, which is unknown to the verifier. Then, $\mathbf{A}'_1 = \mathbf{A}_1 \cdot \mu^Q$ and $\mathbf{A}'_2 = \mu \cdot \mathbf{A}_2^Q$. These values successfully form the new claim $\mathbf{A}' = \prod_{i=0}^{2c-1} P(i, 2^{L-1})^{w_{\lfloor i/2 \rfloor} Q^{i \bmod 2}}$.

When the prover provides μ to the verifier, the number of squarings necessary to verify its original claim is halved. This process is repeated as necessary. To optimize the cost of the interaction on the prover's part, the checkpointing rate does not need to be a power of two, as our method can be easily generalized to decomposing claims of the form $P(0, B2^x)$ where B is the precise checkpointing rate.

3.2 Formal process

We now present a formal description of the interaction between the prover and the verifier. Suppose the prover and the verifier have agreed beforehand on the security parameter λ, the modulus m, the checkpointing rate B, the base of the exponentiation a, the exponent n, and an integer x such that $n < 2^{B2^x}$. Define the formal language

$$\mathcal{L} = \left\{ (b, r, t, w_0, w_1, \ldots, w_{2^{x-t}-1}) : \begin{array}{c} 1 \le b, r < m, 0 \le t \le x \\ r \equiv b^{2^{B2^t}} \displaystyle\prod_{i=0}^{2^{x-t}-1} S(iB2^t, B2^t)^{w_i} \end{array} \right\}$$

The prover seeks to prove that $(1, r, x, 1) \in \mathcal{L}$ where $r \equiv a^n$.

Suppose that the prover claims $(b, r, t, w_0, w_1, \ldots, w_{2^{x-t}-1}) \in \mathcal{L}$.

1. If any of $1 \leq b, r < m, 0 \leq t \leq x$ are not satisfied, the verifier returns `reject`.
2. If $t = 0$, the verifier checks that

$$r \equiv b^{2^B} \prod_{i=0}^{2^x-1} S(iB, B)^{w_i} \equiv b^{2^B} a^c; c = \sum_{i=0}^{2^x-1} w_i \left(\left\lfloor \frac{n}{2^{iB}} \right\rfloor \bmod 2^B \right)$$

   and returns `accept` or `reject` accordingly.

3. Otherwise, the prover computes and sends to the verifier μ, where

$$\mu \equiv \prod_{i=0}^{2^{x-t}-1} u_{(2i+1)B2^{t-1}}^{w_i} = b^{2^{B2^{t-1}}} \prod_{i=0}^{2^{x-t}-1} S(2iB2^{t-1}, B2^{t-1})^{w_i}$$

4. The verifier computes a challenge Q randomly sampled from $\mathbb{N} \cup [1, 2^\lambda]$, and the prover and the verifier recurse on

$$(b\mu^Q, \mu r^Q, t-1, w_0, Qw_0, w_1, Qw_1, \ldots, w_{2^{x-t}-1}, Qw_{2^{x-t}-1})$$

To construct a non-interactive proof, or a certificate, of the result $r \equiv a^n$, it suffices to replace the verifier challenges with a hash of the current state by the Fiat-Shamir heuristic.

4 Proof of conditional soundness

We demonstrate that for *any* (λ, m, B, a, n, x), assuming the hardness of finding an element of $\mathbb{Z}_p^\times$ with order less than 2^λ, no adversary can forge a result and certificates with non-negligible probability with respect to λ.

Specifically, assume the contrary; suppose there exists a randomized polynomial time adversary $\mathcal{A}$ defined as

$$\mathcal{A}(\lambda, m, B, a, n, x; Q_x, Q_{x-1}, \ldots, Q_1) \to I_x, (\mu_x, I_{x-1}), (\mu_{x-1}, I_{x-2}), \ldots, (\mu_1, I_0)$$

