FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Svetlana Ivanova

Practical Batch Proofs of Exponentiation

Computer Science Institute of Charles University

Supervisor of the bachelor thesis: Mgr. Pavel Hubáček, Ph.D.
Study programme: Computer Science

Prague 2024

I would not be able to finish this thesis or complete my studies at MatFyz without my supervisor, Professor Pavel Hubáček. He has helped me a lot, while I was struggling with my mental health, and he has always remained patient, kind and supportive. He has guided me not only through this research, but has also helped me when I needed it. I can not thank him enough for that. Another person that I'm grateful to is my fiancé Honza. He has been with me through the darkest times and put a smile on my face when I thought it was impossible.

Title: Practical Batch Proofs of Exponentiation

Author: Svetlana Ivanova

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Abstract: This thesis studies batch Proofs of Exponentiation (batch PoE). We explore existing batch PoEs and analyze their verification cost. We also introduce two batch PoEs and compare their performance with the performance of the existing approaches. Our batch PoEs outperform the existing ones, both in theory and in practice. We achieve the improvement in the verification costs by decreasing the expected number of group multiplications. For the practical analysis, we choose the values of the protocol parameters as used in practice, and then measure the time of multiplications and exponentiations on the verifier's side of the discussed protocols in our implementation in C++.

Keywords: Proof of Exponentiation, Batching, Verifiable Delay Function

Název práce: Praktické dávkové důkazy pro mocnění

Autor: Svetlana Ivanova

ústav: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Abstrakt: Tato práce se zabývá dávkovými důkazy pro mocnění (dávké PoE). Zkoumáme existující dávkové PoE a analyzujeme jejich ověřovací náklady. Také představujeme dva dávkové PoE a porovnáváme jejich výkon s výkonem existujících přístupů. Naše dávkové PoE překonávají existující, jak teoreticky, tak i prakticky. Zlepšení ověřovacích nákladů dosahujeme snížením očekávaného počtu grupových násobení. Pro praktickou analýzu vybíráme hodnoty parametrů protokolu používané v praxi a pak měříme čas násobení a mocnění na straně ověřovatele pro zkoumané protokoly v naší implementaci v jazyce C++.

Klíčová slova: Důkazy pro mocnění, Ověřitelné zpožďovací funkce, Dávkové důkazy

Contents

Introduction

Proofs of exponentiation

Recently, in the works by Pietrzak [1] and Wesolowski [2], the new constructions of *proofs of exponentiation* were presented. PoEs have a lot of practical applications. In particular, they can be used to construct *Verifiable Delay Functions*, shortly *VDFs*. VDFs, in the last several years, have become widely used for the blockchain networks. We talk more about it in Section 1.3.

In the setting of proof of exponentiation, there are two communicating parties - *verifier* and *prover*. Prover claims that, for two group elements $x, y \in \mathbb{G}$ and an exponent e, it holds that $x^e = y$. Prover wants to prove it to verifier. If what claims prover is true, verifier wants to accept such statement with probability 1 (*completeness*). Otherwise, when communicating with a dishonest prover, verifier accepts with some small probability (*soundness error*). The proof of exponentiation is then a protocol, i.e., an algorithm for verifier and prover that allows to verify the statement $x^e = y$ (*PoE-statement*) efficiently, completely and securely (i.e., with small soundness error).

Of course, verifier could just compute x to the power of e and check, whether $x^e = y$. But in most of the applications of PoEs, the exponents of bit-length 2^{25} are used, and the used groups $\mathbb{G}$ are usually large, too.

Batch proofs of exponentiation

Now imagine that we need to verify several PoE-statements. When there is only a couple of them, we can just execute PoE for every statement separately. But in most applications, there are thousands to millions PoE-statements that need to be verified at once, and the repeating of classical PoE becomes very slow. To resolve this problem, *batch proofs of exponentiation* are used. Batch proof of exponentiaion is, again, a protocol for verifier and prover, and this protocol should be also efficient, complete and secure. Batch proofs of exponentiation (also denoted as batching approaches or batching protocols) are the main subject of this thesis.

Goal of this thesis

The goal of this thesis is to dive into the existing constructions of proofs of exponentiation and batch proofs of exponentiation, such as Wesolowski's PoE [2] and the batching PoEs - the Random Subsets and Random Exponents Protocols presented by Rotem [3]. We aim to compare the verification costs of the Random Subsets Protocol, the Random Exponents Protocol and the repeated compilation of Wesolowski's PoE in theory. We also managed to come up with **two new batch PoE** constructions, inspired by Rotem's protocols [3] and the approaches by Bellare, Garay and Rabin [4]. We evaluate the verification costs for these two new protocols, both in theory and in practice, and proceed by comparing them with the existing Random Subsets and Random Exponents protocols. To do that, we implement these batch PoEs (both Rotem's and ours) in C++.

This thesis is based on our paper with Hoffman and Hubáček [5]. We do not present the proofs of security in this work as they were written by the coauthors. However, we have a more realistic implementation of the protocols in C++. The preliminaries in our work are more self-contained compared to [5] to be accessible also to non-experts.

1 Preliminaries

In this chapter, we describe several theoretical notions that are then used to present our results. We start with defining *proofs of exponentiation* and proceed with the core definition of this thesis - *batch proofs of exponentiation.* We also mention already existing batch proofs of exponentiation and define the comparison metrics, i.e., the metrics parameters that we use to compare the existing and our batching approaches.
The definitions and figures in this thesis are taken from our article [5].

Notation. $[m] := \{1, 2, ..., m\}$ is the set of all positive integers smaller than or equal to m.

1.1 Proofs of exponentiation

1.1.1 Definition

In the setting of a proof of exponentiation *(PoE)* we have two communicating parties - a verifier and a prover. The prover wants to show the verifier that, for the given pair of group elements $(x, y) \in \mathbb{G}$ (where $\mathbb{G}$ is a group of a **hidden** order) and a fixed exponent $e \in \mathbb{N}$, it holds that $x^e = y$. This process *(an interactive proof)* is then called the proof of exponentiation. Of course, verifier could just compute x^e and check whether the computed number corresponds to y. But this solution is extremely slow, as in the majority of PoE applications the **exponents** of bit-length around 2^{25} are used. It means that, even by using the repeated squaring to efficiently calculate x^e in $\log(e)$ steps, one should perform 2^{25} group operations on the group elements of bit-length roughly equal to 2048. This is the reason for using some randomized techniques. Such techniques were presented in the recent researches by Pietrzak [1] and Wesolowski [2].

Let us informally describe an *interactive proof.* For two communicating parties $\mathcal{V}$ and $\mathcal{P}$ (a *verifier* and a *prover*), a language $\mathcal{L}$ and some statement s, $\mathcal{V}$ accepts with probability 1, if the statement s is from $\mathcal{L}$ (*completeness*). Otherwise, if the statement s is not from $\mathcal{L}$, $\mathcal{V}$ accepts with some small probability, which is then called *soundness error.*

Definition 1. ***Interactive proof***. *For a function $\varepsilon : \mathbb{N} \to [0, 1]$, an* interactive proof for a language *$\mathcal{L}$ is a pair of interacting algorithms $(\mathcal{P}, \mathcal{V})$, where $\mathcal{V}$ is a probabilistic polynomial time algorithm, which satisfies the following properties:*

- ***Completeness:*** *For every $x \in \mathcal{L}$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on the common input x, then $\mathcal{V}$ accepts with probability* 1*.*
- ***Soundness:*** *For every $x \notin \mathcal{L}$ and every (computationally unbounded) cheating prover strategy $\widetilde{\mathcal{P}}$, the verifier $\mathcal{V}$ accepts when interacting with $\widetilde{\mathcal{P}}$ with probability less than $\varepsilon(|x|)$, where ε is called the* soundness error.

