Efficient Techniques for Publicly Verifiable Delegation of Computation

Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui, Refik Molva
EURECOM
Campus SophiaTech
Biot Sophia Antipolis, France
{elkhiyao, onen, azraoui, molva}@eurecom.fr

ABSTRACT

With the advent of cloud computing, individuals and companies alike are looking for opportunities to leverage cloud resources not only for storage but also for computation. Nevertheless, the reliance on the cloud to perform computation raises the unavoidable challenge of how to assure the correctness of the delegated computation. In this regard, we introduce two cryptographic protocols for publicly verifiable computation that allow a lightweight client to securely outsource to a cloud server the evaluation of high-degree univariate polynomials and the multiplication of large matrices. Similarly to existing work, our protocols follow the amortized verifiable computation approach. Furthermore, by exploiting the mathematical properties of polynomials and matrices, they are more efficient and give way to *public delegatability*. Finally, besides their efficiency, our protocols are provably secure under well-studied assumptions.

1. INTRODUCTION

Cloud computing is increasingly becoming an attractive option for SMEs interested in minimizing their expenditures by outsourcing their data and computations. However, the lack of security still deters the wide adoption of cloud technology. As a matter of fact, cloud clients lose control over their data once outsourced, and as such they can neither thwart nor detect cloud servers' misbehavior.

Recently, researchers [6, 11, 13, 17, 19] introduced solutions for verifiable outsourced computation whereby a client delegates the execution of computationally demanding operations to the cloud, and further receives the result with some *cryptographic proofs* asserting the correct execution of requested operations. By definition, these cryptographic proofs fulfill the classical security requirements of *correctness* and *soundness*: They neither yield a situation in which a server is *falsely accused* of misbehavior, nor make the client accept an *incorrect result*.

In addition to the previously mentioned security requirements, another key prerequisite that should be taken into account when designing solutions for verifiable computation is the *efficiency* of the proof verification at the client: For a solution to be viable, the computational and the storage complexity of the verification process should naturally be lower than the complexity of the outsourced function. This requirement thus seeks solutions that minimize the computational and the storage load at lightweight clients, in the aim of not offsetting the advantages of cloud computing.

In order to be able to check the proof of correct computation efficiently, the client generates a verification key: While some solutions [6, 13] keep this verification key secret, in which case only the client verifies the correctness of the outsourced computation, other proposals [11, 17–19] allow *public verifiability* which empowers any third party to verify the validity of the outsourced computation.

Besides public verifiability, several schemes achieve *public delegatability* [17–19]. As the name implies, public delegatability enables any third party to submit computation queries to the outsourced function and verify the returned results. Such a property comes in handy in scenarios where an organization outsources the computation of a function to a cloud server, and still wants its employees to delegate the evaluation of that function without exchanging or sharing any secret keys.

In this paper, we focus on the public verifiability and delegatability of two specific functions, namely, high-degree polynomial evaluation and matrix multiplication. Similarly to existing work, we adopt the *amortized model* [13]: In this model, the client is required to execute a one-time expensive pre-processing operation that is leveraged later for efficient verifications. Furthermore, we suitably tailor the algebraic properties of polynomials and matrices to devise cryptographic solutions that compare favorably to existing work, as they offer better performances and contrary to [11, 20] enable public delegatability.

Contributions:

- We first propose a publicly verifiable polynomial evaluation solution whose efficiency derives from the *Euclidean division* of the polynomial to be outsourced by some randomly generated small-degree polynomial. The basic idea of our solution is that the outsourced polynomial and the quotient polynomial are used to produce the proof of correct computation, whereas the divisor and the remainder polynomials are used together to verify the correctness of the evaluation. Thanks to the properties of Euclidean division, our proposal ensures public delegatability while enjoying better performances than existing work [17].

- Secondly, we propose a solution for publicly verifiable matrix multiplication that exploits the associative property of multiplication in the ring of matrices. As such our solution outperforms the schemes in [11, 20] while ensuring the additional feature of public delegatability.

- Both of our solutions are proved to be correct and sound. Their soundness is proved under the *t-strong Diffie Hellman*

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China
© 2016 ACM. ISBN 978-1-4503-4233-9/16/05. . . $15.00
DOI: http://dx.doi.org/10.1145/2897845.2897910

(t-SDH) and *co-computational Diffie-Hellman (co-CDH)* assumptions.

The rest of the paper is organized as follows. Section II formally defines publicly verifiable computation and the underlying security model. The proposed publicly verifiable polynomial evaluation and matrix multiplication solutions are described and evaluated in Sections III and IV respectively. Finally, we review the state of the art in section V.

2. BACKGROUND

2.1 Publicly Verifiable Computation

According to [18], a publicly verifiable computation scheme empowers a client to outsource the evaluation of a function to a *potentially malicious* server while meeting the requirements of:

- *public delegatability:* Any querier (not necessarily the client) can submit inputs to evaluate the outsourced function;
- *public verifiability:* Any verifier (not necessarily the client or the querier) can assess the correctness of the server's results.

Thus, Parno et al. [18] formally define publicly verifiable computation schemes by the following algorithms:

$\mathsf{Setup}(1^\kappa, \mathfrak{f}) \to (\mathsf{param}, \mathsf{PK}_\mathfrak{f}, \mathsf{EK}_\mathfrak{f})$ It is a randomized algorithm executed by the client. It takes as input the security parameter 1^κ and a description of the function $\mathfrak{f}$ to be outsourced, and outputs a set of *public parameters* param that will be used by subsequent algorithms, a *public key* $\mathsf{PK}_\mathfrak{f}$, and an *evaluation key* $\mathsf{EK}_\mathfrak{f}$.

$\mathsf{ProbGen}(x, \mathsf{PK}_\mathfrak{f}) \to (\sigma_x, \mathsf{VK}_x)$ Given an input x in the domain $\mathcal{D}_\mathfrak{f}$ of the outsourced function $\mathfrak{f}$ and public key $\mathsf{PK}_\mathfrak{f}$, the querier calls this algorithm to produce an *encoding* σ_x of input x and a *public verification key* VK_x.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f}) \to \sigma_y$ On input of the encoding σ_x and the evaluation key $\mathsf{EK}_\mathfrak{f}$, the server runs this algorithm to compute an *encoding* σ_y of $\mathfrak{f}$'s output $y = \mathfrak{f}(x)$.

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathsf{out}_y$ A verifier operates this deterministic algorithm to check the correctness of the result σ_y supplied by the server on input σ_x. More precisely, this algorithm first decodes σ_y which yields a value y, and then uses the public verification key VK_x associated with the encoding σ_x to decide whether y is equal to the expected output $\mathfrak{f}(x)$. If so, Verify outputs $\mathsf{out}_y = y$ meaning that $\mathfrak{f}(x) = y$; otherwise it outputs an error $\mathsf{out}_y = \perp$.

Besides the properties of public delegatability and verifiability, a publicly verifiable computation scheme should also ensure the security properties of *correctness* and *soundness*.

2.2 Correctness

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is deemed to be correct, if whenever an *honest* server executes the algorithm $\mathsf{Compute}$ to evaluate a function $\mathfrak{f} \in \mathcal{F}$ on an input $x \in \mathcal{D}_\mathfrak{f}$, this algorithm *always* yields an encoding σ_y that will be accepted by algorithm Verify (i.e. $\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathfrak{f}(x)$).