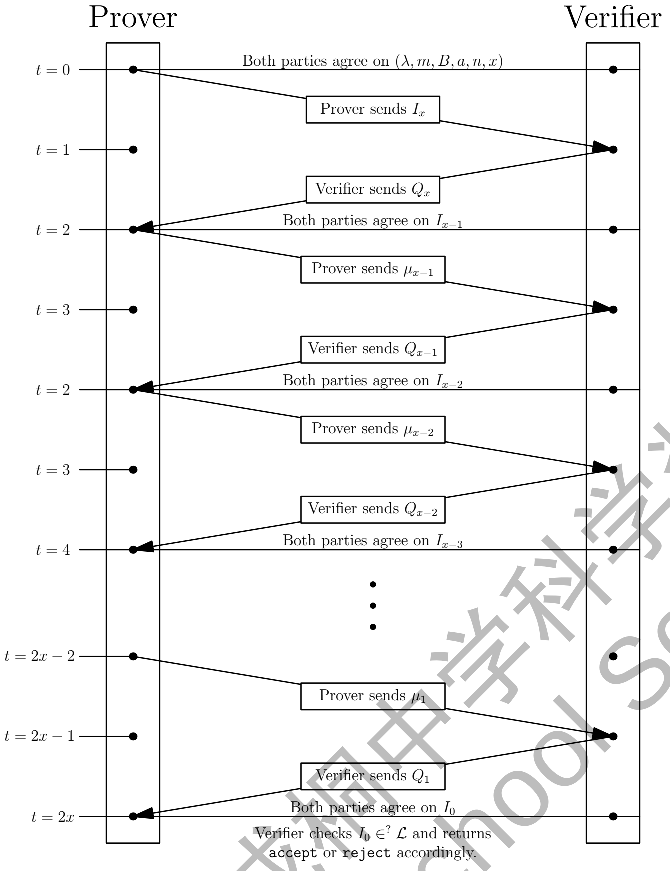

Figure 1: Interactive proof process

When given (λ, m, B, a, n, x) and randomly sampled $Q_x, Q_{x-1}, \ldots, Q_1 \leftarrow \mathbb{N} \cup [1, 2^\lambda]$, $\mathcal{A}$ attempts to generate an input $I_x = (1, r, x, 1)$ and an corresponding interaction $(\mu_x, I_{x-1}), (\mu_{x-1}, I_{x-2}), \ldots, (\mu_1, I_0)$ (see figure 1) and *succeeds* with probability non-negligible in λ.

We say that $\mathcal{A}$ *succeeds* if and only if

1. $I_x \notin \mathcal{L}$ and $I_0 \in \mathcal{L}$, i.e. $\mathcal{A}$ deceives a verifier with challenges $Q_x, Q_{x-1}, \ldots, Q_1$.
2. For all y and all $Q'_y, Q'_{y-1}, \ldots, Q_1 \leftarrow \mathbb{N} \cup [1, 2^\lambda]$, we have that when $\mathcal{A}$ is run with the same random tape (i.e. makes the same random decisions) it must hold that

$I_y = I'_y$ and $\mu_y = \mu'_y$, where

$$\mathcal{A}(\lambda, m, B, a, n, x; Q_x, \ldots, Q_{y+1}, Q_y, \ldots, Q_1) \to I_x, (\mu_x, I_{x-1}), (\mu_{x-1}, I_{x-2}), \ldots, (\mu_1, I_0)$$
$$\mathcal{A}(\lambda, m, B, a, n, x; Q_x, \ldots, Q_{y+1}, Q'_y, \ldots, Q'_1) \to I'_x, (\mu'_x, I'_{x-1}), (\mu'_{x-1}, I'_{x-2}), \ldots, (\mu'_1, I'_0)$$

The latter condition is necessary to ensure that $\mathcal{A}$ does not "look ahead" and base decisions (of I and μ) based on future challenges.

Therefore, we seek to prove that

Theorem 1. *For some (λ, m, B, a, n, x), if $\mathcal{A}$ succeeds with probability non-negligible with respect to λ, there exists an adversary that can use $\mathcal{A}$ twice to obtain an element of $\mathbb{Z}_p^\times$ with order less than 2^λ with non-negligible probability, breaking the low-order assumption.*

To this end we will prove two claims:

1. If, using $\mathcal{A}$, an adversary finds a state $I_y \notin \mathcal{L}$, a prover message μ_y, and two separate challenges Q_y, Q'_y such that the resulting states I_y and I'_y are both in $\mathcal{L}$, then the attacker can recover some element $E \not\equiv 1$ and some exponent $0 < r < 2^\lambda$ such that $E^r \equiv 1$, breaking the low order assumption.
2. If $\mathcal{A}$ succeeds with probability m, an adversary using $\mathcal{A}$ twice succeeds in finding (I_y, μ_y, Q_y, Q'_y) with probability at least $m(m/x - 2^{-\lambda})$.

