EURECOM
Department of Networking and Security
Campus SophiaTech
CS 50193
06904 Sophia Antipolis cedex
FRANCE

Research Report RR-14-295

Efficient Techniques for Publicly Verifiable Delegation of Computation

April 03$^{\text{th}}$, 2015

Kaoutar Elkhiyaoui, Melek Önen, Monir Azraoui and Refik Molva

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : {kaoutar.elkhiyaoui,melek.onen, monir.azraoui, refik.molva}@eurecom.fr

---

[1]EURECOM's research is partially supported by its industrial members: BMW Group Research and Technology, IABG, Monaco Telecom, Orange, Principauté de Monaco, SAP, SFR, ST Microelectronics, Symantec.

Efficient Techniques for Publicly Verifiable Delegation of Computation

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. However, instead of using algebraic pseudo-random functions, we exploit the mathematical properties of polynomials and matrices to propose more efficient and more viable solutions. Besides their efficiency, our protocols are provably secure under standard assumptions.

Index Terms

Cloud computing, Verifiability

Index Terms

Verifiable Computation, Cloud Computing, Delegation

Contents

1 Introduction **1**

2 Publicly Verifiable Computation **2**
2.1 Correctness . . . 3
2.2 Soundness . . . 3

3 Publicly Verifiable Polynomial Evaluation **5**
3.1 Protocol Overview . . . 5
3.2 Bilinear Pairings . . . 6
3.3 Protocol Description . . . 6
3.4 Security Analysis . . . 7
3.5 Performance Analysis . . . 8

4 Publicly Verifiable Matrix Multiplication **9**
4.1 Protocol Overview . . . 9
4.2 Base Protocol . . . 10
4.3 Optimized Protocol for Verifiable Matrix Multiplication . . . 12
4.4 Security Analysis . . . 13
4.5 Performance Analysis . . . 14

5 Related Work **15**

6 Conclusion **17**

A Proof of Theorem 2 **20**

B Proof of Theorem 4 **24**

C Proof of Lemma 1 **28**

D Proof of Lemma 2 **31**

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 [1–5] 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 proof* 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 prequisite 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 [1, 2] keep this verification key secret, in which case only the client verifies the correctness of the outsourced computation, other proposals [3–6] allow *public verifiability* which empowers any third party verifier to assess the validity of the outsourced computation.

In this paper, we focus on the public verifiability of two specific functions, namely, high degree polynomial evaluation and matrix multiplication. Similarly to [3, 7], we adopt the *amortized model* [1]: In this model, the client is required to execute a one-time expensive pre-processing operation that is leveraged later for efficient verifications. While authors in [3] and [7] use *algebraic pseudo-random functions* (*algebraic PRF*) which thanks to their *closed-form efficiency* give way to an efficient verification process, we instead take advantage of the mathematical properties of polynomials and matrices to propose two cryptographic solutions that compare favorably to existing work. Notably, we exploit the properties of Euclidean division for polynomial evaluation and the properties of dot product for matrix multiplication.

Contributions:

- We first propose a verifiable polynomial evaluation solution whose efficiency derives from the Euclidean division of the actual polynomial by some randomly generated small degree polynomial. The basic idea of our solution is that the client securely stores this small degree polynomial together with the remainder one and outsources the actual polynomial with the quotient polynomial. Thanks to the properties of Euclidean division, the pre-processing of data before outsourcing in our scheme outperforms existing solutions [3].

- Secondly, we propose a solution for verifiable matrix multiplication which similarly to [3], requires the client to construct an auxiliary matrix. However, to optimize the verification operation, our solution generates this matrix as the product of two secret vectors as opposed to using algebraic PRFs. In this manner, we ensure that the verification at the client amounts to computing a dot product and performing a constant number of exponentiations and bilinear pairings, which makes our solution significantly more performant than related work.

- Both of our solutions are publicly verifiable and are proved to be correct and sound. Their soundness is proved under standard assumptions, namely the *co-computational Diffie-Hellman (co-CDH)* and *external Diffie-Hellman (XDH)* assumptions.