As mentioned before, in the setting of proof of exponentiation, $\mathcal{P}$ wants to convince $\mathcal{V}$ that $x^e = y$ for group elements $x, y \in \mathbb{G}$ and a given exponent $e \in \mathbb{N}$. Proof of exponentiation is then an interactive proof for the language of such statements.

Definition 2. ***Proof of exponentiation (PoE)*** *is an interactive proof for the language*

$$\mathcal{L} = \{(x, y, e) \in \mathbb{G}^2 \times \mathbb{N}, x^e = y\}$$

Note that x^e can be calculated in $\log(e)$ sequential steps by repeated squaring. It means that an efficient PoE should be much faster. In our work, we will use Wesolowski's construction of PoE [2]. As mentioned earlier, it uses randomized technique, and therefore provides fast verification.

1.1.2 Wesolowski's proof of exponentiation

Let us discuss an example of a proof of exponentiation - Wesolowski's PoE. For a statement $x^{2^t} \stackrel{?}{=} y$ the task is to design an efficient complete and sound protocol, which allows the verification of this statement.
In the setting of Wesolowski's PoE, there are 3 steps:

1. A verifier $\mathcal{V}$ samples a prime l from some set of primes uniformly at random.
2. The prover $\mathcal{P}$ then computes $\pi = x^{\lfloor 2^t/l \rfloor}$ and sends it to $\mathcal{V}$.
3. $\mathcal{V}$ calculates $r = 2^t \bmod l$ and accepts, if $\pi^l x^r = y$.

The set of primes from the first step is $Primes(2k)$ - the set containing the first 2^{2k} primes, where k is a parameter for this protocol. Now, for the described protocol to be PoE, it should be complete and sound. The completeness is straightforward; if $\mathcal{P}$ is honest, then

$$\pi^l = x^{\lfloor 2^t/l \rfloor \cdot l} = x^{\frac{2^t - r}{l} \cdot l} = x^{2^t - r},$$

$$\pi^l x^r = x^{2^t - r} \cdot x^r = x^{2^t} = y.$$

The soundness of the protocol depends on the parameter k (it means that k is *the security parameter*) and is described in the paper by Wesolowski [2].

In Figure 1.1 we show how a prover $\mathcal{P}$ and a verfieier $\mathcal{V}$ act in Wesolowski's PoE.

We base our implementation of different batching approaches that are described further on an implementation of Wesolowski's PoE by Attias, Vigneri and Dimitrov of the Network team of the IOTA Foundation [6].

1.2 Batch proofs of exponentiation

1.2.1 Definition

In our work, we focus on the batch proofs of exponentiation (alternative names are batching approaches, batching protocols) - the effective interactive proofs for verifying several PoE instances at once. Frequently, in the higher-level protocols, PoEs instances are generated repeatedly. The natural solution for reducing communication and time complexities is to securely combine, i.e., *batch* these instances (usually by multiplication) into much smaller amount. Batching protocols can be viewed as the technique that is used on top of the existing PoEs, where PoE serves as the black box. In our case, this "black box" is Wesolowski's PoE.

Parameters: $\mathbb{G}, t, k$

1. t is the logarithm of the exponent from PoE statement $e = 2^t$
2. k is the security parameter
3. $Primes(2k)$ is a set containing the first 2^{2k} primes
4. group $\mathbb{G}$
5. common exponent e

Statement: $y = x^e$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples a prime l uniformly at random from $Primes(2k)$ and sends it to $\mathcal{P}$.
2. $\mathcal{P}$ computes $\pi = x^{\lfloor 2^t/l \rfloor}$ and sends it to $\mathcal{V}$
3. $\mathcal{V}$ computes $r = 2^t \bmod l$, and accepts if $x, y, \pi \in \mathbb{G}$ and $\pi^l x^r = y$

Figure 1.1 Wesolowski's PoE [2].

Definition 3. ***Batch proof of exponentiation (Batch PoE)****. A* batch proof of exponentiation *for m statements in a group $\mathbb{G}$ is an interactive proof for the language*

$$\mathcal{L} = \{\{(x_i, y_i, e)\}_{i \in [m]} \in \{\mathbb{G}^2 \times \mathbb{N}\}^m \mid x_i^e = y_i \textit{ for all } i \in [m]\}.$$

1.2.2 Existing approaches

In his paper, Rotem [3] suggested two general batching techniques: *Random Subsets Protocol* and *Random Exponents Protocol.* In fact, Rotem's protocols are based on the approaches of Bellare, Garay and Rabin [4] that are also the main foundation for our results.

Random Subsets Protocol

In the setting of the Random Subsets Protocol, we have a group $\mathbb{G}$ and m PoE-statements (the statements of form $x_i^e = y_i$ for $i \in [m]$, $x_i, y_i \in \mathbb{G}$). We also have a parameter ρ, which is the number of repetitions of the following steps:

1. $\mathcal{V}$ samples the random subset $\mathcal{S}$ of a set $[m]$ and sends $\mathcal{S}$ to $\mathcal{P}$,
2. both $\mathcal{V}$ and $\mathcal{P}$, based on the generated set $\mathcal{S}$, choose the subset of the input statements; if $i \in \mathcal{S}$, then $\mathcal{P}$ and $\mathcal{V}$ take the i-th statement, otherwise they don't,
3. $\mathcal{V}$ and $\mathcal{P}$ then separately calculate the product of the chosen statements, it results in one batch statement,

Parameters: $\mathbb{G}, e, m, \rho$
Statements: $\{y_i = x_i^e\}_{i\in[m]}$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples a matrix $B \leftarrow \{0,1\}^{\rho\times m}$ uniformly at random and sends it to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ construct new statements $\{y'_i = (x'_i)^e\}_{i\in[\rho]}$, where
$$y'_i = \prod_{j\in[m]} y_j^{B_{i,j}} \quad \text{and} \quad x'_i = \prod_{j\in[m]} x_j^{B_{i,j}}.$$
3. $\mathcal{V}$ and $\mathcal{P}$ run ρ many `PoE` on $\{y'_i = (x'_i)^e\}_{i\in[\rho]}$ in parallel.

Figure 1.2 Random Subsets Protocol [3].

4. any PoE protocol is then run on this final statement.

In the Random Subsets protocol, these steps are executed **in parallel**. The precise description of the Random Subsets protocol is presented in Figure 1.2.