Proof of first claim. When the adversary indeed succeeds in finding (I_y, μ_y, Q_y, Q'_y), we have some I_y such that $I_y \notin \mathcal{L}$ and two I_{y-1}, I'_{y-1}, caused by Q_y and Q'_y respectively, such that $I_{y-1}, I'_{y-1} \in \mathcal{L}$.

For a state $I = (b, r, t, w_0, w_1, \ldots, w_{2^{x-t}-1})$, define $R(I) = r$, $B(I) = b$, and $C(I)$ as

$$C(I) = \sum_{i=0}^{2^{x-t}-1} w_i \left(\left\lfloor \frac{n}{2^{iB2^t}} \right\rfloor \bmod 2^{B2^t} \right)$$

By $I_y \notin \mathcal{L}$, we have

$$R(I_y) \not\equiv B(I_y)^{2^{B2^y}} a^{C(I_y)}$$

or

$$\begin{aligned}
R(I_y) &\not\equiv \left(B(I_y)^{2^{B2^{y-1}}}\right)^{2^{B2^{y-1}}} a^{C(I_y)} \\
&= \left(B(I_y)^{2^{B2^{y-1}}}\right)^{2^{B2^{y-1}}} a^{\sum_{i=0}^{2^{x-y}-1} w_i\left(\left\lfloor \frac{n}{2^{iB2^y}} \right\rfloor \bmod 2^{B2^y}\right)} \\
&= \left(B(I_y)^{2^{B2^{y-1}}}\right)^{2^{B2^{y-1}}} a^{\sum_{i=0}^{2^{x-y}-1} w_i\left(\left\lfloor \frac{n}{2^{(2i+0)B2^{y-1}}} \right\rfloor \bmod 2^{B2^{y-1}}\right)} \\
&\qquad\qquad a^{2^{B2^{y-1}} \sum_{i=0}^{2^{x-y}-1} w_i\left(\left\lfloor \frac{n}{2^{(2i+1)B2^{y-1}}} \right\rfloor \bmod 2^{B2^{y-1}}\right)}
\end{aligned}$$

If we define c_1 and c_2 as

$$\begin{aligned}
c_1 &= \sum_{i=0}^{2^{x-y}-1} w_i\left(\left\lfloor \frac{n}{2^{(2i+0)B2^{y-1}}} \right\rfloor \bmod 2^{B2^{y-1}}\right) \\
c_2 &= \sum_{i=0}^{2^{x-y}-1} w_i\left(\left\lfloor \frac{n}{2^{(2i+1)B2^{y-1}}} \right\rfloor \bmod 2^{B2^{y-1}}\right)
\end{aligned}$$

it follows that at least one of the following does not hold:

$$R(I_y) \equiv \mu_y^{2^{B2^{y-1}}} a^{c_2} \tag{1}$$

$$\mu_y \equiv B(I_y)^{2^{B2^{y-1}}} a^{c_1} \tag{2}$$

By $I_{y-1}, I'_{y-1} \in \mathcal{L}$, for I_{y-1} have $R(I_{y-1}) \equiv B(I_{y-1})^{2^{B2^{y-1}}} a^{C(I_{y-1})}$. At the same time, our recursion $I_y \to I_{y-1}$ is defined with $B(I_{y-1}) = B(I_y)\mu_y^Q$, $R(I_{y-1}) = \mu_y R(I_y)^Q$, and $C(I_{y-1}) = c_1 + Qc_2$; expanding for I_y and I'_y gives

$$\begin{aligned}
\mu_y R(I_y)^Q &\equiv \left(B(I_y)\mu_y^Q\right)^{2^{B2^{y-1}}} a^{c_1+Qc_2} \\
\mu_y R(I_y)^{Q'} &\equiv \left(B(I_y)\mu_y^{Q'}\right)^{2^{B2^{y-1}}} a^{c_1+Q'c_2}
\end{aligned}$$

Rearranging,

$$\mu_y / \left(B(I_y)^{2^{B2^{y-1}}} a^{c_1}\right) \equiv \left(\mu_y^{2^{B2^{y-1}}} a^{c_2}/R(I_y)\right)^Q \equiv \left(\mu_y^{2^{B2^{y-1}}} a^{c_2}/R(I_y)\right)^{Q'}$$

If (1) is false, then $E^{|Q-Q'|} \equiv 1$ where

$$E = \mu_y^{2^{B2^{y-1}}} a^{c_2}/R(I_y) \not\equiv 1$$

for $0 < |Q - Q'| \leq 2^\lambda$, breaking the low-order assumption.