The rest of the paper is organized as follows. Section II formally defines publicly verifiable computation and the underlying security model. The proposed 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 Publicly Verifiable Computation

A publicly verifiable computation scheme is defined by four polynomial-time algorithms that enable a client $\mathcal{C}$ to outsource the evaluation of a family of functions $\mathcal{F}$ to a *potentially malicious* server $\mathcal{S}$, while allowing a third party verifier $\mathcal{V}$ to assess the correctness of the results output by server $\mathcal{S}$ (cf. [6]).

- $\mathsf{Setup}(1^\kappa, \mathfrak{f}) \to (\mathsf{param}_\mathfrak{f}, \mathsf{SK}_\mathfrak{f}, \mathsf{EK}_\mathfrak{f})$: It is a randomized algorithm executed by client $\mathcal{C}$. It takes as input the security parameter κ and a description of the function $\mathfrak{f} \in \mathcal{F}$ to be outsourced, and outputs a set of *public parameters* $\mathsf{param}_\mathfrak{f}$ that will be used by subsequent algorithms, a *secret key* $\mathsf{SK}_\mathfrak{f}$ that will be stored at client $\mathcal{C}$ and an *evaluation key* $\mathsf{EK}_\mathfrak{f}$ that will be transferred to server $\mathcal{S}$.

- $\mathsf{ProbGen}(x, \mathsf{SK}_\mathfrak{f}) \to (\sigma_x, \mathsf{VK}_x)$: Given an input x in the domain $\mathcal{D}_\mathfrak{f}$ of function $\mathfrak{f}$ and the client's secret key $\mathsf{SK}_\mathfrak{f}$, this algorithm generates an *encoding* σ_x of x that will be transmitted to server $\mathcal{S}$, and a *public verification key* VK_x that will be used by verifier $\mathcal{V}$ afterwards to check the correctness of the result returned by server $\mathcal{S}$.

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

- $\mathsf{Verify}(\sigma_y, \mathsf{VK}_x) \to \mathsf{out}_y$: Verifier $\mathcal{V}$ operates this deterministic algorithm to check the correctness of the result σ_y supplied by server $\mathcal{S}$ 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 $\mathsf{out}_y = \perp$.

A publicly verifiable computation scheme should assure that if server $\mathcal{S}$ is honest (i.e. executes the algorithm $\mathsf{Compute}$ correctly), then algorithm Verify always accepts the results provided by $\mathcal{S}$ (i.e. algorithm Verify never outputs an error $\perp$). This matches the *correctness* property of publicly verifiable computation. Verifiable computation schemes should also guarantee that a malicious server $\mathcal{S}$ cannot make the algorithm Verify (and therewith verifier $\mathcal{V}$) accept a result that is not correctly computed. This corresponds to the *soundness* property of publicly verifiable computation.

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

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

2.2 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 $\mathcal{V}$ 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 verifier $\mathcal{V}$ 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. $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, \mathsf{EK}_\mathfrak{f})$).

To capture the adversarial capabilities of an adversary (i.e. malicious server) $\mathcal{A}$ against a publicly verifiable computation scheme for a family of functions $\mathcal{F}$, we define a *soundness game* in which adversary $\mathcal{A}$ has an *oracle access* to the outputs of algorithms Setup and $\mathsf{ProbGen}$ (cf. [3, 6]).

Algorithm 1: Learning phase of the soundness game of publicly verifiable computation

$(\mathsf{param}_{\mathfrak{f}}, \mathsf{EK}_{\mathfrak{f}}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^{\kappa}, \mathfrak{f})$;
for $i := 1$ **to** t **do**
  $\mathcal{A} \rightarrow x_i$;
  $(\sigma_{x_i}, \mathsf{VK}_{x_i}) \leftarrow \mathcal{O}_{\mathsf{ProbGen}}(x_i, \mathsf{EK}_{\mathfrak{f}})$;
end

Algorithm 2: Challenge phase of the soundness game of publicly verifiable computation

$\mathcal{A} \rightarrow x^*$;
$(\sigma_{x*}, \mathsf{VK}_{x*}) \leftarrow \mathcal{O}_{\mathsf{ProbGen}}(x^*, \mathsf{EK}_{\mathfrak{f}})$;
$\mathcal{A} \rightarrow \sigma_{y*}$;
$\mathsf{out}_{y*} \leftarrow \mathsf{Verify}(\sigma_{y*}, \mathsf{VK}_{x*})$;

This formally means that adversary $\mathcal{A}$ is allowed to query the following oracles during the soundness game:

- $\mathcal{O}_{\mathsf{Setup}}$: When queried with a security parameter κ and a description of a function $\mathfrak{f} \in \mathcal{F}$, this oracle first executes the algorithm Setup which outputs a set of public parameters $\mathsf{param}_{\mathfrak{f}}$, an evaluation key $\mathsf{EK}_{\mathfrak{f}}$ and a secret key $\mathsf{SK}_{\mathfrak{f}}$ that thereafter will be associated with function $\mathfrak{f}$; then returns the public parameters $\mathsf{param}_{\mathfrak{f}}$ and the evaluation key $\mathsf{EK}_{\mathfrak{f}}$.