Random Exponents Protocol

Another batch PoE, presented by Rotem, is the Random Exponents Protocol. For every of the m given PoE statements, $\mathcal{V}$ generates a random exponent and sends these m exponents to $\mathcal{P}$. Both $\mathcal{V}$ and $\mathcal{P}$ separately raise the PoE statements to these random powers and calculate the product of such statements. Then any PoE protocol is executed to verify this one combined statement. The exact description of the Protocol is presented in Figure 1.3.

For this protocol to be sound, the low order assumption must hold:

Definition 4. *Low order assumption. Let* `GGen`*(1^λ) be a randomized algorithm that outputs the description of a group $\mathbb{G}$ of unknown order. We say that the* low order assumption *holds for* `GGen` *if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability of winning the following game is negligible in λ:*

1. *$\mathcal{A}$ takes as input the description of a group $\mathbb{G}$ output by* `GGen`*(1^λ).*
2. *$\mathcal{A}$ outputs a pair $(d, \alpha) \in [2^\lambda] \times \mathbb{G}$.*
3. *$\mathcal{A}$ wins if and only if $\alpha \neq 1$ and $\alpha^d = 1$.*

The low order assumption means that it is computationally hard to find the elements of low (i.e., $\leq 2^\lambda$) order in some groups of hidden order. There is still not much known about the usability of this assumption in practice. In [7] the assumption is analyzed for RSA groups. In class groups, it is broken for Mersenne primes and other special types of primes [8].

Parameters: $\mathbb{G}, e, m, \ell$
Statements: $\{y_i = x_i^e\}_{i\in[m]}$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples a vector $r \leftarrow [2^\ell]^m$ uniformly at random and sends it to $\mathcal{P}$.

2. $\mathcal{V}$ and $\mathcal{P}$ both construct one new statement $\tilde{y} = (\tilde{x})^e$, where

$$\tilde{y} = \prod_{i\in[m]} (y_i)^{r_i} \quad \text{and} \quad \tilde{x} = \prod_{i\in[m]} (x_i)^{r_i}.$$

3. $\mathcal{V}$ and $\mathcal{P}$ run `PoE` on statement $\tilde{y} = (\tilde{x})^e$.

Figure 1.3 Random Exponents Protocol [3].

1.2.3 Comparison metrics

Metrics

The main metrics that we use to evaluate batch PoEs are *expected number of group multiplications in verification*, *number of produced proofs* and *soundness error* (enumerated in the order of practical importance). Another important factor is the communication complexity, i.e., how much data is sent in communication between a verifier and a prover, but we do not focus on it in this thesis.

The expected number of group multiplications in verification. The effectiveness of PoE (or batch PoE) can be measured by the verification cost, because it is the main parameter when it comes to the practical use cases of PoEs. All PoEs and batching protocols that we mention in this thesis use just three types of operations: choosing a random element, exponentiation of the group element and multiplication of two group elements. We focus on the *expected* number of group multiplications on the verifier's side of the protocol, because:

1. choosing a random element is usually a light-weighted operation compared to group multiplication, and,

2. exponentiation of the group element can be expressed in the number of group multiplications.

When raising the instances to uniformly random l-bit exponents, one has to perform $1.5l$ multiplications in expectation per each exponentiation: for every bit of the exponent, the squaring should be performed and, additionally, if this bit equals 1, then there is one more multiplication needed. Since the exponents are random, every bit of the exponent is equal to 1 with probability 0.5, therefore the expected number of multiplications per every bit of the exponent is $1+1\cdot 0.5 = 1.5$, which results in $1.5l$ number of multiplications in expectation when raising to the exponent of bit-length l. Since all PoEs and batch PoEs that are described in this thesis, use some kind of randomization, it makes sense to consider the *expected* number of multiplications.

Number of produced proofs. Another important metric for evaluation of the batching protocol is the number of produced PoE proofs. Proofs are produced by the prover, and the prover is usually the one who does the most computations in any PoE, so producing proofs is both computationally expensive and space inefficient, as the proofs are large.

Soundness. Soundness is included in the definition of PoE, and, therefore, a batch PoE. It shows, how hard it is to trick a verifier into accepting the wrong PoE statement(s). Soundness error is the probability of verifier accepting the wrong statement. This probability should be small for the batch PoE (or just PoE) to be secure and usable. But soundness error is not usually a constant number; it is a function of one (or, sometimes, several) parameters - these are then called the *security parameters*. In this thesis, when the symbol λ is used, it represents some security parameter. When comparing two batch PoEs, it is important to compare them with the same security level (i.e., with the same order of soundness error), because the number of multiplications (and therefore the verification cost) can be dependent on the security parameter. This is the case for all PoEs and batch PoEs, described in this thesis. For example, the soundness of the Random Subsets Protocol depends on the number of chosen subsets, and, as a consequence, on the number of multiplications on the verifier's side. If the soundness of one batch PoE is $2^{-\lambda}$ and of another batch PoE it is just λ^{-1}, then the setting of $\lambda = \lambda_0$ for the second protocol implies that the security parameter for the first protocol should be $\lambda = \log(\lambda_0)$. We explore the soundness of the Random Subsets Protocol and the Random Exponents protocol further in this chapter.

The evaluation of Wesolowski's PoE

Based on the defined metrics, we now evaluate Wesolowski's PoE.

Soundness. Wesolowski's PoE is sound [2] and its soundness depends on the security parameter k. We choose k to be 128 as suggested in the paper [2].

The expected number of group multiplications in verification. In Wesolowski's PoE setting, $\mathcal{V}$ verifies a statement $\pi^l x^r \overset{?}{=} y$, where $\pi = x^{\lfloor \frac{2^t}{l} \rfloor}$ and $r = 2^t \bmod l$. l is chosen uniformly at random from the set of the first 2^{2k} primes denoted as $Primes(2k)$.
The last of the first 2^{2k} primes has more than $2k$ bits. The number of bits can be estimated by solving the equation $\frac{n}{ln(n)} = 2^{2k}$ (prime number theorem), and the bit length of the 2^{2k}-th prime number is then approximately $\lceil \log_2(n) \rceil$. For $k = 128$ it is approximately 183, but since we choose the prime number randomly from the set $Primes(2k)$, it could have less bits. For simplicity we say that the expected bit length of the chosen prime number l is k, i.e., 128.
Then r also has k bits in expectation. We have already showed that the expected number of multiplications when raising a group element to the power of bit length len is $1.5 \cdot len$. It implies that the number of group multiplications to compute $\pi^l x^r$ is:

$$1.5 \cdot \log_2(l) + 1.5 \cdot \log_2(r) + 1 = 1.5k + 1.5k + 1 = 3k + 1,$$

where $+1$ is for multiplying π^l and x^r.

1.2.4 Evaluation of the existing approaches

We choose the soundness error of the batching protocols (excluding the soundness error of the PoE protocol, in our case Wesolowski's PoE) to be roughly 2^{-128}, that is 128 bits.

Random Subsets Protocol - analysis and practical parameters

Let us recall the Random Subsets Protocol. We start with m statements. In the first step of the protocol, $\mathcal{V}$ chooses ρ subsets of the set $[m]$ and sends it to $\mathcal{P}$. For each of the ρ subsets, both $\mathcal{P}$ and $\mathcal{V}$ compute the new combined statement - they calculate the product of the original statements, which indices are in the current subset. By doing this, $\mathcal{V}$ and $\mathcal{P}$ obtain ρ combined statements and then they run ρ PoEs in parallel. $\mathcal{V}$ accepts, if all PoEs succeed.
The detailed description is in Figure 1.2. Based on the defined metrics, we now evaluate the Random Subsets Protocol.