Definition 1. *A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is correct,* **iff** *for any function $\mathfrak{f} \in \mathcal{F}$ and any input $x \in \mathcal{D}_\mathfrak{f}$:*

If $\mathsf{ProbGen}(x, \mathsf{PK}_\mathfrak{f}) \to (\sigma_x, \mathsf{VK}_x)$ *and* $\mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f}) \to \sigma_y$*, then:*

$$\Pr(\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathfrak{f}(x)) = 1$$

Algorithm 1: Soundness experiment of publicly verifiable computation

```
$(\mathsf{param}, \mathsf{PK}_\mathfrak{f}, \mathsf{EK}_\mathfrak{f}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\kappa, \mathfrak{f})$;
$\mathcal{A} \to x$;
$(\sigma_x, \mathsf{VK}_x) \leftarrow \mathcal{O}_{\mathsf{ProbGen}}(x, \mathsf{PK}_\mathfrak{f})$;
$\mathcal{A} \to \sigma_y$;
$\mathsf{out}_y \leftarrow \mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$;
```

2.3 Soundness

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is said to be sound, if for any $\mathfrak{f} \in \mathcal{F}$ and for any $x \in \mathcal{D}_\mathfrak{f}$, a server cannot convince a verifier to accept an incorrect result. Notably, a verifiable computation scheme is sound if it assures that the only way a server generates a result σ_y that will be accepted by a verifier as a valid encoding of the evaluation of some function $\mathfrak{f} \in \mathcal{F}$ on an input x, is by correctly computing σ_y (i.e. $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f})$).

Similarly to [18], we capture the adversarial capabilities of an adversary (i.e. malicious server) $\mathcal{A}$ against a publicly verifiable computation scheme for a family of functions $\mathcal{F}$ through a *soundness experiment* (cf. Algorithm 1).

In this experiment, adversary $\mathcal{A}$ first accesses the output of algorithm Setup by calling oracle $\mathcal{O}_{\mathsf{Setup}}$. When queried with a security parameter 1^κ and a description of a function $\mathfrak{f} \in \mathcal{F}$, oracle $\mathcal{O}_{\mathsf{Setup}}$ returns the set of public parameters param, public key $\mathsf{PK}_\mathfrak{f}$, and evaluation key $\mathsf{EK}_\mathfrak{f}$.

Afterwards, adversary $\mathcal{A}$ outputs a challenge input $x \in \mathcal{D}_\mathfrak{f}$ and submits the latter together with public key $\mathsf{PK}_\mathfrak{f}$ to oracle $\mathcal{O}_{\mathsf{ProbGen}}$. Oracle $\mathcal{O}_{\mathsf{ProbGen}}$ accordingly executes algorithm $\mathsf{ProbGen}$ and outputs a pair of matching encoding σ_x and public verification key VK_x.

Finally, adversary $\mathcal{A}$ generates an encoding σ_y and runs algorithm Verify on the pair $(\sigma_y, \mathsf{VK}_x)$.

Let out_y denote the output of algorithm Verify at the end of the experiment. We say that adversary $\mathcal{A}$ succeeds in the soundness experiment of publicly verifiable computation if $\mathsf{out}_y \neq \perp$ and $\mathsf{out}_y \neq \mathfrak{f}(x)$.

Definition 2. *Let $\Pi_{\mathcal{A},\mathfrak{f}}$ denote the probability that adversary $\mathcal{A}$ succeeds in the soundness experiment of publicly verifiable computation (i.e. $\Pr(\mathsf{out}_y \neq \perp \ \wedge \ \mathsf{out}_y \neq \mathfrak{f}(x))$).*

A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is sound, **iff***: For any adversary $\mathcal{A}$ and for any $\mathfrak{f} \in \mathcal{F}$, $\Pi_{\mathcal{A},\mathfrak{f}} \leqslant \epsilon$ and ϵ is a negligible function in the security parameter κ.*

3. PUBLICLY VERIFIABLE POLYNOMIAL EVALUATION

3.1 Protocol Overview

The solution we propose for publicly verifiable evaluation of polynomials draws upon the basic properties of *Euclidean division* of polynomials. Specifically the fact that for any pair of polynomials A and $B \neq 0$ of degree d and 2 respectively, the Euclidean division of A by B yields a unique pair of polynomials Q and R such that: **i.)** $A = QB + R$ and **ii.)** the degree of *quotient* polynomial Q equals $d - 2$, whereas the *remainder* polynomial R has a degree $\leqslant 1$.

Now a client which would like to outsource the evaluation of a polynomial A of degree d, first defines a polynomial $B(X) = X^2 + b_0$ for a randomly chosen b_0, and divides A by B to get the quotient polynomial $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and the remainder polynomial $R(X) = r_1 X + r_0$. Next, the client outsources polynomial A together with quotient polynomial Q to the server and publishes the public key $\mathsf{PK}_A = (g^{b_0}, g^{r_1}, g^{r_0})$. Consequently, whenever a querier wants to evaluate polynomial A at point x, it first computes and advertises the public verification key $\mathsf{VK}_x = (g^{B(x)}, g^{R(x)})$, and then transmits x to the server. The latter in turn computes $y = A(x)$ and generates the proof $\pi = Q(x)$. Given the server's output (y, π), a verifier checks whether $g^y = (g^{B(x)})^{\pi} g^{R(x)}$.

The efficiency of the verification in the solution sketched above stems from the fact that B and R are small-degree polynomials. Indeed, to verify the correctness of a result (y, π) provided by the server on an input x, the verifier performs a small and constant number of computations as opposed to carrying out $\mathcal{O}(d)$ operations to evaluate polynomial A.

It is clear that the soundness of such a protocol relies on the secrecy of polynomials B and R. However since B is a two-degree polynomial, the secrecy of these two polynomials can be easily compromised by disclosing the quotient polynomial Q. To remedy this shortcoming, the client encodes polynomial Q using an *additively homomorphic one-way* encoding. Namely, each coefficient q_i of polynomial Q is encoded as h^{q_i}. In this manner, we allow the server to compute the proof $\pi = h^{Q(x)}$ of correct execution while ensuring the confidentiality of polynomials B and R.

Finally, we use bilinear pairings to let verifiers assess the correctness of the server's results. Accordingly, we show that our solution is sound under the $\lfloor d/2 \rfloor$-*Strong Diffie-Hellman* ($\lfloor d/2 \rfloor$-SDH) assumption.

Before describing our protocol in full details, we recall the definitions of bilinear pairings and the SDH assumption.

3.2 Bilinear Pairings

Definition 3 (Bilinear Pairing)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite order p.*

A bilinear pairing is a map e: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, with the following properties:

1. *e is bilinear: $\forall\, \alpha, \beta \in \mathbb{Z}_p$, $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, $e(g^{\alpha}, h^{\beta}) = e(g, h)^{\alpha\beta}$;*
2. *e is computable: There is an efficient algorithm to compute $e(g, h)$ for any $(g, h) \in \mathbb{G}_1 \times \mathbb{G}_2$;*
3. *e is non-degenerate: If g is a generator of $\mathbb{G}_1$ and h is a generator of $\mathbb{G}_2$, then $e(g, h)$ is a generator of $\mathbb{G}_T$.*

Definition 4 (t-SDH Assumption)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite prime order p such that there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*

We say that the t-**Strong Diffie-Hellman assumption** *(t-SDH) holds, if given the tuple $(g, g^{\alpha}, h, h^{\alpha}, ..., h^{\alpha^t}) \in \mathbb{G}_1^2 \times \mathbb{G}_2^{t+1}$ for some randomly chosen $\alpha \in \mathbb{F}_p^*$, the probability to produce a pair $(\beta, h^{1/(\beta+\alpha)}) \in \mathbb{F}_p \backslash \{-\alpha\} \times \mathbb{G}_2$ is negligible.*

3.3 Description

We assume here that the client wants to outsource the evaluation of a d-degree polynomial $A(X) = \sum_{i=0}^{d} a_i X^i$ with coefficients $a_i \in \mathbb{F}_p$ where p is a large prime.

$\mathsf{Setup}(1^{\kappa}, A)$ Given security parameter 1^{κ} and a description of polynomial A, algorithm Setup first selects two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Then it picks a generator g and a generator h of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and defines the set of public parameters as:

$$\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h).$$

Next, algorithm Setup selects randomly $b_0 \in \mathbb{F}_p^*$ such that polynomial $B(X) = X^2 + b_0$ does not divide polynomial A and performs the Euclidean division of polynomial A by polynomial B in $\mathbb{F}_p[X]$. We denote the resulting quotient polynomial by $Q(X) = \sum_{i=0}^{d-2} q_i X^i$ and the resulting remainder polynomial by $R(X) = r_1 X + r_0$[1].