- $\mathcal{O}_{\mathsf{ProbGen}}$: When invoked with evaluation key $\mathsf{EK}_{\mathfrak{f}}$ and input x in $\mathcal{D}_{\mathfrak{f}}$, this oracle calls the algorithm $\mathsf{ProbGen}$ with x and secret key $\mathsf{SK}_{\mathfrak{f}}$ matching the evaluation key $\mathsf{EK}_{\mathfrak{f}}$, and outputs the resulting public encoding σ_x and public verification key VK_x.

In the learning phase of the soundness game (cf. Algorithm 1), adversary $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with a security parameter κ and a function $\mathfrak{f} \in \mathcal{F}$ in order to get the public parameters $\mathsf{param}_{\mathfrak{f}}$ and the evaluation key $\mathsf{EK}_{\mathfrak{f}}$. Next, adversary $\mathcal{A}$ *adaptively* invokes oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with inputs x_i and receives as a result the matching pairs of public encoding σ_{x_i} and public verification key VK_{x_i}.

In the challenge phase (see Algorithm 2), adversary $\mathcal{A}$ outputs a challenge input $x^* \in \mathcal{D}_{\mathfrak{f}}$ and calls the oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with evaluation key $\mathsf{EK}_{\mathfrak{f}}$ and x^* so as to get the matching pair of encoding σ_{x*} and public verification key VK_{x*}. Thereafter, adversary $\mathcal{A}$ generates an encoding σ_{y*}. At the end of the challenge phase, algorithm Verify takes as input the pair $(\mathsf{VK}_{x*}, \sigma_{y*})$ and outputs a value out_{y*}.

We say that adversary $\mathcal{A}$ succeeds in the soundness game of publicly verifiable computation if $\mathsf{out}_{y*} \neq \perp$ and $\mathsf{out}_{y*} \neq \mathfrak{f}(x^*)$.

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

Definition 2. *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. Notably the fact that for any pair of polynomials A and $B \neq 0$ of degree d and 1 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 the *quotient* polynomial Q equals $d-1$, whereas the the *remainder* polynomial R is constant. Therefore, the idea underpinning our scheme is as follows: Client $\mathcal{C}$ first picks a *random polynomial* B of degree 1, then divides A by B to get the quotient polynomial Q of degree $d-1$ and the constant remainder polynomial R. Next client $\mathcal{C}$ outsources polynomial A together with quotient polynomial Q to server $\mathcal{S}$ while safeguarding polynomial B and remainder R. Consequently, whenever client $\mathcal{C}$ wants to evaluate polynomial A at point x, it transmits x and the corresponding public verification key VK_x (which is defined as a function of x and polynomials B and R) to server $\mathcal{S}$. Server $\mathcal{S}$ in turn computes $y = A(x)$ and generates the proof $\pi = Q(x)$. To verify the result (y, π) output by server $\mathcal{S}$, verifier $\mathcal{V}$ checks whether $y = \pi B(x) + R$ using verification key VK_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 server $\mathcal{S}$ on an input x, verifier $\mathcal{V}$ is required to perform constant number of computations as opposed to carrying out $\mathcal{O}(d)$ operations to evaluate polynomial A.

Clearly, the soundness of such a protocol relies on the secrecy of polynomials B and R. However since B is one-degree polynomial, the secrecy of these two polynomials can be easily compromised by disclosing the quotient polynomial Q. To remedy this shortcoming, client $\mathcal{C}$ encodes polynomial Q using an *additively homomorphic one-way* encoding. Namely, each coefficient q_i of polynomial Q is encoded as g^{q_i}. Thus, we allow server $\mathcal{S}$ to *obliviously* compute an *exponent encoding* $\pi = g^{Q(x)}$ of the evaluation of Q at point x while ensuring the confidentiality of polynomials B and R.