The expected number of group multiplications in verification. The only step where the group multiplication is performed is the calculation of the combined statement for x and y for each of the ρ subsets. The expected number of group multiplications on the verifier's side is $2 \cdot \rho$ (because there are two combined statements for every subset - one for x and one for y) times the expected number of elements in the randomly chosen subset of $[m]$. For every element of the $[m]$, it is chosen with the probability $\frac{1}{2}$, therefore the expected number of elements in the random subset of $[m]$ is $\frac{1}{2} \cdot m$.

Finally, the expected number of group multiplication for the verifier in the Random Subsets Protocol is $2 \cdot \rho \cdot \frac{1}{2} \cdot m = \rho \cdot m$.

Number of produced proofs. Due to the parallel run of ρ PoEs in the last step of the Random Subsets Protocol, the protocol produces ρ proofs.

Soundness. In [3], the soundness of the Random Subsets Protocol is proved to be

$$\delta(\lambda) = \delta_{PoE}(\lambda) + 2^{-\rho},$$

where $\delta_{PoE}(\lambda)$ is the soundness of the used PoE. For the Random Subsets protocol to reach the desired security level of 128 bits (i.e., the soundness error of 2^{-128}), we set $\rho = \lambda = 128$.

Summary. We denote $\text{mult}_{protocol}$ as the expected number of group multiplications in verification for the *protocol*, λ is the corresponding security parameter, ρ is the number of repetitions, $\delta_{protocol}$ is the soundness error and $\text{proofs}_{protocol}$ is the number of produced PoE-proofs. Then, for the Random Subsets Protocol (RS) we get:

$$\begin{gathered}
\rho = \lambda = 128, \\
\delta_{RS}(\lambda) = \delta_{PoE}(\lambda) + 2^{-\rho} = \delta_{PoE}(\lambda) + 2^{-\lambda}, \\
\text{mult}_{RS}(\lambda) = \rho m = \lambda m, \\
\text{proofs}_{RS}(\lambda) = \rho = \lambda.
\end{gathered}$$

Random Exponents Protocol - analysis and practical parameters

In the Random Exponents Protocol, $\mathcal{V}$ generates m random exponents $r_1, ..., r_m \in [2^l]$ and sends them to $\mathcal{P}$. In the second step, both $\mathcal{V}$ and $\mathcal{P}$ compute the new instances by exponentiation of the original statements to the corresponding exponent:

$$x_i^e \stackrel{?}{=} y \to (x_i^e)^{r_i} \stackrel{?}{=} y^{r_i}.$$

After that, $\mathcal{V}$ and $\mathcal{P}$ calculate the product of the new instances. In the last step, PoE is run to check this generated statement. The detailed description is in Figure 1.3.

Expected number of group multiplications in verification. In the Random Exponents Protocol, the group multiplications are performed in the exponentiation of the input statements to the powers $r_1, r_2, ..., r_m \in [2^l]$ and during the calculation of the product of the powered instances. We have already explained that the expected number of multiplications when raising a group element to the l-bit exponent is $1.5l$. Then the exponentiation step in the Random Exponents Protocol takes $1.5l \cdot 2 \cdot m$ group multiplications (because we exponentiate both x and y). The product calculation takes $m \cdot 2$ multiplications. Therefore, the expected number of the group multiplications in verification is:

$$1.5l \cdot 2 \cdot m + m \cdot 2 = (3l + 2) \cdot m.$$

Number of produced proofs. In the Random Exponents Protocol, PoE is executed only once during the final step, so only 1 PoE-proof is constructed.

Soundness. Let $\delta_{PoE}(\lambda)$ denote the soundness error of the used PoE, where λ is the security parameter. Assuming that the low order assumption for the group $\mathbb{G}$ generated by $\texttt{GGen}(\lambda)$ holds with soundness error μ, the soundness error of the Random Exponents Protocol can be expressed as follows:

$$\delta_{RE}(\lambda) \leq \delta_{PoE}(\lambda) + \mu + \tfrac{1}{2^l}.$$

In his paper, Rotem [3] shows a slightly different soundness error estimation, but it was reevaluated in our paper in Lemma 2 [5].
The Random Exponents Protocol uses the low order assumption, i.e., that it is computationally hard to find the group elements of order $\leq 2^\lambda$, where λ is the same security parameter used in the soundness error estimation. The desired security level is 128 bits, therefore we set l to be 128.

Summary. The parameters for the Random Exponents Protocol are the security parameter λ and the bit length of the generated random exponents l. As discussed, we set l to be 128. Using the notation defined earlier we get:

$$\begin{gathered} l = \lambda, \\ \delta_{RE}(\lambda) \leq \delta_{PoE}(\lambda) + 2^{-l} = \delta_{PoE}(\lambda) + 2^{-\lambda}, \\ \text{mult}_{RE}(\lambda) = (3l + 2) \cdot m = (3\lambda + 2) \cdot m, \\ \text{proofs}_{RE} = 1. \end{gathered}$$

1.3 Applications

PoEs are closely connected to the *verfiable delay functions (VDFs)*. In fact, PoEs can be used to construct VDFs. The idea behind VDF is as follows: for some fixed number t, we want to construct an efficiently verifiable and secure function f, such that evaluating $f(x)$ for every x takes t sequential steps. While PoE is interactive, VDF is not, and it complicates the construction of VDFs from PoEs. This issue can be resolved by using Fiat-Shamir heuristic [9].

There are several applications of VDF, such as randomness beacons, proof of replication, resource efficient blockchains and some more [10]. For example, in Chia Network blockchain [11], 32 VDF proofs are generated in expectation every 10 minutes. It means that every 10 minutes, some user (or group of users) is chosen to calculate 32 PoE-statements. Then, every user in the blockchain network should verify these calculated statements. The problem appears when on-boarding new users: they should verify all historical blocks, and there are many of them - now, in Chia Network there is already around 4.8 million PoE-statements. In such blockchain setup, batch PoE can be used for the efficient on-boarding - miners could compute a single batch PoE for all intermediate PoE statements, which would significantly reduce the storage overhead and user's initial computation. And while it is not relevant for Chia blockchain, as the groups in which PoE-statements are evaluated constantly change, batching can be applied in another similar setup with fixed group.

2 Our Protocols

In this chapter, we present our batch PoEs - the Hybrid protocol and the Bucket protocol. We also state the theorems describing the soundness of the suggested approaches. We do not provide the proof of the soundness theorems as a part of this thesis, however, it can be found in our paper [5].

2.1 Hybrid protocol

2.1.1 Description

The hybrid protocol uses both random subsets and random exponents techniques. In the first step, $\mathcal{V}$ generates ρ random subsets and ρ random exponents. $\mathcal{V}$ sends this information to $\mathcal{P}$. Then, in the second step, ρ batching instances are calculated by the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ the same way as in the Random Subsets Protocol. The difference is in the third step: instead of the parallel running of ρ PoEs, we combine these ρ instances together by using the Random Exponents approach, i.e., we raise every one of these ρ instances to the corresponding random exponent, generated by $\mathcal{V}$ during the first step, and then multiply all ρ statements together to form one batched instance. This instance is then verified by using any PoE protocol. The exact description of the protocol is presented in Figure 2.1.