Thereupon, algorithm Setup computes the public key

$$\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) = (g^{b_0}, h^{r_1}, h^{r_0}).$$

To compute evaluation key EK_A algorithm Setup computes $\mathbf{q}_i = h^{q_i} \in \mathbb{G}_2$ for all $0 \leqslant i \leqslant d-2$, and lets

$$\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2}).$$

Algorithm Setup concludes its execution by outputting the tuple $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$.

$\mathsf{ProbGen}(x, \mathsf{PK}_A)$ On input of a point $x \in \mathbb{F}_p$ and public key $\mathsf{PK}_A = (\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0)$, algorithm $\mathsf{ProbGen}$ first computes

$$\mathsf{VK}_{(x,B)} = \mathbf{b_0} g^{x^2}$$
$$\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0$$

and then outputs the public encoding $\sigma_x = x$ and the public verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_A)$ Given $\sigma_x = x$ and evaluation key $\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-2})$, algorithm $\mathsf{Compute}$ evaluates

$$y = A(x) = \sum_{i=0}^{d} a_i x^i \mod p,$$

generates the proof

$$\pi = \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i},$$

and outputs the encoding $\sigma_y = (y, \pi)$.

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$ Provided with encoding $\sigma_y = (y, \pi)$ and verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$, algorithm Verify checks whether the following equation holds:

$$e(g, h^y) = e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)}). \qquad (1)$$

If so, then Verify outputs y meaning that $A(x) = y$; otherwise it outputs $\perp$.

3.4 Security Analysis

Here we state and prove the main security theorems pertaining to our protocol for publicly verifiable polynomial evaluation.

Theorem 1. *The scheme proposed above for publicly verifiable polynomial evaluation is correct.*

[1] R is a polynomial of degree at most 1, i.e. r_1 could be 0.

Proof. If on input $\sigma_x = x \in \mathbb{F}_p$, the server executes algorithm Compute correctly, then the latter's output will correspond to

$$\sigma_y = (y, \pi) = (A(x), h^{Q(x)}).$$

Indeed, we have:

$$\pi = \prod_{i=0}^{d-2} \mathbf{q}_i^{x^i} = \prod_{i=0}^{d-2} h^{q_i x^i} = h^{\sum_{i=0}^{d-2} q_i x^i} \\ = h^{Q(x)}.$$

Given that $A = QB + R$ in $\mathbb{F}_p[X]$ and that the order of $e(g, h)$ is equal to p, we get:

$$e(g,h)^{A(x)} = e(g,h)^{Q(x)B(x)+R(x)} \\ = e(g, h^{Q(x)})^{B(x)} e(g,h)^{R(x)}.$$

As $y = A(x)$ and $\pi = h^{Q(x)}$ we have:

$$e(g,h)^y = e(g,\pi)^{B(x)} e(g,h)^{R(x)} \\ = e(g^{B(x)}, \pi) e(g, h^{R(x)}).$$

Since

$$\mathsf{VK}_{(x,B)} = \mathbf{b}_0 g^{x^2} = g^{b_0 + x^2} = g^{B(x)}$$

and

$$\mathsf{VK}_{(x,R)} = \mathbf{r}_1^x \mathbf{r}_0 = h^{r_1 x + r_0} = h^{R(x)},$$

we conclude that

$$e(g,h)^y = e(\mathsf{VK}_{(x,B)}, \pi) e(g, \mathsf{VK}_{(x,R)})$$

and that Verify outputs $y = A(x)$. ☐

Theorem 2. *The scheme proposed above for publicly verifiable polynomial evaluation is sound under the $\lfloor d/2 \rfloor$-SDH assumption.*

Proof. Assume there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable polynomial evaluation with a non-negligible advantage ϵ. We demonstrate in what follows that there exists another adversary $\mathcal{B}$ that breaks the $\lfloor d/2 \rfloor$-SDH assumption with a non-negligible advantage $\geqslant \epsilon$.

Let $\mathcal{O}_{\mathsf{sdh}}$ be an oracle which when queried returns the pair (g, g^α) in $\mathbb{G}_1$ and the tuple $(h, h^\alpha, h^{\alpha^2}, ..., h^{\alpha^{\lfloor d/2 \rfloor}})$ in $\mathbb{G}_2$ for randomly generated α in $\mathbb{F}_p^*$.

In order to break $\lfloor d/2 \rfloor$-SDH, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{sdh}}$ to obtain a tuple $(g, g^\alpha, h, h^\alpha, ..., h^{\alpha^{\lfloor d/2 \rfloor}})$; then simulates the soundness experiment (see Algorithm 1) to adversary $\mathcal{A}$. Namely, when $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with polynomial $A(X) = \sum_{i=0}^{d} a_i X^i$ in $\mathbb{F}_p[X]$, adversary $\mathcal{B}$ simulates $\mathcal{O}_{\mathsf{Setup}}$'s response as follows:

1. It defines the public parameters

$$\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$$

2. To compute the evaluation key $\widehat{\mathsf{EK}}_A = (A, \widehat{\mathbf{q}}_0, ..., \widehat{\mathbf{q}}_{d-2})$, it proceeds as described below:
   - It lets $\widehat{\mathbf{q}}_{d-2} = h^{a_d}$ and $\widehat{\mathbf{q}}_{d-3} = h^{a_{d-1}}$;
   - For each $2 \leqslant k \leqslant d-2$, it computes

$$\widehat{\mathbf{q}}_{d-2-k} = \prod_{i=0}^{\lfloor k/2 \rfloor} h^{a_{d-k+2i}(-1)^i \alpha^i}$$

3. It computes the public key $\widehat{\mathsf{PK}}_A = (\widehat{\mathbf{b}}_0, \widehat{\mathbf{r}}_1, \widehat{\mathbf{r}}_0)$ as following:

$$\widehat{\mathbf{b}}_0 = g^\alpha \\ \widehat{\mathbf{r}}_0 = \prod_{i=0}^{\lfloor d/2 \rfloor} h^{a_{2i}(-1)^i \alpha^i} \\ \widehat{\mathbf{r}}_1 = \prod_{i=0}^{\lfloor (d-1)/2 \rfloor} h^{a_{2i+1}(-1)^i \alpha^i}.$$

   If $(\widehat{\mathbf{r}}_0, \widehat{\mathbf{r}}_1) = (1,1)$, then adversary $\mathcal{B}$ stops the experiment.

4. Otherwise, it returns public parameters $\widehat{\mathsf{param}}$, evaluation key $\widehat{\mathsf{EK}}_A$ and public key $\widehat{\mathsf{PK}}_A$ to adversary $\mathcal{A}$.

It can easily be shown that if adversary $\mathcal{B}$ does not stop the experiment, then the distribution of the tuple $(\widehat{\mathsf{param}}, \widehat{\mathsf{PK}}_A, \widehat{\mathsf{EK}}_A)$ returned by adversary $\mathcal{B}$ is *statistically indistinguishable* from the distribution of $(\mathsf{param}, \mathsf{PK}_A, \mathsf{EK}_A)$ in the soundness experiment. As a matter of fact, if we denote for all $0 \leqslant i \leqslant d-2$, $\widehat{\mathbf{q}}_i = h^{q_i}$ and if we let $(\widehat{\mathbf{r}}_0, \widehat{\mathbf{r}}_1) = (h^{r_0}, h^{r_1})$, then we can easily verify that:

- $a_d = q_{d-2} \mod p$ and $a_{d-1} = q_{d-3} \mod p$;
- for all $2 \leqslant i \leqslant d-2$, $a_i = \alpha q_i + q_{i-2} \mod p$;
- $a_1 = \alpha q_1 + r_1 \mod p$ and $a_0 = \alpha q_0 + r_0 \mod p$;
- $(r_0, r_1) \neq (0, 0)$.