To allow verifier $\mathcal{V}$ to check the correctness of the results (y, π) returned by server $\mathcal{S}$, we employ bilinear pairings, and accordingly, we show that our solution is sound under the *co-computational Diffie-Hellman* (co-CDH) assumption.

Before describing our protocol in full details, we recall the definitions of bilinear pairings and co-CDH 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 (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.*

3.3 Protocol Description

Our protocol for publicly verifiable computation of polynomials comprises four phases:

Setup Without loss of generality, we assume that client $\mathcal{C}$ 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.

To this effect, client $\mathcal{C}$ calls algorithm Setup with polynomial A and prime p. Algorithm Setup selects two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and picks a generator g and a generator h of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Algorithm Setup then defines the set of public parameters:

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

Next Setup randomly selects a 1-degree polynomial B with coefficient $b_i,\ i \in \{0,1\}$ in $\mathbb{F}_p$ (i.e. $B(X) = b_1 X + b_0$) that does not divide A. Note that a random polynomial of degree 1 divides A with probability $\leqslant \frac{d}{p}$. It then performs the Euclidean division of polynomial A by polynomial B in $\mathbb{F}_p[X]$, which results in a quotient polynomial $Q(X) = \sum_{i=0}^{d-1} q_i X^i$ and a constant remainder polynomial $R \neq 0$ (i.e. $A = QB + R$).

Thereupon, Setup sets the secret key SK_A associated with polynomial A to $\mathsf{SK}_A = (g, B, R)$.

To compute the evaluation key EK_A matching secret key SK_A, algorithm Setup encodes polynomial Q in the exponent. Namely, for each coefficient q_i of polynomial Q, algorithm Setup computes $\mathbf{q}_i = g^{q_i}$. Finally, algorithm Setup defines the evaluation key $\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-1})$.

At the end of this step, client $\mathcal{C}$ stores SK_A *securely*, transfers EK_A to server $\mathcal{S}$ and publishes param_A.

Problem Generation To evaluate polynomial A at point $x \in \mathbb{F}_p$, client $\mathcal{C}$ invokes algorithm $\mathsf{ProbGen}$ with secret key $\mathsf{SK}_A = (g, B, R)$ and x. As a result, algorithm $\mathsf{ProbGen}$ calculates $B(x) = b_1 x + b_0 \mod p$ and checks whether $B(x) = 0 \mod p$[1]. If it is the case, then algorithm $\mathsf{ProbGen}$ halts and returns $A(x) = R$. Otherwise, it computes $\mathsf{VK}_{(x,B)} = e(g,h)^{B(x)^{-1}}$ and $\mathsf{VK}_{(x,R)} = e(g,h)^{RB(x)^{-1}}$. Finally, $\mathsf{ProbGen}$ outputs the public encoding $\sigma_x = x$ and the public verification key $\mathsf{VK}_x = (\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)})$.

Computation Given $\sigma_x = x$ and the evaluation key $\mathsf{EK}_A = (A, \mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_{d-1})$, server $\mathcal{S}$ executes algorithm $\mathsf{Compute}$. Accordingly, algorithm $\mathsf{Compute}$ evaluates $y = A(x) = \sum_{i=0}^{d} a_i x^i \mod p$, generates the exponent encoding $\pi = \prod_{i=0}^{d-1} \mathbf{q}_i^{x^i}$ of $Q(x)$ and finally outputs the encoding $\sigma_y = (y, \pi)$.

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

$$\mathsf{VK}_{(x,B)}^{y} \stackrel{?}{=} e(\pi, h)\mathsf{VK}_{(x,R)} \tag{1}$$

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

3.4 Security Analysis

Here we state 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.*

Proof. If on a client $\mathcal{C}$'s query $\sigma_x = x \in \mathbb{F}_p$, server $\mathcal{S}$ follows the instructions of the **Computation** step correctly, then server $\mathcal{S}$'s response $\sigma_y = (y, \pi)$ is equal to $(A(x), g^{Q(x)})$. Indeed, we have:

$$\pi = \prod_{i=0}^{d-1} \mathbf{q}_i^{x^i} = \prod_{i=0}^{d-1} g^{q_i x^i} = g^{\sum_{i=0}^{d-1} q_i x^i} = g^{Q(x)}$$

[1]Polynomial B has one root in $\mathbb{F}_p$.