2.1.2 Metrics evaluation

Expected number of group multiplications in verification. Group multiplications in the Hybrid Protocol are performed during the “random subsets” phase (where for every one of the chosen ρ random subsets we calculate the product of the statements, whose indices are in the given subset) and during the “random exponents” phase (where every statement, obtained by the product calculation in the “random subsets” phase, is raised to the corresponding random exponent).
The expected number of multiplications in verification in the “random subsets” phase is as follows:

$$\rho \cdot \tfrac{m}{2} \cdot 2 = \rho m,$$

where ρ is for the number of random subsets, $\frac{m}{2}$ stands for the expected number of elements in each such subset and 2 is for calculating the products for x and y. The expected number of multiplications in verification in the “random exponents” phase is the same formula as in the Random Exponents Protocol and can be expressed as:

$$(3l + 2)\rho,$$

where l is the bit-length of the generated random exponents and ρ is the number of instances to batch.

The complete number of group multiplications during verification in the Hybrid Protocol is $\rho m + (3l + 2)\rho = (3l + 2 + m)\rho$.

Parameters:

- group $\mathbb{G}$, common exponent e, number of statements m
- number of repetitions of subset multiplications ρ
- size of random coins ℓ

Statements: $\{y_i = x_i^e\}_{i\in[m]}$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples a matrix $B \leftarrow \{0,1\}^{\rho\times m}$ and a vector $r \leftarrow [2^\ell]^\rho$ uniformly at random and sends both to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ both construct new statements $\{y_i' = (x_i')^e\}_{i\in[\rho]}$, where
$$y_i' = \prod_{j\in[m]} y_j^{B_{i,j}} \quad \text{and} \quad x_i' = \prod_{j\in[m]} x_j^{B_{i,j}}.$$
3. $\mathcal{V}$ and $\mathcal{P}$ both construct one new statement $\tilde{y} = (\tilde{x})^e$, where
$$\tilde{y} = \prod_{i\in[\rho]} (y_i')^{r_i} \quad \text{and} \quad \tilde{x} = \prod_{i\in[\rho]} (x_i')^{r_i}.$$
4. $\mathcal{V}$ and $\mathcal{P}$ run `PoE` on statement $\tilde{y} = (\tilde{x})^e$.

Figure 2.1 Our hybrid batch proof of exponentiation.

Number of produced proofs. The PoE is run only in the final step of the Hybrid Protocol, therefore it produces 1 PoE proof.

Soundness. The Hybrid protocol uses the Random Exponents protocol that is based on the low order assumption. Therefore, the soundness of the Hybrid protocol is also based on the low order assumption. The exact soundness of the Hybrid Protocol is proved in our paper [5] and is expressed by the Theorem 1.

Theorem 1. *Let `PoE` be a proof of exponentiation with soundness error $\delta_{PoE}(\lambda)$ and let $\mathbb{G}$ be a group output by `GGen`(λ). Assuming the low order assumption for `GGen` with soundness error μ, the hybrid batching protocol presented in Figure 2.1 has soundness error at most $\delta_{PoE}(\lambda) + \mu + 2^{-\rho} + 2^{-\ell}$*

To reach the desired security level of 128 bits and using the fact that the security parameter in our setting of Wesolowski's PoE is $\lambda = 128$, we set

$$\rho = \ell = \lambda = 128.$$

The soundness of the Hybrid Protocol is then as follows:

$$\delta_{HP}(\lambda) \leq \delta_{PoE}(\lambda) + \mu + 2 \cdot 2^{-\lambda}.$$

Summary. The Hybrid Protocol has 3 parameters: the security parameter λ, which in our setting is 128, the number of generated subsets (also is set to $\lambda = 128$) and the bit length of the random exponents (again, equals to $\lambda = 128$). Therefore, the metrics evaluation can be expressed as:

$$\text{mult}_{HP} = (3l + m + 2)\rho = (3\lambda + m + 2)\lambda,$$

$$\text{proofs}_{HP} = 1,$$

$$\delta_{HP}(\lambda) \leq \delta_{PoE}(\lambda) + \mu + 2 \cdot 2^{-\lambda},$$

where mult_{HP} denotes the expected number of multiplications in verification, proofs_{HP} means the number of produced PoE-proofs, $\delta_{HP}(\lambda)$ stands for the soundness error of the Hybrid Protocol, μ is the soundness error of low order assumption in the group $\mathbb{G}$, $\delta_{PoE}(\lambda)$ is the soundness error of the used PoE.

2.2 Bucket protocol

2.2.1 Description

The Bucket protocol is an adaptation of the bucket test of Bellare, Garay, and Rabin [4] to the setting of batch proofs of exponentiation. The main idea of the Bucket protocol is to repeat the following steps ρ times:

1. divide the input instances into $K = 2^k$ buckets,
2. in every bucket, calculate the product of the instances, which results in the total of K instances,
3. obtain 1 instance from these K instances by using the Random Exponents technique, i.e., raise every instance to some random power and then calculate the product of these instances, without producing the PoE-proof.

With this steps, we get ρ instances, for which we then run the Random Exponent protocol (meaning with PoE proof). The detailed description of the bucket protocol is in Figure 2.2.

2.2.2 Metrics evaluation

Expected number of group multiplications in verification. In the bucket protocol, we repeat the described 3 steps ρ times. In the first step, we divide m instances between $K = 2^k$ buckets using the randomization. The expected number of instances in the bucket is then $m \cdot \frac{1}{2^k}$, as the probability of an instance ending up in the particular bucket is $\frac{1}{2^k}$.

During the second step, we calculate the product of instances in each bucket. It results in $2^k \cdot \frac{m}{2^k} \cdot 2 = 2m$ group multiplications (number of buckets times the expected number of instances in the bucket times 2 - for x and y).

During the third step, we use the Random Exponents technique with the exponents of size k (from $K = 2^k$) on the 2^k instances. In the Random Exponents technique, the expected number of group multiplications in verification is $(3l+2)m$, where m is the number of instances to batch and l is the bit-length of random exponents. In our case, for using the Random Exponents technique, we "pay" $(3k + 2) \cdot 2^k$ group multiplications in expectation.