This entails that the polynomials defined as $Q(X) = \sum_{i=0}^{d-2} q_i X^i$, $B(X) = X^2 + \alpha$ and $R(X) = r_1 X + r_0$ verify the following equality: $A = BQ + R$ with $R \neq 0$.

Therefore we can safely conclude (i) that polynomial B does not divide polynomial A; (ii) that each $\widehat{\mathbf{q}}_i$ correctly encodes the i^{th} coefficient of the quotient polynomial Q that results from the Euclidean division of polynomial A by polynomial B; (iii) that the pair $(\widehat{\mathbf{r}}_0, \widehat{\mathbf{r}}_1)$ correctly encodes the corresponding remainder polynomial R.

Eventually, adversary $\mathcal{A}$ selects a challenge value $x \in \mathbb{F}_p$ and calls oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(x, \widehat{\mathsf{PK}}_A)$. Accordingly, adversary $\mathcal{B}$ computes the response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and returns verification key

$$\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)}) = (\widehat{\mathbf{b}}_0 g^{x^2}, \widehat{\mathbf{r}}_0 \widehat{\mathbf{r}}_1^x).$$

Finally, adversary $\mathcal{A}$ returns a pair (y, π) such that $y \neq A(x)$ and (y, π) is accepted by algorithm Verify with a non-negligible advantage ϵ.

Consequently, adversary $\mathcal{B}$ breaks $\lfloor d/2 \rfloor$-SDH by first computing $A(x)$ and the proof

$$\pi^* = \prod_{i=0}^{d-2} \widehat{\mathbf{q}}_i^{x^i}$$

and finally outputting:

$$(\beta, h^{1/(\beta+\alpha)}) = \left(x^2, \left(\frac{\pi}{\pi^*} \right)^{(y-A(x))^{-1}} \right).$$

Indeed, since the pair (y, π) passes the verification, it satisfies Equation 1, namely:

$$e(g,h)^y = e(\widehat{\mathbf{b}}_0 g^{x^2}, \pi) e(g, \widehat{\mathbf{r}}_0 \widehat{\mathbf{r}}_1^x) = e(g^{x^2+\alpha}, \pi) e(g, \widehat{\mathbf{r}}_0 \widehat{\mathbf{r}}_1^x). \tag{2}$$

Furthermore, by construction:

$$e(g,h)^{A(x)} = e(g^{x^2+\alpha}, \pi^*) e(g, \widehat{\mathbf{r}}_0 \widehat{\mathbf{r}}_1^x). \tag{3}$$

Algorithm	Computation	Client's storage	Server's storage
Setup	1 prng and d mul in $\mathbb{F}_p$ 1 exp in $\mathbb{G}_1$ $d+1$ exp in $\mathbb{G}_2$	$\mathcal{O}(1)$	$\mathcal{O}(d)$
ProbGen	1 mul in $\mathbb{F}_p$ 1 exp and 1 mul in $\mathbb{G}_1$ 1 exp and 1 mul in $\mathbb{G}_2$	–	–
Compute	$2d-3$ mul in $\mathbb{F}_p$ $d-1$ exp and $d-2$ mul in $\mathbb{G}_2$	–	–
Verify	1 exp and 1 div in $\mathbb{G}_2$ 2 pairings	–	–

Table 1: Computation and storage requirements of our protocol for publicly verifiable polynomial evaluation

By dividing Equation 2 by 3, we obtain:

$$e(g,h)^{(y-A(x))} = e\left(g^{x^2+\alpha}, \frac{\pi}{\pi^*}\right).$$

Since $y \neq A(x)$, the above equation implies:

$$e(g,h) = e\left(g^{x^2+\alpha}, \left(\frac{\pi}{\pi^*}\right)^{(y-A(x))^{-1}}\right).$$

Hence if adversary $\mathcal{B}$ does not stop the experiment, then it will be able to break the $\lfloor d/2 \rfloor$-SDH assumption.

Now if adversary $\mathcal{B}$ aborts the experiment which occurs when $(\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1) = (1,1)$, then adversary $\mathcal{B}$ can conclude that B divides A. This means that by using a factorization algorithm in $\mathbb{F}_p[X]$ on polynomial A, adversary $\mathcal{B}$ will be able to find α, and therewith, break the $\lfloor d/2 \rfloor$-SDH assumption.

Thus, we deduce that if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable polynomial evaluation with a non-negligible advantage ϵ, then there is an adversary $\mathcal{B}$ that breaks the $\lfloor d/2 \rfloor$-SDH assumption with a non-negligible advantage $\geqslant \epsilon$. □

Remark 1. *Notice that if $B(X) = X^\delta + b_0$, then using a similar argument as the one above, we can easily show that our protocol for verifiable polynomial evaluation is secure under the t-SDH assumption for $t \geqslant \lfloor d/\delta \rfloor$.*

3.5 Performance Analysis

The reader may refer to Table 1 for a summary of the performances of our protocol for publicly verifiable polynomial evaluation.

Algorithm Setup first generates a random coefficient $b_0 \in \mathbb{F}_p^*$ to construct polynomial B and conducts an Euclidean division of polynomial A by polynomial B. The latter operation consists of d multiplications and additions, where d is the degree of polynomial A. Once the Euclidean division is performed, algorithm Setup performs one exponentiation in $\mathbb{G}_1$ to derive $\mathbf{b}_0$, and $d+1$ exponentiations in $\mathbb{G}_2$ to compute $\mathbf{r}_0$, $\mathbf{r}_1$ and $\mathbf{q}_i$. Although computationally expensive, algorithm Setup is executed only once by the client. Besides, its computational cost is *amortized* over the large number of verifications that third-party verifiers can carry out.

On the other hand, algorithm ProbGen computes the verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$ which demands a constant number of operations that does not depend on the degree of polynomial A. More precisely, ProbGen's work consists of computing x^2 in $\mathbb{F}_p$, performing one exponentiation and one multiplication in $\mathbb{G}_1$ to get $\mathsf{VK}_{(x,B)} = g^{B(x)}$, and running one exponentiation and one multiplication in $\mathbb{G}_2$ to obtain $\mathsf{VK}_{(x,R)} = h^{R(x)}$.

Furthermore, algorithm Compute runs in two steps: (i) the evaluation of polynomial A at point x which requires at most d additions and multiplications in $\mathbb{F}_p$ if the server uses *Horner's rule*; and (ii) the generation of the proof π which involves $d-3$ multiplications in $\mathbb{F}_p$ and $d-1$ exponentiations and $d-2$ multiplications in $\mathbb{G}_2$.

Finally, the work at third-party verifiers only consists of one exponentiation and one division in $\mathbb{G}_2$ and the computation of 2 bilinear pairings.

With respect to storage, the client is required to store and publish the public key $(\mathbf{b}_0, \mathbf{r}_1, \mathbf{r}_0) \in \mathbb{G}_1 \times \mathbb{G}_2^2$. The server however keeps the $d+1$ coefficients $a_i \in \mathbb{F}_p$ of polynomial A and the $d-1$ encodings $\mathbf{q}_i \in \mathbb{G}_2$.

The reader may refer to Table 1 for a summary of the performances of our protocol for publicly verifiable polynomial evaluation.

4. PUBLICLY VERIFIABLE MATRIX MULTIPLICATION

4.1 Protocol Overview

The protocol we introduce in this section relies on the intuition already expressed in [11], which states that in order to verify that a server correctly multiplies an (n,m)-matrix M of elements M_{ij} with some column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$, it suffices that the client randomly picks a *secret* (n,m)-matrix R of elements R_{ij}, and supplies a server with (n,m)-matrix M and an auxiliary (n,m)-matrix $\mathcal{N}$ such that $\mathcal{N}_{ij} = \tilde{g}^{M_{ij}} g^{R_{ij}}$ (where $\tilde{g} = g^\delta$ for some randomly generated δ). Consequently, when a client prompts the server to multiply matrix M with vector $\vec{x}$, the latter returns vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ and proof $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)^\intercal$, such that $\pi_i = \tilde{g}^{y_i} g^{\sum_{j=1}^m R_{ij} x_j}$ if the server is honest. If we denote $\pi_i = g^{\gamma_i}$ and $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n)^\intercal$, then loosely speaking, the verification process consists of checking whether $\vec{\gamma} = \delta \vec{y} + R\vec{x}$.