Protocol Phase	Computation	Client's storage	Server's storage
Setup	2 prng and d mul in $\mathbb{F}_p$ d exp in $\mathbb{G}_1$	$\mathcal{O}(1)$	$\mathcal{O}(d)$
Problem Generation	1 mul and 1 inv in $\mathbb{F}_p$ 2 exp in $\mathbb{G}_T$	–	–
Computation	$d + (d-2)$ mul in $\mathbb{F}_p$ d exp and $d-1$ mul in $\mathbb{G}_2$	–	–
Verification	1 exp and 1 mul in $\mathbb{G}_T$ 1 pairing	–	–

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

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

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

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

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

Thus since $B(x) \neq 0$ $e(g, h)^{yB(x)^{-1}} = e(\pi, h)e(g, h)^{RB(x)^{-1}}$.

By recalling that $(\mathsf{VK}_{(x,B)}, \mathsf{VK}_{(x,R)}) = (e(g, h)^{B(x)^{-1}}, e(g, h)^{RB(x)^{-1}})$, we conclude that $\mathsf{VK}^y_{(x,B)} = e(\pi, h)\mathsf{VK}_{(x,R)}$. □

Theorem 2. *The scheme proposed above for publicly verifiable polynomial evaluation is sound under the co-CDH assumption in* $\mathbb{G}_1$.

For ease of exposition, the proof of this theorem is deferred to Appendix A.

3.5 Performance Analysis

To outsource the evaluation of a polynomial A in $\mathbb{F}_p[X]$ to a cloud server $\mathcal{S}$, client $\mathcal{C}$ first generates two random coefficients $b_0, b_1 \in \mathbb{F}_p$ to construct polynomial B. Afterwards client $\mathcal{C}$ conducts an Euclidean division of polynomial A by polynomial B which consists of d multiplications and additions, where d is the degree of polynomial A. Once the Euclidean division is performed, client $\mathcal{C}$ computes $d-1$ exponentiations to encode the coefficients q_i of the quotient polynomial Q as $\mathbf{q}_i = g^{q_i} \in \mathbb{G}_1$. Although computationally expensive, the **Setup** phase is executed only once by client $\mathcal{C}$, and as a result, its computational cost is *amortized* over the large number of verifications that verifier $\mathcal{V}$ can carry out.

At the end of the **Setup** phase, client $\mathcal{C}$ stores the coefficients of polynomial B and remainder R, which corresponds to storing 3 coefficients in $\mathbb{F}_p$. Server $\mathcal{S}$ on the other hand, keeps the $d+1$ coefficients $a_i \in \mathbb{F}_p$ of polynomial A and the d encodings $\mathbf{q}_i \in \mathbb{G}_1$.

When client $\mathcal{C}$ wants to evaluate polynomial A at a point $x \in \mathbb{F}_p$, it 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, the **Problem Generation** phase consists of an evaluation of polynomial B at point x in $\mathbb{F}_p$, an inversion in $\mathbb{F}_p$, and 2 exponentiations in $\mathbb{G}_T$.

Upon receipt of a client $\mathcal{C}$'s query $\sigma_x = x$, server $\mathcal{S}$ enters the **Computation** phase which comprises two steps: **i.)** the evaluation of polynomial A at point x which requires at most d additions and multiplications in $\mathbb{F}_p$ if server $\mathcal{S}$ uses *Horner's rule*; and **ii.)** the generation of the proof π which involves $d-2$ multiplications in $\mathbb{F}_p$ and d exponentiations and $d-1$ multiplications in $\mathbb{G}_1$.

Finally, the **Verification** phase at verifier $\mathcal{V}$ only calls for 1 exponentiation and 1 multiplication in $\mathbb{G}_T$ and the computation of one bilinear pairing.

Table 1 depicts 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 [3] which states that in order to verify that a server $\mathcal{S}$ correctly multiplies an (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ (where p is a large prime) with some column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal \in \mathbb{F}_p^m$, it suffices that client $\mathcal{C}$ randomly picks a *secret* (n, m)-matrix R of elements R_{ij}, and supplies server $\mathcal{S}$ with (n, m)-matrix M and (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).