Overall, the repetition of these 3 steps ρ times results in the expected number of group multiplications in verification being:

Parameters:

- group $\mathbb{G}$, common exponent e, number of statements m
- number of buckets $K = 2^k$, where $k \leq \lambda$
- number of repetitions of bucketings $\rho = \lceil \lambda/(k-2) \rceil$
- size of random coins ℓ

Statements: $\{y_i = x_i^e\}_{i\in[m]}$ in $\mathbb{G}$
Protocol:

1. $\mathcal{V}$ samples two matrices $B \leftarrow [K]^{\rho\times m}$ and $R \leftarrow [K]^{\rho\times K}$ and a vector $r \leftarrow [2^\ell]^\rho$ uniformly at random and sends both to $\mathcal{P}$.
2. $\mathcal{V}$ and $\mathcal{P}$ both construct new statements $\left\{y'_{i,b} = (x'_{i,b})^e\right\}_{i\in[\rho],b\in[K]}$, where
$$y'_{i,b} = \prod_{j\in[m],B_{i,j}=b} y_j \quad \text{and} \quad x'_{i,b} = \prod_{j\in[m],B_{i,j}=b} x_j.$$
3. $\mathcal{V}$ and $\mathcal{P}$ both construct new statements $\{y''_i = (x''_i)^e\}_{i\in[\rho]}$, where
$$y''_i = \prod_{b\in[K]} (y'_{i,b})^{R_{i,b}} \quad \text{and} \quad x''_i = \prod_{b\in[K]} (x'_{i,b})^{R_{i,b}}.$$
4. $\mathcal{V}$ and $\mathcal{P}$ both construct one new statement $\tilde{y} = (\tilde{x})^e$, where
$$\tilde{y} = \prod_{i\in[\rho]} (y''_i)^{r_i} \quad \text{and} \quad \tilde{x} = \prod_{i\in[\rho]} (x''_i)^{r_i}.$$
5. $\mathcal{V}$ and $\mathcal{P}$ run `PoE` on statement $\tilde{y} = (\tilde{x})^e$.

Figure 2.2 Our Bucket batch proof of exponentiation based on the bucket test from Bellare et al. [4].

$$\rho \cdot (2m + (3k+2)2^k).$$

Finally, on the obtained ρ statements we again use the Random Exponents protocol (it means, we not only calculate the product of the instances raised to the random exponents, but we also call the used PoE for verification, and it results in producing PoE-proof) with the bit-length of exponents equal to l and the expected multiplication costs on the verifier's side is then:

$$(3l + 2)\rho.$$

Overall, the expected number of group multiplication in verification for the Bucket Protocol is:

$$\rho \cdot (2m + (3k+2)2^k + (3l+2)) = \lceil \tfrac{\lambda}{k-2} \rceil (2m + (3k+2)2^k + (3\lambda + 2)),$$

because we use $l = \lambda = 128$ and we set ρ to be $\lceil \frac{\lambda}{k-2} \rceil$ [5].

Soundness. The soundness of the Bucket Protocol is proved in our paper [5] and is stated as follows:

Theorem 2. *Let $\texttt{PoE}$ be a proof of exponentiation with soundness error $\delta_{PoE}(\lambda)$ and let $\mathbb{G}$ be a group output by $\texttt{GGen}(\lambda)$. Assuming the low order assumption for $\texttt{GGen}$ with soundness error $\mu \leq 2^{-k} - 2^{-\lambda}$, the bucket batching protocol presented in Figure 2.2 has soundness error at most $\delta_{PoE}(\lambda) + \mu + 2^{-\lambda+1} + 2^{-\ell}$*

Using $l = \lambda$, we get:

$$\delta_{BP}(\lambda) \leq \delta_{PoE}(\lambda) + \mu + 2^{-\lambda+1} + 2^{-\lambda}.$$

Number of produced proofs. PoE-proof is produced only in the final phase of the protocol, i.e., while using the Random Exponents Protocol for batching of ρ instances, therefore:

$$\text{proofs}_{BP} = 1.$$

Summary. With $l = \lambda = 128$ and $\rho = \lceil \frac{\lambda}{k-2} \rceil$, the Bucket Protocol can be evaluated using the defined metrics as follows:

$$\text{mult}_{BP} = \lceil \tfrac{\lambda}{k-2} \rceil (2m + (3k+2)2^k + (3\lambda + 2)),$$

$$\text{proofs}_{BP} = 1,$$

$$\delta_{BP}(\lambda) \leq \delta_{PoE}(\lambda) + \mu + 2^{-\lambda+1} + 2^{-\lambda}.$$

Note that k can be chosen optimally with respect to the number of statements m, e. g., by iterating through various options for k and minimizing mult_{BP}.

3 Protocols Comparison

In this chapter, we first look into the theoretical comparison of the existing batching approaches, described in Chapter 1 (Random Subsets Protocol, Random Exponents Protocol) and our approaches, described in Chapter 2 (Hybrid Protocol, Bucket Protocol), based on the defined comparison metrics and suggested values of the protocol parameters. After that, we present our experiments implementation and show the comparison results in practice. We then discuss the results and possible influences.

3.1 Theoretical comparison

Based on the calculations, presented in “Summary” sections in Chapter 1 and Chapter 2, we fill in Table 3.1 (presented in our paper [5]), where “No Batching” states for the repeated run of the used PoE protocol, resulting in the number of PoE-proofs equal to number of instances m.

In the “No Batching” setting with using Wesolowski’s PoE, the number of multiplications is the number of instances m times the verification cost of Wesolowski’s PoE, i.e., $(3l + 1)$. Therefore, the total number of multiplications in verification for the repeated Wesolowski’s PoE is $(3l + 1)m$, which is similar to the Random Exponents Protocol when setting $l = \lambda$ (and even seems to be faster). However, when using a non-interactive variant of Wesolowski’s PoE, it is necessary to set $l = 2\lambda$ due to an attack by Boneh, Bünz, and Fish [12]. Therefore, the Random Exponents Protocol will likely speed up the batch verification by a factor of two in practice[1]. Moreover, as described above, the repeated Wesolowski’s PoE results in m PoE-proofs, which is very space consuming when performing batching, for example, on 10^6 instances.

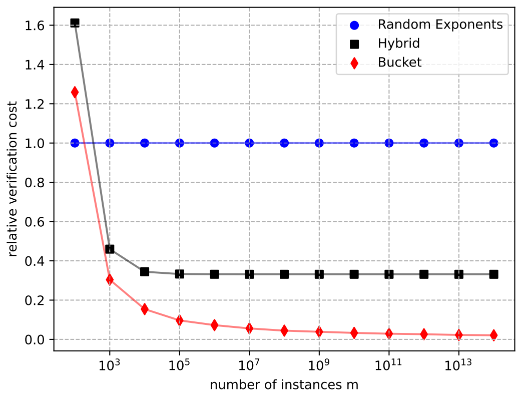
From Table 3.1, we can see that the Random Subsets Protocol performs less group multiplications compared to the Random Exponents Protocol, however it results in λ PoE-proofs (compared to the **single** proof for the Random Exponents Protocol) and requires parallelism.

The Hybrid Protocol offers roughly the same number of expected group multiplications as the Random Subsets Protocol up to an additive overhead independent on the number of instances m, while producing only 1 PoE-proof.

The Bucket Protocol multiplication cost depends on the number of buckets and also consists of the additive overhead and the part dependent on the number of instances m.