Now to transform this intuition into a viable solution, one must ensure that the verification process is much less computationally demanding than the matrix multiplication $M\vec{x}$ for all vectors $\vec{x}$. In [11], the authors speed up the verification process by generating the secret matrix R using dedicated *algebraic PRFs* that optimize the multiplication $R\vec{x}$. Although this solution gives way to an efficient verification process that takes $\mathcal{O}(n+m)$ time, it does not enable public delegatability: Only the client can submit multiplication queries to the server.

We tackle this issue by observing that for any vector $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$, the verification of whether $\vec{\lambda}\vec{\gamma} = \delta\vec{\lambda}\vec{y} + \vec{\lambda}(R\vec{x})$ takes $\mathcal{O}(n)$ time if the vector $\vec{\lambda}R$ is computed beforehand. Therefore, we define the public key by an exponent encoding of $\vec{\lambda}R$, and the verification key for vector $\vec{x}$ by an exponent encoding of $(\vec{\lambda}R)\vec{x}$.

More concretely, we generate the elements in the auxiliary matrix $\mathcal{N}$ as $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$ for $g_i = g^{\lambda_i}$, we let the public key PK_M be a vector of m components $\mathsf{PK}_j = e(\prod_{i=1}^n g_i^{R_{ij}}, h)$, and we compute the verification key for vector $\vec{x}$ as $\mathsf{VK}_x = \prod_{j=1}^m \mathsf{PK}_j^{x_j}$. Therefore, the problem generation combined with the verification take $\mathcal{O}(n+m)$ time as opposed to performing $\mathcal{O}(nm)$ operations to compute the matrix multiplication $\vec{y} = M\vec{x}$.

As a result, the proposed solution does not only offer public delegatability, but also is sound under the assumption of co-computational Diffie-Hellman (*co-CDH*).

Definition 5 (co-CDH Assumption). *Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of the same finite prime order p such that there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.*

We say that the **co-computational Diffie-Hellman assumption** *(co-CDH) holds in $\mathbb{G}_1$, if given $g, g^\alpha \in \mathbb{G}_1$ and $h, h^\beta \in \mathbb{G}_2$ for random $\alpha, \beta \in \mathbb{F}_p^*$, the probability to compute $g^{\alpha\beta}$ is negligible.*

4.2 Protocol for Verifiable Matrix Multiplication

Without loss of generality, we assume that a client outsources to a server the multiplication operations involving an (n,m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ ($1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$) with p being a large prime.

$\mathsf{Setup}(1^\kappa, M)$ Given security parameter 1^κ and matrix M, algorithm Setup chooses two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. It then selects a generator h of group $\mathbb{G}_2$ and computes $\tilde{h} = h^\delta$ for a randomly selected δ in $\mathbb{F}_p^*$. Thereafter, it randomly picks n generators[2] g_i of $\mathbb{G}_1$, for all $1 \leqslant i \leqslant n$. Subsequently, algorithm Setup defines the public parameters associated with matrix M as:

$$\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{g_i\}_{1 \leqslant i \leqslant n}, h, \tilde{h}).$$

Afterwards, algorithm Setup computes the evaluation key EK_M as follows:

- It selects an (n,m)-random matrix R of elements R_{ij} in $\mathbb{F}_p^*$.
- It derives another (n,m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} = g_i^{\delta M_{ij} + R_{ij}}, \forall\, 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m$.
- Finally, it sets the evaluation key to
$$\mathsf{EK}_M = (M, \mathcal{N}).$$

Next, algorithm Setup determines public key PK_M as depicted hereafter:

- It generates m keys $\mathsf{PK}_j = e(\prod_{i=1}^n g_i^{R_{ij}}, h)$, $1 \leqslant j \leqslant m$.
- Then, it lets $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$.

At the end of its execution, algorithm Setup outputs public parameters param, public key PK_M and evaluation key EK_M.

$\mathsf{ProbGen}(\vec{x}, \mathsf{PK}_M)$ On input of a column vector $\vec{x} = (x_1, x_2 ..., x_m)^\intercal$ in $\mathbb{F}_p^m$ and public key $\mathsf{PK}_M = (\mathsf{PK}_1, \mathsf{PK}_2, ..., \mathsf{PK}_m)$ associated with matrix M, algorithm $\mathsf{ProbGen}$ derives

$$\mathsf{VK}_x = \prod_{j=1}^m \mathsf{PK}_j^{x_j}$$

and returns the encoding $\sigma_x = \vec{x}$ and the verification key VK_x.

$\mathsf{Compute}(\sigma_x, \mathsf{EK}_M)$ Provided with encoding $\sigma_x = \vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and evaluation key $\mathsf{EK}_M = (M, \mathcal{N})$, algorithm $\mathsf{Compute}$ multiplies matrix M with vector $\vec{x}$ which yields a column vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$, evaluates the product:

$$\Pi = \prod_{i=1}^n \prod_{j=1}^m \mathcal{N}_{ij}^{x_j}$$

and outputs the encoding $\sigma_y = (\vec{y}, \Pi)$.

$\mathsf{Verify}(\sigma_y, \mathsf{VK}_x)$ Given $\sigma_y = (\vec{y}, \Pi)$ and verification key VK_x, algorithm Verify checks whether the following equality holds:

$$e(\Pi, h) \stackrel{?}{=} e(\prod_{i=1}^n g_i^{y_i}, \tilde{h})\mathsf{VK}_x. \tag{4}$$

If so, algorithm Verify outputs $\vec{y}$ meaning that $M\vec{x} = \vec{y}$; otherwise it outputs $\perp$.

4.3 Security Analysis

In this section, we formally prove the security properties of our solution for publicly verifiable matrix multiplication.

Theorem 3. *The solution described above for publicly verifiable matrix multiplication is correct.*

Proof. If when queried with vector $\vec{x} = (x_1, x_2, ..., x_n)^\intercal$, the server correctly operates algorithm $\mathsf{Compute}$, then Equation 4 always holds.

Actually in that case, σ_y corresponds to the pair $(\vec{y}, \Pi)$ such that $\vec{y} = (y_1, y_2, ..., y_n)^\intercal = M\vec{x}$ and $\Pi = \prod_{i=1}^n \prod_{j=1}^m \mathcal{N}_{ij}^{x_j}$. This implies that for all $1 \leqslant i \leqslant n$: $y_i = \sum_{j=1}^m M_{ij}x_j \mod p$, and as the order of g_i is p, it also implies that:

$$\begin{aligned}
\Pi &= \prod_{i=1}^n \prod_{j=1}^m \mathcal{N}_{ij}^{x_j} = \prod_{i=1}^n \prod_{j=1}^m \left(g_i^{\delta M_{ij} + R_{ij}}\right)^{x_j} \\
&= \prod_{i=1}^n \prod_{j=1}^m \left(g_i^{\delta M_{ij}x_j + R_{ij}x_j}\right) = \prod_{i=1}^n \prod_{j=1}^m g_i^{\delta M_{ij}x_j} g_i^{R_{ij}x_j} \\
&= \prod_{i=1}^n g_i^{\delta \sum_{j=1}^m M_{ij}x_j} \prod_{i=1}^n \prod_{j=1}^m g_i^{R_{ij}x_j} \\
&= \prod_{i=1}^n g_i^{\delta y_i} \prod_{i=1}^n \prod_{j=1}^m g_i^{R_{ij}x_j}
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}
e(\Pi, h) &= e(\prod_{i=1}^n g_i^{\delta y_i} \prod_{i=1}^n \prod_{j=1}^m g_i^{R_{ij}x_j}, h) \\
&= e(\prod_{i=1}^n g_i^{y_i}, h^\delta) e(\prod_{i=1}^n \prod_{j=1}^m g_i^{R_{ij}x_j}, h) \\
&= e(\prod_{i=1}^n g_i^{y_i}, h^\delta) \prod_{j=1}^m e(\prod_{i=1}^n g_i^{R_{ij}}, h)^{x_j}
\end{aligned}$$

[2]Without loss of generality, we can assume that $g_i = g^{\lambda_i}$ for random λ_i in $\mathbb{F}_p^*$.