When client $\mathcal{C}$ prompts server $\mathcal{S}$ to multiply matrix M with vector $\vec{x}$, server $\mathcal{S}$ returns a pair of column vectors $\vec{y} = M\vec{x} = (y_1, y_2, ..., y_n)^\intercal$ and $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)^\intercal$, such that: $\pi_i = \tilde{g}^{y_i} g^{\sum_{j=1}^{m} R_{ij} x_j}$. If we denote $\vec{z} = (z_1, z_2, ..., z_n)^\intercal = R\vec{x}$, then the verification process would consist of checking whether π_i actually equals $\tilde{g}^{y_i} g^{z_i}$.

Now to transform this intuition into a viable solution, one must ensure that the matrix multiplication $R\vec{x}$ (and therewith the verification process) is much less computationally demanding than the matrix multiplication $M\vec{x}$ for all vectors $\vec{x}$. Along these lines, we suggest a base protocol (cf. Section 4.2) which similarly to the work of [3] generates the secret matrix R in a way that optimizes the multiplication $R\vec{x}$. However instead of using *algebraic PRF* as in [3], we construct the matrix R as the product of randomly generated column vector $\vec{a} = (a_1, a_2, ..., a_n)^\intercal$ and row vector $\vec{b} = (b_1, b_2, ..., b_m)$ (i.e. $R = \vec{a}\vec{b}$ and $R_{ij} = a_i b_j$). Therefore, the matrix multiplication $R\vec{x}$ requires $\mathcal{O}(n+m)$ operations rather than $\mathcal{O}(nm)$ – which is comparable to [3]. Indeed, if $R = \vec{a}\vec{b}$ and $\vec{z} = R\vec{x}$, then $z_i = \sum_{j=1}^{m} R_{ij}x_j = a_i(\vec{b} \cdot \vec{x}^\intercal)$ (where $\vec{b} \cdot \vec{x}^\intercal = \sum_{j=1}^{m} b_j x_j$ is the dot product of $\vec{b}$ and $\vec{x}^\intercal$).

Although the cost of computing the vector $R\vec{x}$ is considerably diminished, the verification process still calls for $\mathcal{O}(n)$ exponentiations and bilinear pairings to check whether $\pi_i = \tilde{g}^{y_i} g^{z_i}$, which can be computationally prohibitive to lightweight verifiers. Hence, we propose an optimization of the base protocol (cf. Section 4.3) which – at the cost of a slightly more expensive Setup algorithm – empowers the verifier to assess the correctness of the results sent by the server by only performing $\mathcal{O}(n)$ multiplications and $\mathcal{O}(1)$ exponentiations and bilinear pairings. Besides ensuring an efficient verification process, our solution also gives way to an efficient problem generation that outperforms existing work [3]. Namely, the problem generation in our scheme consists of performing $\mathcal{O}(m)$ multiplications to compute the dot product $\vec{b} \cdot \vec{x}^\intercal$ and $\mathcal{O}(1)$ exponentiations, instead of $\mathcal{O}(n+m)$ exponentiations and $\mathcal{O}(n)$ bilinear pairings in the case of [3].

Finally, we point out that the soundness of our solution is based on co-computational Diffie-Hellman (*co-CDH*) assumption and external Diffie Hellman (*XDH*) assumption in bilinear groups.

Definition 5 (XDH 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.*

We say that the **external Diffie-Hellman assumption** *(XDH) holds, if the decisional Diffie-Hellman assumption (DDH) holds in $\mathbb{G}_1$.*

4.2 Base Protocol

Our solution for publicly verifiable matrix multiplication consists of the following four phases:

Setup Without loss of generality, we assume that client $\mathcal{C}$ wants to outsource the multiplication operations involving an (n, m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ ($1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$) where p is a large prime.

In this respect, client $\mathcal{C}$ invokes the algorithm Setup with matrix M and prime p.

Algorithm Setup selects two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, picks a generator g and a generator h of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and computes $\tilde{g} = g^\delta$ and $\tilde{h} = h^\delta$ such that δ is randomly selected from $\mathbb{F}_p^*$. Next, it defines the set of public parameters:

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

Algorithm Setup thereafter picks two random vectors $\vec{a} = (a_1, a_2, ..., a_n)^\intercal \in \mathbb{F}_p^n$ and $\vec{b} = (b_1, b_2, ..., b_m) \in \mathbb{F}_p^m$ and sets the secret key SK_M associated with matrix M to $\mathsf{SK}_M = (\delta, \vec{a}, \vec{b})$.