This comparison can be better understood by the theoretical evaluation of the multiplication costs depending on the number of instances and a parameter k for the Bucket protocol. It can be done by calculating the values of described functions $\text{mult}_{RS}, \text{mult}_{RE}, \text{mult}_{HP}, \text{mult}_{BP}$ for different number of input statements and optimal values for k. We present the results of such comparison (source code [14]) in Figure 3.2 and Figure 3.1. We can see that already for $m = 10^3$ both

[1]There also might be additional gains because, when verifying the Wesolowski’s proof, the verifier needs to additionally perform some computation, which induces potential non-trivial overhead. The additional overhead might be at least half of the verification time, as reported by Attias et al. [13] (see Table 1 in [13])



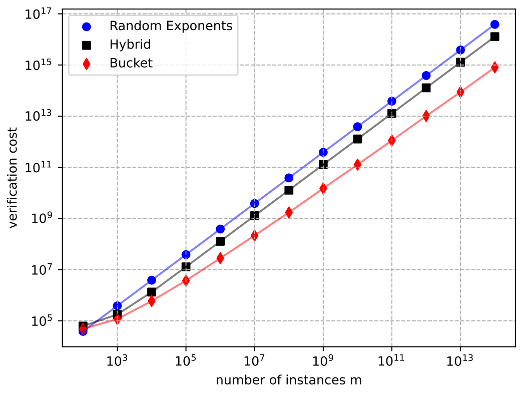


Figure 3.1 The relative number of multiplications on 10^2 to 10^{14} instances compared to the Random Exponents Protocol [3] for $\lambda = 128$ and the optimal value of the parameter k in the Bucket Protocol.

Figure 3.2 The absolute number of multiplications on 10^2 to 10^{14} instances for $\lambda = 128$ and the optimal value of the parameter k in the Bucket Protocol.

Protocol	# multiplications	# proofs
No batching	-	m
Random Subsets	λm	λ
Random Exponents	$(3\lambda + 2)m$	1
Hybrid	$\lambda(m + 3\lambda + 2)$	1
Bucket	$\left\lceil \frac{\lambda}{k-2} \right\rceil \left(2m + (3k+2)2^k + (3\lambda + 2)\right)$	1

Table 3.1 The complexity of various batch PoEs for m instances with security parameter λ, and 2^k buckets in the Bucket Protocol.

Hybrid and Bucket protocols outperform the Random Exponents Protocol (Hybrid protocol is roughly 2 times faster, Bucket roughly 3 times). At around 10^5 to 10^6 instances the Hybrid protocol becomes 3 times faster than the Random Exponents Protocol, as also expected from Table 3.1. Starting from 10^7 instances, the Bucket Protocol shows the decrease of the expected group multiplications by an order of magnitude (compared to the Random Exponents Protocol). Thanks to the optimal choice of the parameter k in the Bucket Protocol, the difference of performances of the Bucket and the Random Exponents protocol continues to increase.

3.2 Experiments

As a part of this thesis, we implemented the Random Exponents Protocol, the Hybrid Protocol, the Bucket Protocol and the sequential version of the Random Subsets Protocol in C++ [14]. This implementation is based on the IOTA Network Team implementation [6] of Wesolowski's PoE. To run the experiments, we select the number of batching instances m and then:

1. randomly choose an RSA group with 2048-bits modulus $N = pq$ (p and q are 1024-bits primes),

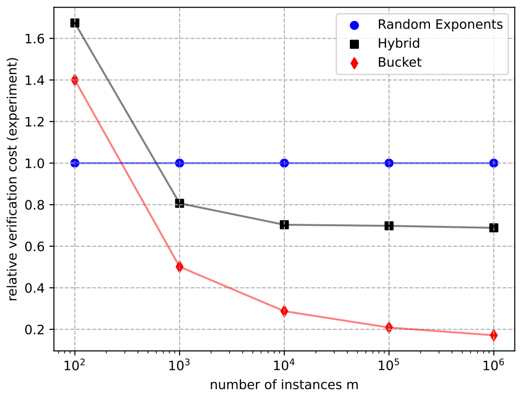


Figure 3.3 Experiments result: the relative times on 10^2 to 10^{14} instances compared to the Random Exponents Protocol [3] for $\lambda = 128$ and the optimal value of the parameter k in the Bucket Protocol.

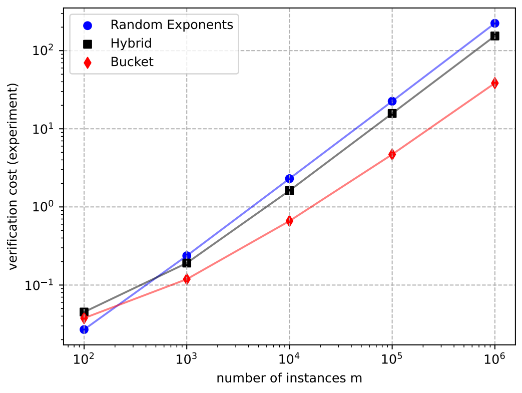


Figure 3.4 Experiments result: the times in seconds on 10^2 to 10^{14} instances for $\lambda = 128$ and the optimal value of the parameter k in the Bucket Protocol.

# instances	Random Exponents	Hybrid	Bucket
100	0.027	0.045	0.038
1000	0.237	0.191	0.119
10000	2.30	1.62	0.661
100000	22.5	15.7	4.70
1000000	224	154	38.4

Table 3.2 The experimental evaluation of time (in seconds) of various batch PoEs for m instances with security parameter λ, and 2^k buckets in the Bucket Protocol.

2. sample m PoE-statements of form $x_i^{2^T} \overset{?}{=} y_i$, where $T = 2^{25}$, $x_i, y_i \in \mathbb{G}, i \in [m]$,
3. set the security parameter $\lambda = 128$,
4. choose the optimal value of the parameter k for the Bucket Protocol,
5. run the comparison: for the sampled PoE-statements, we sequentially run the mentioned protocols (except for the Random Subsets Protocol) and measure the time spent by performing multiplication and exponentiation on the verifier's side.

For every value of m, we run the described experiment several times to eliminate the abnormal situations and then calculate the mean value of the received results for each protocol. The number of instances m that we perform the experiments for are $10^2, 10^3, 10^4, 10^5, 10^6$. We run the experiments on a machine with a 12 core 3.70GHz Apple M2 Max processor with 32 GB of RAM.

Note that we do not perform the experiments for the Random Subsets Protocol, since it produces m PoE-proofs and, in our implementation, the protocol is not parallelized. We also measure only the multiplications costs on the verifier's side for each protocol (including the verification part in Wesolowski's PoE) due to the following reasons:

1. In the theoretical analysis of the protocols, we focus on the expected number of multiplications, and we want to test it in practice.

2. Other steps, for example the sampling of the random subsets, are implementation-dependent. However, they might slow down the Hybrid and the Bucket protocols in practice.

The results of the experiments are presented in Figure 3.4 and Figure 3.3, the exact running times can be found in Table 3.2.

When comparing Figure 3.3 and Figure 3.1, we see that the Bucket Protocol behaves as expected with a small overhead, which becomes less and less visible as the number of instances increase.

The Hybrid Protocol turns out slower than expected (in Figure 3.3 it approaches approximately $0.7\times$ the time of Random Exponents, while in Figure 3.1 it is approximately $0.38\times$ the time of Random Exponents). Since we only measured the time connected to the group multiplications and exponentiations, it is an unexpected result. We believe the reason for such behaviour is the randomized memory access when calculating the product of the instances in the generated random subset. Let us explain it in more details. On every of the ρ iterations in the Hybrid Protocol, we choose a random subset of $[m]$. This subset consists of the indices of the statements. When calculating the product of the statements, corresponding to this subset, we need to go through the generated indices and multiply the correspondingly chosen instances. Each such accessing of the instance with the index i (where i is the randomly chosen index, because it is the element of the random subset) results in the random memory access x_i, y_i of the vectors/arrays of big memory size.