As $\tilde{h} = h^{\delta}$ and $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$, where

$$\mathsf{PK}_j = e(\prod_{i=1}^{n} g_i^{R_{ij}}, h)$$

we get:

$$e(\Pi, h) = e(\prod_{i=1}^{n} g_i^{y_i}, \tilde{h})\mathsf{VK}_x$$

and we conclude that Verify outputs $\vec{y} = M\vec{x}$. $\square$

Theorem 4. *The solution described above for publicly verifiable matrix multiplication is sound under the co-CDH assumption in* $\mathbb{G}_1$.

Proof. Assume there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable delegation of matrix multiplication with a non-negligible advantage ϵ. We show in what follows how an adversary $\mathcal{B}$ can use adversary $\mathcal{A}$ to break the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \simeq \epsilon$.

To break the co-CDH assumption, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{co-cdh}}$ which in turn outputs the pair $(g, g^{\alpha}) \in \mathbb{G}_1^2$ and the pair $(h, h^{\beta}) \in \mathbb{G}_2^2$.

Later, adversary $\mathcal{B}$ simulates the soundness experiment (cf. Algorithm 1) to adversary $\mathcal{A}$ as following:

When adversary $\mathcal{A}$ calls the oracle $\mathcal{O}_{\mathsf{Setup}}$ with some matrix M of elements M_{ij} in $\mathbb{F}_p$, adversary $\mathcal{B}$ simulates the oracle $\mathcal{O}_{\mathsf{Setup}}$ of the soundness experiment by executing algorithm Setup as depicted in Section 4.2 except for the following:

1. It lets $\widehat{g} = g^{\alpha}$ and $\widehat{h} = (h^{\beta})^{\delta}$, computes for all $1 \leqslant i \leqslant n$, $\widehat{g}_i = \widehat{g}^{\lambda_i}$ for some randomly chosen $\lambda_i \in \mathbb{F}_p^*$, and sets the public parameters to
$$\widehat{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\widehat{g}_i\}_{1 \leqslant i \leqslant n}, h, \widehat{h}).$$
2. It generates an (n, m)-random matrix $\widehat{\mathcal{N}}$ of elements $\widehat{\mathcal{N}}_{ij} \in \mathbb{G}_1$;
3. It computes for all $1 \leqslant j \leqslant m$,
$$\widehat{\mathsf{PK}}_j = \frac{e(\prod_{i=1}^{n} \widehat{\mathcal{N}}_{ij}, h)}{e(\prod_{i=1}^{n} \widehat{g}_i^{M_{ij}}, \widehat{h})}; \tag{5}$$
4. It defines the public key associated with matrix M as $\widehat{\mathsf{PK}}_M = (\widehat{\mathsf{PK}}_1, ..., \widehat{\mathsf{PK}}_m)$;
5. Finally, it sets the corresponding evaluation key to $\widehat{\mathsf{EK}}_M = (M, \widehat{\mathcal{N}})$.

Adversary $\mathcal{B}$ concludes its simulation of the oracle $\mathcal{O}_{\mathsf{Setup}}$ by outputting public parameters $\widehat{\mathsf{param}}$, public key $\widehat{\mathsf{PK}}_M$ and evaluation key $\widehat{\mathsf{EK}}_M$.

Note here that the simulated output of oracle $\mathcal{O}_{\mathsf{Setup}}$ in the game is statistically indistinguishable from the distribution of the output of algorithm Setup in the soundness experiment. Namely, the following is true:

- The statistical distribution of matrix $\widehat{\mathcal{N}}$ is identical to the distribution of matrix $\mathcal{N}$ generated by algorithm Setup.
- For all vectors $\vec{x} = (x_1, ..., x_m)^{\intercal} \in \mathbb{F}_p^m$ and $\vec{y} = (y_1, ..., y_n)^{\intercal} = M\vec{x}$, the simulated public key $\widehat{\mathsf{PK}}_M = (\widehat{\mathsf{PK}}_1, ..., \widehat{\mathsf{PK}}_m)$ verifies this equation:
$$e(\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}, h) = e(\prod_{i=1}^{n} \widehat{g}_i^{y_i}, \widehat{h}) \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{x_j}.$$

Therefore, we conclude that the distribution of matrix $\widehat{\mathcal{N}}$ and public key $\widehat{\mathsf{PK}}_M$ is the same as the distribution of matrix $\mathcal{N}$ and $\mathsf{PK}_M = (\mathsf{PK}_1, ..., \mathsf{PK}_m)$ produced by algorithm Setup.

At the end of the experiment, adversary $\mathcal{A}$ picks a challenge vector $\vec{x} = (x_1, x_2, ..., x_m)^{\intercal}$ and queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with the pair $(\vec{x}, \widehat{\mathsf{PK}}_M)$.

As a result, adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ and outputs the pair $(\vec{x}, \widehat{\mathsf{VK}}_x)$ with $\widehat{\mathsf{VK}}_x = \prod_{j=1}^{m} \widehat{\mathsf{PK}}_j^{x_j}$.

Afterwards, adversary $\mathcal{A}$ returns a response $\sigma_y = (\vec{y}, \Pi)$ such that $\vec{y} \neq M\vec{x}$.

In the remainder of this proof, we denote $\vec{y}^* = (y_1^*, y_2^*, ..., y_n^*)^{\intercal} = M\vec{x}$.

To break the co-CDH assumption in $\mathbb{G}_1$, adversary $\mathcal{B}$ first fetches the vector $\vec{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_n)$ used to compute the powers $\widehat{g}_i = \widehat{g}^{\lambda_i}$ and verifies whether $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$. If so, adversary $\mathcal{B}$ aborts the game; otherwise it breaks co-CDH by returning:

$$g^{\alpha\beta} = \left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}\right)^{(\delta\vec{\lambda}(\vec{y}-\vec{y}^*))^{-1}}.$$

Indeed, if $\sigma_y = (\vec{y}, \Pi)$ passes the verification, then this implies that the following equation holds:

$$e(\Pi, h) = e(\prod_{i=1}^{n} \widehat{g}_i^{y_i}, \widehat{h})\widehat{\mathsf{VK}}_x. \tag{6}$$

Also given Equation 5, we have:

$$e(\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}, h) = e(\prod_{i=1}^{n} \widehat{g}_i^{y_i^*}, \widehat{h})\widehat{\mathsf{VK}}_x \tag{7}$$

By dividing Equation 6 with Equation 7, we obtain:

$$\begin{aligned} e\left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(\prod_{i=1}^{n} \widehat{g}_i^{y_i - y_i^*}, \widehat{h}\right) \\ &= e\left(\prod_{i=1}^{n} \widehat{g}^{\lambda_i(y_i - y_i^*)}, \widehat{h}\right) \\ &= e\left(\widehat{g}^{\sum_{i=1}^{n} \lambda_i(y_i - y_i^*)}, \widehat{h}\right) \\ &= e\left(\widehat{g}^{\vec{\lambda}(\vec{y}-\vec{y}^*)}, \widehat{h}\right) \end{aligned}$$

As $\widehat{g} = g^{\alpha}$ and $\widehat{h} = h^{\beta\delta}$, we deduce that

$$\begin{aligned} e\left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}, h\right) &= e\left(g^{\alpha\vec{\lambda}(\vec{y}-\vec{y}^*)}, h^{\beta\delta}\right) \\ &= e\left(g^{\alpha\beta}, h\right)^{\delta\vec{\lambda}(\vec{y}-\vec{y}^*)} \end{aligned}$$