To generate the evaluation key EK_M corresponding to secret key SK_M, algorithm Setup first computes the (n, m)-matrix $R = \vec{a}\vec{b}$ (i.e. for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$: $R_{ij} = a_i b_j \mod p$) and generates another (n, m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} \in \mathbb{G}_1$ such that $\mathcal{N}_{ij} = \tilde{g}^{M_{ij}} g^{R_{ij}}$, $\forall\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m$, then defines the evaluation key EK_M associated with matrix M as $\mathsf{EK}_M = (M, \mathcal{N})$.

At the end of this phase, client $\mathcal{C}$ stores SK_M *securely*, publishes param_M and transfers EK_M to server $\mathcal{S}$.

Problem Generation To multiply a column vector $\vec{x} = (x_1, x_2 ..., x_m)^\intercal$ with matrix M, client $\mathcal{C}$ executes the algorithm $\mathsf{ProbGen}$. On input of secret key $\mathsf{SK}_M = (\delta, \vec{a}, \vec{b})$ and vector $\vec{x}$, algorithm $\mathsf{ProbGen}$ evaluates the dot product $\tau_x = \vec{b} \cdot \vec{x}^\intercal = \sum_{j=1}^{m} b_j x_j \mod p$, computes $\mathsf{VK}_{i,x} = e(g, h)^{a_i \tau_x}$ (for all $1 \leqslant i \leqslant n$), and finally returns the encoding $\sigma_x = \vec{x}$ and the verification key $\mathsf{VK}_x = (\mathsf{VK}_{1,x}, \mathsf{VK}_{2,x}, ...,$ $\mathsf{VK}_{n,x})$

Computation Provided with encoding $\sigma_x = \vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and evaluation key $\mathsf{EK}_M = (M, \mathcal{N})$, server $\mathcal{S}$ calls algorithm $\mathsf{Compute}$. Algorithm $\mathsf{Compute}$ multiplies matrix M with vector $\vec{x}$ which yields a column vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ (i.e. $y_i = \sum_{j=1}^{m} M_{ij} x_j \mod p$), computes a vector $\vec{\pi} = (\pi_1, \pi_2, ..., \pi_n)^\intercal$ such that: $\pi_i = \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}, \forall\ 1 \leqslant i \leqslant n$, and finally outputs the encoding $\sigma_y = (\vec{y}, \vec{\pi})$.

Verification Given $\sigma_y = (\vec{y}, \vec{\pi})$ and verification key $\mathsf{VK}_x = (\mathsf{VK}_{1,x}, \mathsf{VK}_{2,x}, ...,$ $\mathsf{VK}_{n,x})$, verifier $\mathcal{V}$ runs algorithm Verify which checks whether the following equality holds:

$$e(\pi_i, h) \stackrel{?}{=} e(g, \tilde{h})^{y_i} \mathsf{VK}_{i,x}\ \ ,\ \forall\ 1 \leqslant i \leqslant n \tag{2}$$

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

Notice here that if server $\mathcal{S}$ executes the computation phase correctly, then Equation 2 always holds. Notably, we have $\vec{y} = (y_1, y_2, ..., y_n)^\intercal = M\vec{x}$ which implies that for all $1 \leqslant i \leqslant n$: $y_i = \sum_{j=1}^{m} M_{ij} x_j \mod p$. Moreover, as the order of g and $\tilde{g}$ is p, we have for all $1 \leqslant i \leqslant n$:

$$\begin{aligned}
\pi_i = \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j} &= \prod_{j=1}^{m} \left(\tilde{g}^{M_{ij}} g^{R_{ij}}\right)^{x_j} \\
&= \tilde{g}^{\sum_{j=1}^{m} M_{ij} x_j} g^{\sum_{j=1}^{m} R_{ij} x_j} = \tilde{g}^{y_i} g^{\sum_{j=1}^{m} a_i b_j x_j} \\
&= \tilde{g}^{y_i} g^{a_i \sum_{j=1}^{m} b_j x_j} = \tilde{g}^{y_i} g^{a_i (\vec{b} \cdot \vec{x}^\intercal)} = \tilde{g}^{y_i} g^{a_i \tau_x}
\end{aligned}$$