If (2) is false and (1) is true, then $\mu_y/\left(B(I_y)^{2^{B2^{y-1}}} a^{c_1}\right) \not\equiv 1$ while $\mu_y^{2^{B2^{y-1}}} a^{c_2}/R(I_y) \equiv \left(\mu_y^{2^{B2^{y-1}}} a^{c_2}/R(I_y)\right)^Q \equiv 1$; such a case is impossible.

This demonstrates that in the event of the adversary succeeding, an element E is generated with order less than 2^λ, breaking the low order assumption as required. □

It remains to analyze the probability of the adversary succeeding in finding the desired setting (I_y, μ_y, Q_y, Q'_y) with only two uses of $\mathcal{A}$.

Proof of second claim. The structure of our proof mirrors the forking argument used in "A Survey of Two Verifiable Delay Functions " [4]. Specifically, we reinterpret $\mathcal{A}$ as part of a new process $\mathcal{A}'$ that is more amenable to an application of the generalized forking lemma introduced by Bellare and Neven [2].

Let us abstract $\mathcal{A}$ as a probabilistic Turing machine with random tape R. Define $\mathcal{A}'(\lambda, m, B, a, n, x; Q_x, Q_{x-1}, \ldots, Q_1; R)$ to represent an execution of $\mathcal{A}$ with the given parameters and random tape R; $\mathcal{A}'$ returns $(\epsilon, \epsilon, \epsilon, \epsilon)$ if $\mathcal{A}$ fails, and otherwise outputs (y, I_y, μ_y, I_{y-1}) where $y = \arg\min(y : I_y \notin \mathcal{L})$.

For a given (λ, m, B, a, n, x), the adversary then proceeds as follows:

1. The adversary randomly samples $Q_x, Q_{x-1}, \ldots, Q_1 \leftarrow \mathbb{N} \cup [1, 2^\lambda]$ and generates a random tape R.
2. The adversary executes $y, I_y, \mu_y, I_{y-1} \leftarrow \mathcal{A}'(\lambda, m, B, a, n, x; Q_x, Q_{x-1}, \ldots, Q_1; R)$, and if $y = \epsilon$ the adversary fails.
3. The adversary randomly samples $Q'_y, Q'_{y-1}, \ldots, Q'_1 \leftarrow \mathbb{N} \cup [1, 2^\lambda]$.
4. The adversary executes

$$y', I'_y, \mu'_y, I'_{y-1} \leftarrow \mathcal{A}'(\lambda, m, B, a, n, x; Q_x, Q_{x-1}, \ldots, Q_{y+1}, Q'_y, Q_{y-1}, \ldots, Q'_1; R)$$

and if $y' = \epsilon$ or $y' \neq y$ the adversary fails.

5. By the second condition for success of $\mathcal{A}$, it now must hold that $I'_y = I_y$ and $\mu'_y = \mu_y$, as neither the random tape nor $Q_{1\ldots x}$ have changed.
6. If $Q_y \neq Q'_y$, the adversary succeeds and generates $(y, I_y, \mu_y, Q_y, Q'_y, I_{y-1}, I'_{y-1})$; otherwise, it fails.

By the generalized forking lemma, if $\mathcal{A}$ succeeds with probability ε, then the adversary succeeds with probability of at least $\varepsilon(\varepsilon/x - 1/2^\lambda)$. It follows that if $\mathcal{A}$ succeeds with non-negligible probability, then the adversary also succeeds with non-negligible probability, as $x \in O(\text{poly} \log m) \in O(\text{poly } \lambda)$. □

Combined, these two lemmas complete the desired proof of conditional soundness for our modular exponentiation proof scheme: if the low order assumption holds in $\mathbb{Z}_p^\times$, then for any initial configuration (λ, m, B, a, n, x), the probability of an adversary convincing the verifier for a state $I \notin \mathcal{L}$ is negligible in λ.