Therefore if $\vec{\lambda}(\vec{y}-\vec{y}^*) \neq 0 \mod p$, then $\delta\vec{\lambda}(\vec{y}-\vec{y}^*) \neq 0 \mod p$ ($\delta \in \mathbb{F}_p^*$) and we can compute:

$$g^{\alpha\beta} = \left(\frac{\Pi}{\prod_{i=1}^{n}\prod_{j=1}^{m} \widehat{\mathcal{N}}_{ij}^{x_j}}\right)^{(\delta\vec{\lambda}(\vec{y}-\vec{y}^*))^{-1}}$$

Hence, adversary $\mathcal{B}$ breaks the co-CDH assumption in $\mathbb{G}_1$ as long

Algorithm	Computation cost	Client's storage	Server's storage
Setup	nm prng in $\mathbb{F}_p$ and nm mul in $\mathbb{F}_p$ $m(n-1)$ mul and $2nm$ exp in $\mathbb{G}_1$ m pairings	$\mathcal{O}(n+m)$	$\mathcal{O}(nm)$
ProbGen	$(m-1)$ mul and m exp in $\mathbb{G}_T$	–	–
Compute	nm mul in $\mathbb{F}_p$ $(n-1)(m-1)$ mul and nm exp in $\mathbb{G}_1$	–	–
Verify	$(n-1)$ mul and n exp in $\mathbb{G}_1$ 1 mul in $\mathbb{G}_T$ 2 pairings	–	–

Table 2: Computation and storage requirements of our protocol for publicly verifiable matrix multiplication

as $\vec{\lambda}\vec{y} \neq \vec{\lambda}\vec{y}^* \mod p$. Fortunately, under the hardness of discrete logarithm, the probability that adversary $\mathcal{B}$ finds $\vec{y}$ such that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$ is negligible.

Lemma 1. *If adversary $\mathcal{A}$ outputs $\vec{y}$ such that $\vec{\lambda}\vec{y} = \vec{\lambda}\vec{y}^* \mod p$, then adversary $\mathcal{B}$ can break the discrete logarithm (DL) assumption in $\mathbb{G}_1$.*

Proof Sketch. Assume there is an adversary $\mathcal{A}$ that outputs a vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ verifying the property above with a non-negligible advantage ϵ. Here we show that there is another adversary $\mathcal{B}$ which uses adversary $\mathcal{A}$ to break the discrete logarithm assumption in $\mathbb{G}_1$ with a non-negligible advantage $\geqslant \epsilon/n$.

Assume that adversary $\mathcal{B}$ receives $\breve{g} \in \mathbb{G}_1$ and is required to output $\lambda \in \mathbb{F}_p$ such that $\breve{g} = g^\lambda$.

To this effect, adversary $\mathcal{B}$ simulates the soundness experiment as depicted in Algorithm 1. More precisely, upon receipt of an (n, m)-matrix M, it simulates the output of $\mathcal{O}_{\mathsf{Setup}}$ exactly as the soundness experiment except for the following:

- It selects k randomly in $\{1, 2, ..., n\}$ and lets $\breve{g}_k = \breve{g}$;
- for all $1 \leqslant i \leqslant n$, $i \neq k$, it randomly selects $\lambda_i \in \mathbb{F}_p^*$ and sets $\breve{g}_i = \breve{g}^{\lambda_i}$;
- it sets the public parameters to $\widetilde{\mathsf{param}} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \{\breve{g}_i\}_{1\leqslant i\leqslant n}, h, \tilde{h})$.

Adversary $\mathcal{A}$ eventually returns a pair of vectors $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ that verify $\vec{y} \neq M\vec{x}$ and $\vec{\lambda}\vec{y} = \vec{\lambda}M\vec{x} \mod p$, whereby $\vec{\lambda} = (\lambda_1, ..., \lambda_{k-1}, \lambda, \lambda_{k+1}, ..., \lambda_n)$.

If we denote $\vec{y}^* = (y_1^*, y_2^*, ..., y_n^*)^\intercal = M\vec{x}$, then the above equality entails that

$$\lambda = \frac{\sum_{i=1, i\neq k}^{n} \lambda_i (y_i^* - y_i)}{y_k - y_k^*}$$

as long as $y_k \neq y_k^*$.

Since $\vec{y} \neq \vec{y}^*$, then there is at least one index $1 \leqslant j \leqslant n$ such that $y_j \neq y_j^*$. Since k is randomly chosen from $\{1, ..., n\}$, the probability that $y_k \neq y_k^*$ is at least $1/n$, and consequently, adversary $\mathcal{B}$ will be able to break the discrete logarithm assumption with advantage $\geqslant \epsilon/n$.

□

To summarize, if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable matrix multiplication with a non-negligible advantage ϵ, then there exists an adversary $\mathcal{B}$ that breaks the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \simeq \epsilon$. □

4.4 Performance Analysis

Algorithm Setup generates the (n, m)-random matrix R which requires the generation of nm random numbers in $\mathbb{F}_p$. To compute the elements $\mathcal{N}_{ij}$ of matrix $\mathcal{N}$ as $g_i^{\delta M_{ij} + R_{ij}}$, algorithm Setup performs nm multiplications and nm additions in $\mathbb{F}_p$, and nm exponentiations in $\mathbb{G}_1$. Furthermore, the generation of public key PK_M demands $m(n-1)$ multiplications in $\mathbb{G}_1$, nm exponentiations in $\mathbb{G}_1$ and m pairings. It should be noted that while algorithm Setup involves expensive operations such as exponentiations and pairings, it is executed only once by the client, and consequently, its cost is *amortized* over the large number of verifications that a verifier can perform.

To multiply a vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ with matrix M, algorithm ProbGen computes $\mathsf{VK}_x = \prod_{j=1}^{m} \mathsf{PK}_j^{x_j}$. This involves $m-1$ multiplications and m exponentiations in $\mathbb{G}_T$.

Moreover, algorithm Compute consists of two operations: (i) the matrix multiplication $\vec{y} = M\vec{x}$ which requires nm multiplications and additions in $\mathbb{F}_p$; and (ii) the generation of the proof Π which involves nm exponentiations and $(n-1)(m-1)$ multiplications in $\mathbb{G}_1$.

Finally, algorithm Verify evaluates two bilinear pairings, $(n-1)$ multiplications and n exponentiations in $\mathbb{G}_1$, and one multiplication in $\mathbb{G}_T$.

As for storage, the server is required to keep the (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ and the (n, m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} \in \mathbb{G}_1$. On the other hand, the client is required to store and publish the public parameters which are of size $\mathcal{O}(n)$ and the public key PK_M whose size is $\mathcal{O}(m)$. We highlight the fact that the public parameters' size can be made constant: Instead of advertising the set $\{g_i\}_{1\leqslant i\leqslant n}$, the client can select a hash function $\mathcal{H} : \mathbb{F}_p^* \rightarrow \mathbb{G}_1\backslash\{1\}$ and compute the generators g_i as $\mathcal{H}(i)$, for all $1 \leqslant i \leqslant n$. On the downside, this optimization makes our scheme secure only in the random oracle model.

Table 2 summarizes the performance analysis of our scheme for publicly verifiable matrix multiplication.