Note that both the Random Exponents Protocol and the Bucket Protocol do not have such overhead in our evaluations, as in the Random Exponents case, all the memory access to the arrays/vectors of x's and y's are sequential, and in the Bucket case, the random memory access is performed while distributing the instances between the buckets, but it is not included in our measurements, since we measure only the cost of multiplications and exponentiations.

Conclusion

Our contributions

In this work, we presented two new batching approaches for PoEs - the Hybrid Protocol and the Bucket Protocol. These approaches are inspired by the protocols of Rotem [3] and the techniques of Bellare, Garay and Rabin [4]. We calculated the expected number of group multiplications in verification, which roughly corresponds to the verification costs, for both our approaches and the protocols by Rotem - the Random Subsets Protocol and the Random Exponents Protocol [3]. Based on this estimation of verification costs, we found out that both the Hybrid and the Bucket protocols outperform the Random Subsets and the Random Exponents protocols. We also implemented the Hybrid, Bucket and the Random Exponents protocols in C++ and measured the time of group multiplications and exponentiations performed by these protocols.

Based on our theoretical evaluation, the Hybrid Protocol outperforms the Random Exponents protocol roughly by the order of 3, starting from 10^4 batching instances. However, the practical measurement from our implementation shows that the Hybrid Protocol is only 1.4 times faster (here we mean the measured time of the group multiplications and exponentiations) than the Random Exponents Protocol. We believe that the reason for such behaviour is the randomized memory access caused by the usage of the Random Subsets technique in the Hybrid Protocol, while in the Random Exponents Protocol most of the memory accesses are sequential.

The Bucket Protocol outperforms the Random Exponents Protocol by an order of magnitude both in theory and in practice. For the large numbers of instances (batch PoEs are motivated by practical applications with millions of instances), the gap between the performance of the Bucket Protocol and the Random Exponents Protocol increases due to the optimal choice of the parameter k in the Bucket Protocol.

We note that, while we were measuring only the expected number of group multiplications for each protocol, there are some non-trivial overheads that were not measured by such method. In particular, the generation of the random subsets in the Hybrid Protocol, the distribution of the instances between the buckets in the Bucket protocol and the generation of the random exponents for all three approaches. However, the optimal implementation of the protocols could minimize some of these overheads. The practically optimal implementation of the discussed protocols still remains an open question.

Various open problems are left for further research, such as:

1. Aiming at more efficient batching protocols. Is it possible to construct a batch PoE with strictly linear or sublinear verification?

2. The batching protocols for the PoE-statements with different exponents. All the discussed protocols relay on the fact that we can multiply the PoE-statements together and still obtain a PoE-statement with the same exponent. However this is not the case for the PoE-statements with different exponents. In [15], the batch PoE for instances with varying exponents was

presented, but it relies on Pietrzak's PoE, and therefore can not be used with different PoE (for example, Wesolowski's PoE).

Bibliography

1. Pietrzak, Krzysztof. Simple Verifiable Delay Functions. In: Blum, Avrim (ed.). *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, vol. 124, 60:1–60:15. LIPIcs. Available from DOI: `10.4230/LIPICS.ITCS.2019.60`.
2. Wesolowski, Benjamin. Efficient Verifiable Delay Functions. *J. Cryptol.* 2020, vol. 33, no. 4, pp. 2113–2147. Available from DOI: `10.1007/S00145-020-09364-X`.
3. Rotem, Lior. Simple and Efficient Batch Verification Techniques for Verifiable Delay Functions. In: Nissim, Kobbi; Waters, Brent (eds.). *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III*. Springer, 2021, vol. 13044, pp. 382–414. Lecture Notes in Computer Science. Available also from: `https://eprint.iacr.org/2021/1209`.
4. Bellare, Mihir; Garay, Juan A.; Rabin, Tal. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In: Nyberg, Kaisa (ed.). *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*. Springer, 1998, vol. 1403, pp. 236–250. Lecture Notes in Computer Science. Available from DOI: `10.1007/BFB0054130`.
5. Hoffmann, Charlotte; Hubáček, Pavel; Ivanova, Svetlana. Practical Batch Proofs of Exponentiation. *IACR Cryptol. ePrint Arch.* 2024, p. 145. Available also from: `https://eprint.iacr.org/2024/145`.
6. Attias, Vidal; Vigneri, Luigi; Dimitrov, Vassil. VDF Simulator. *IOTA Foundation*. 2020. Available also from: `https://github.com/iotaledger/vdf/blob/master`.
7. Seres, István András; Burcsi, Péter. A Note on Low Order Assumptions in RSA groups. *IACR Cryptol. ePrint Arch.* 2020, p. 402. Available also from: `https://eprint.iacr.org/2020/402`.
8. Belabas, Karim; Kleinjung, Thorsten; Sanso, Antonio; Wesolowski, Benjamin. A note on the low order assumption in class group of an imaginary quadratic number fields. *IACR Cryptol. ePrint Arch.* 2020, p. 1310. Available also from: `https://eprint.iacr.org/2020/1310`.
9. Fiat, Amos; Shamir, Adi. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko, Andrew M. (ed.). *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*. Springer, 1986, vol. 263, pp. 186–194. Lecture Notes in Computer Science. Available from DOI: `10.1007/3-540-47721-7\_12`.
10. Boneh, Dan; Bonneau, Joseph; Bünz, Benedikt; Fisch, Ben. Verifiable Delay Functions. *IACR Cryptol. ePrint Arch.* 2018, p. 601. Available also from: `https://eprint.iacr.org/2018/601`.
11. Chia Network Inc. Chia Network. 2023. Available also from: `http://chia.net`.

12. Boneh, Dan; Bünz, Benedikt; Fisch, Ben. A Survey of Two Verifiable Delay Functions. *IACR Cryptol. ePrint Arch.* 2018, p. 712. Available also from: `https://eprint.iacr.org/2018/712`.

13. Attias, Vidal; Vigneri, Luigi; Dimitrov, Vassil S. Efficient Verification of the Wesolowski Verifiable Delay Function for Distributed Environments. *IACR Cryptol. ePrint Arch.* 2022, p. 520. Available also from: `https://eprint.iacr.org/2022/520`.

14. Ivanova, Svetlana. Implementation of the Random Exponents, Bucket and Hybrid protocols. 2024. Available also from: `https://github.com/i-v-lana/batch-poe-implementation`.

15. Hoffmann, Charlotte; Hubáček, Pavel; Kamath, Chethan; Klein, Karen; Pietrzak, Krzysztof. Practical Statistically-Sound Proofs of Exponentiation in Any Group. In: Dodis, Yevgeniy; Shrimpton, Thomas (eds.). *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II.* Springer, 2022, vol. 13508, pp. 370–399. Lecture Notes in Computer Science. Available from doi: `10.1007/978-3-031-15979-4\_13`.

List of Figures

List of Tables