5 Final remarks

5.1 Implementation and time-space tradeoff

A direct usage of the formal definition of our proof scheme defined in Section 3.2 is impractical for the purposes of distributed primality testing, as it requires the calculation of $b^{2^{B2^{t-1}}}$ for $t = x \to 1$, incurring $2L$ squarings; exactly what we want to avoid. Using the abstraction described in Section 3.1, we see that it is equivalent to instead store $b^{2^{Bi}}$ for $i = 0 \ldots 2^x - 1$; however, for smaller B, the primality test becomes bottlenecked by disk I/O. In distributed computing, this entails balancing B not only between the additional prover cost and the verifier, but also the disk space the prover has available.

On the other hand, we can reverse the time-space tradeoff, by instead choosing a larger B for the prover. Yet this does not necessarily mean verification becomes more expensive; the final bottleneck for the verifier is always the verification of $r \equiv b^{2^B} a^c$, which takes polynomial (in L) squarings, as opposed to the interaction, which takes a polylogarithmic number of squarings. This is exactly the initial form, i.e. the prover needs that $(b, r, x', 1) \in \mathcal{L}'$ where $\mathcal{L}'$ is defined as in Section 3.2 with some λ, the same m, some different B' and x' such that $B'2^{x'} = B$, the same a, and a new $n = c$. In other words, the prover and verifier can recurse again on $(b, r, x', 1)$ and $\mathcal{L}'$, reducing the number of squarings for the verifier to B', at the cost of B extra squarings from the prover.

5.2 Implications of conditional soundness

Due to Shor's algorithm, the low-order assumption does not hold in the quantum computing model [1], as there exists a quantum adversary that computes the discrete logarithm in $\tilde{O}((\log m)^2)$. This does not yet pose a practical threat to distributed computing purposes: for probabilistic Fermat prime tests, not only is $\log m \gg 10^5$ several orders of magnitude larger than the intended scale of Shor's algorithm for decrypting public-key cryptosystems, but moreover because $n < m$ for the purpose of primality tests, classical algorithms also run in $\tilde{O}((\log m)^2)$ and are likely much faster. This *de facto* safety may change if our results are used to optimize other applications, such as cryptographic distributed exponentiation.

6 References

[1] P.W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science.* 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[2] Mihir Bellare and Gregory Neven. “Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma”. In: *Proceedings of the 13th ACM Conference on Computer and Communications Security.* CCS ’06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 390–399. ISBN: 1595935185. DOI: 10.1145/1180405.1180453. URL: https://doi.org/10.1145/1180405.1180453.

[3] Yves Gallot. “Genefer, A Program for Finding Large Probable Generalized Fermat Primes: Mathematical representation and algorithms”. In: (2017). URL: https://app.assembla.com/spaces/genefer/subversion/source/HEAD/trunk/doc/geneferMath.pdf.

[4] Dan Boneh, Benedikt Bünz, and Ben Fisch. *A Survey of Two Verifiable Delay Functions.* Cryptology ePrint Archive, Paper 2018/712. https://eprint.iacr.org/2018/712. 2018. URL: https://eprint.iacr.org/2018/712.

[5] Krzysztof Pietrzak. “Simple Verifiable Delay Functions”. In: *10th Innovations in Theoretical Computer Science Conference (ITCS 2019).* Ed. by Avrim Blum. Vol. 124. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 60:1–60:15. ISBN: 978-3-95977-095-8. DOI: 10.4230/LIPIcs.ITCS.2019.60. URL: http://drops.dagstuhl.de/opus/volltexte/2018/10153.

[6] Pavel Atnashev. *Efficient Proth/PRP Test Proof Scheme.* Mar. 2020. URL: https://www.mersenneforum.org/showthread.php?t=25323.

[7] Rytis Slatkevičius. *PrimeGrid.* URL: https://www.primegrid.com/.

7 Acknowledgements

The author would like to thank his advisor, Yves Gallot, not only for his invaluable advice for this paper, but also for his continued development of Genefer and its contribution to PrimeGrid, without either of which the author would never have realized the power of number theory in distributed computing.

The author would like to thank his principal, Wan Jie, for his support in our research.

This paper is dedicated to Ruvim Breydo, who taught me how to do real math.