5. RELATED WORK

Verifiable Polynomial Evaluation. Benabbas et al. [6] were the first to use algebraic PRFs for the problem of verifiable polynomial evaluation. Their solution only works in the symmetric-key setting, thus does not enable public verifiability as the schemes presented in this paper. In the same line of work, Fiore and Gennaro [11] devise new algebraic PRFs, also used by Zhang and Safavi-Naini [20], to develop publicly verifiable solutions. Compared to these two solutions, our protocol induces the same amount of computational costs but with the additional property of public delegatability. Another solution for public verification considers signatures for

	Setup	ProbGen	Compute	Verify	Hardness Assumptions	Public Delegatability
Fiore and Gennaro [11]	1 pairing $2(d+1)$ exp in $\mathbb{G}_1$ 1 exp in $\mathbb{G}_T$	1 pairing 1 exp in $\mathbb{G}_1$	$(d+1)$ exp in $\mathbb{G}_1$	1 pairing 1 exp in $\mathbb{G}_T$	co-CDH DLin	No
Papamanthou et al. [17]	*Polynomial preparation* $2d+1$ exp in $\mathbb{G}_1$		$d+1$ exp in $\mathbb{G}_1$	2 pairing 2 exp in $\mathbb{G}_1$	d-SBDH	Yes
Our scheme	$d+1$ exp in $\mathbb{G}_2$ 1 exp in $\mathbb{G}_1$	1 exp in $\mathbb{G}_1$ 1 exp in $\mathbb{G}_2$	$d-1$ exp in $\mathbb{G}_2$	2 pairings 1 exp in $\mathbb{G}_2$	$\lfloor d/2 \rfloor$-SDH	Yes

Table 3: Comparison of computation complexity with existing work for polynomial evaluation

	Setup	ProbGen	Compute	Verify	Hardness Assumptions	Public Delegatability
Fiore and Gennaro [11]	$3nm$ exp in $\mathbb{G}_1$	n pairings $2(n+m)$ exp in $\mathbb{G}_1$	nm exp in $\mathbb{G}_1$	n pairings n exp in $\mathbb{G}_T$	co-CDH DLin	No
Zhang and Blanton [21]	1 pairing	n exp in $\mathbb{G}_1$ m exp in $\mathbb{G}_2$ $(n+1)$ exp in $\mathbb{G}_T$	nm exp in $\mathbb{G}_2$	n pairings $(n+1)$ exp in $\mathbb{G}_T$	M-DDH XDH	Yes
Our scheme	$2nm$ exp in $\mathbb{G}_1$ m pairings	m exp in $\mathbb{G}_T$	nm exp in $\mathbb{G}_1$	2 pairings n exp in $\mathbb{G}_1$	co-CDH	Yes

Table 4: Comparison of computation complexity with existing work for matrix multiplication

correct computation [17], and uses polynomial commitments [16] to construct these signatures. Besides public verifiability, this solution implements public delegatability. However, the construction by [17] relies on the d-SBDH assumption, whereas our solution is secure under a weaker assumption that is the $\lfloor d/2 \rfloor$-SDH. It is worth mentioning that our protocol can be changed to rely on the $\lfloor d/\delta \rfloor$-SDH assumption, where δ is the degree of the divisor polynomial, as specified in Remark 1 of Section 3.3, and therefore our scheme can accommodate higher-degree polynomials.

Table[3] 3 compares the computational costs our solution for polynomial evaluation with the work described by Fiore and Gennaro [11] and Papamanthou et al. [17].

Verifiable Matrix Multiplication.

Fiore and Gennaro [11] and Zhang and Safavi-Naini [20] exploit algebraic PRFs for publicly verifiable matrix multiplications. However, only the client which outsourced the matrix can submit input vectors to the outsourced multiplication, hence their constructions do not meet the public delegatability requirement. Zhang and Blanton [21] present a construction for publicly delegatable and verifiable outsourcing of matrix multiplication that uses mathematical properties of matrices instead of algebraic PRFs. Unlike our work, the public verifiable scheme suggested in [21] does not transfer the matrix M to the server during Setup (whose purpose is reduced to generating the public parameters). Instead, the problem generation phase prepares the matrix and the input vector for the delegation. This construction is secure under the multiple decisional Diffie Hellman (M-DDH) and the eXternal Diffie-Hellman (XDH) assumptions, which are stronger than the co-CDH assumption we rely on in our solution.

Table[3] 4 depicts a comparison of our proposal for matrix multiplication with the solution proposed by Fiore and Gennaro [11] and Zhang and Blanton [21].

Arbitrary functions. A significant collection of work applies succinct non-interactive arguments of knowledge (SNARKs) to the problem of verifiable computation of arbitrary functions [4, 5, 7, 19]. One of the most relevant applications of the SNARK approach [7] appears in Pinocchio [19]. Pinocchio translates the outsourced function into an arithmetic circuit, which is then converted into a Quadratic Arithmetic Program [14]. As such, it enables public delegatability and public verifiability. However, the security of Pinocchio and other SNARK-based protocols relies on non-falsifiable assumptions as proved by Gentry and Wichs [15], whereas the security of our schemes only relies on falsifiable assumptions.

Parno et al. [18] propose a solution for public delegation and verification of computation using Attribute-Based Encryption (ABE). However, this scheme is limited to the computation of Boolean functions that output a single bit. For functions with more than one output bit, the client has to repeatedly (for each output bit) launch several instances of the protocol. Alderman et al. [1, 2] also propose an ABE-based protocol for Boolean functions. The authors adopt a scenario orthogonal to ours, in which queriers are identified according to access control policies. Furthermore, Alderman et al. [1, 2] introduce the concept of *blind verifiability*. In a nutshell, their protocols distinguish queriers from verifiers: The latter may only be authorized to verify an outsourced computation but not to learn its results. In the present paper, the blind verifiability property is out of scope.

Homomorphic MACs and signatures. Another type of solutions use homomorphic MACs [3, 9, 12] or homomorphic signatures [8, 10]. These solutions generally induce a verification as costly as the computation of the outsourced function itself. Homomorphic MACs proposed by Backes et al. [3] take advantage of algebraic PRFs to allow efficient verification, provided that the data is indexed. This solution however is suitable for quadratic functions only. Similarly, Catalano et al. [10] propose homomorphic signatures for polynomial functions with efficient verification and suggest that they can be used for a publicly verifiable computation scheme. Nevertheless, their construction uses expensive multilinear pairings.

6. CONCLUSION

In this paper, we introduced two protocols for publicly verifi-

[3]Table 3 and Table 4 compare the computational complexity of our solution with existing work only in terms of exponentiations and bilinear pairings which are the most computationally expensive operations.

able delegation of computation which enable a client to securely outsource the evaluation of arbitrary degree univariate polynomials and the multiplication of large matrices. We built our protocols upon the *algebraic* properties of polynomials and matrices. This paved the way for practical solutions that are provably secure against adaptive adversaries under the co-CDH and the SDH assumptions.

7. ACKNOWLEDGEMENTS

This work was partially supported by the TREDISEC project (G.A. no 644412), funded by the European Union (EU) under the Information and Communication Technologies (ICT) theme of the Horizon 2020 (H2020) research and innovation programme.

References

[1] James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. Revocation in publicly verifiable outsourced computation. In *Information Security and Cryptology*, pages 51–71. Springer, 2014.

[2] James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. Access control in publicly verifiable outsourced computation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS*, volume 15, pages 657–662, 2015.

[3] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 863–874. ACM, 2013.

[4] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[5] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a Von Neumann architecture. In *USENIX Security*, pages 781–796, 2014.

[6] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer Berlin Heidelberg, 2011.

[7] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[8] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Advances in Cryptology–EUROCRYPT 2011*, pages 149–168. Springer, 2011.

[9] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *EUROCRYPT*, pages 336–352. Springer, 2013.

[10] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology–CRYPTO 2014*, pages 371–389. Springer, 2014.

[11] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 501–512. ACM, 2012.

[12] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *Advances in Cryptology-ASIACRYPT 2013*, pages 301–320. Springer, 2013.

[13] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*, pages 465–482. Springer, 2010.

[14] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, volume 7881, pages 626–645. Springer, 2013.

[15] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM, 2011.

[16] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010*, pages 177–194. Springer, 2010.

[17] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography*, pages 222–242. Springer, 2013.

[18] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer Berlin Heidelberg, 2012.

[19] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy (SP), 2013*, pages 238–252. IEEE, 2013.

[20] Liang Feng Zhang and Reihaneh Safavi-Naini. Verifiable delegation of computations with storage-verification trade-off. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, volume 8712 of *Lecture Notes in Computer Science*, pages 112–129. Springer International Publishing, 2014.

[21] Yihua Zhang and Marina Blanton. Efficient secure and verifiable outsourcing of matrix multiplications. Cryptology ePrint Archive, Report 2014/133, 2014.