As $\tilde{g} = g^\delta$ and $\tilde{h} = h^\delta$, we obtain:

$$\begin{aligned}
e(\pi_i, h) &= e(\tilde{g}^{y_i} g^{a_i \tau_x}, h) = e(g^{\delta y_i}, h) e(g^{a_i \tau_x}, h) \\
&= e(g, h^\delta)^{y_i} e(g, h)^{a_i \tau_x} = e(g, \tilde{h})^{y_i} \mathsf{VK}_{i,x}
\end{aligned}$$

Although correct, the above protocol is computationally demanding for clients and verifiers. Clients have to perform $\mathcal{O}(n)$ exponentiations in $\mathbb{G}_T$ to compute verification keys, whereas verifiers need to carry out $\mathcal{O}(n)$ exponentiations in $\mathbb{G}_T$ and $\mathcal{O}(n)$ bilinear pairings to assess the correctness of the outsourced computation. To address this issue, we propose an optimization to this protocol which, at the price of a slightly more expensive **Setup** phase, allows clients to generate verification keys with a constant number of exponentiations, and enables verifiers to check the correctness of the delegated computation with a constant number of exponentiations and bilinear pairings.

4.3 Optimized Protocol for Verifiable Matrix Multiplication

Similarly to the base protocol, our optimized solution for verifiable matrix multiplication runs in four phases:

Setup Client $\mathcal{C}$ calls algorithm Setup with (n, m)-matrix M and prime number p.

Algorithm Setup as before chooses two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p that admit a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, selects a generator g and a generator h of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, computes $\tilde{g} = g^\delta$ and $\tilde{h} = h^\delta$ for a randomly selected δ in $\mathbb{F}_p^*$, picks a random vector $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n) \in \mathbb{F}_p^n$, and subsequently defines the public parameters associated with matrix M as:

$$\mathsf{param}_M = (p, \vec{\gamma}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, \tilde{h})$$

Afterwards, algorithm Setup randomly selects a vector $\vec{a} = (a_1, a_2, ..., a_n)^\intercal \in \mathbb{F}_p^n$ and evaluates the dot product: $\nu = \vec{\gamma} \cdot \vec{a}^\intercal = \sum_{i=1}^n \gamma_i a_i \mod p$. Then algorithm Setup selects another random vector $\vec{b} = (b_1, b_2, ..., b_m) \in \mathbb{F}_p^m$ and sets the secret key SK_M associated with matrix M to $\mathsf{SK}_M = (\delta, \nu, \vec{b})$.

To derive the evaluation key EK_M matching secret key SK_M, algorithm Setup computes for all $1 \leqslant i \leqslant n$ the pair $(g_i, \tilde{g}_i) = (g^{\gamma_i}, \tilde{g}^{\gamma_i}) = (g_i, g_i^\delta)$, then defines an (n, m)-matrix $\mathcal{N}$ as $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{a_i b_j}$ $(\forall\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m)$ and finally sets the evaluation key EK_M to $(M, \mathcal{N})$.

As in the base protocol, client $\mathcal{C}$ stores SK_M *securely*, publishes param_M and sends EK_M to server $\mathcal{S}$.

Problem Generation To multiply a column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ with matrix M, client $\mathcal{C}$ calls algorithm $\mathsf{ProbGen}$ which on input of secret key $\mathsf{SK}_M = (\delta, \nu, \vec{b})$ and $\vec{x}$ evaluates the dot product $\tau_x = \nu(\vec{b} \cdot \vec{x}^\intercal)$, sets the verification key to $\mathsf{VK}_x = e(g, h)^{\tau_x}$ and outputs the public encoding $\sigma_x = \vec{x}$ together with verification key VK_x.

Computation On inputs of encoding $\sigma_x = \vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and evaluation key $\mathsf{EK}_M = (M, \mathcal{N})$, server $\mathcal{S}$ invokes algorithm $\mathsf{Compute}$. Algorithm $\mathsf{Compute}$ in turn multiplies matrix M with vector $\vec{x}$ which results in the column vector $\vec{y} = (y_1, y_2, ..., y_n)^\intercal$ (i.e. $y_i = \sum_{i=1}^{m} M_{ij} x_j \mod p$), evaluates the product $\Pi = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j}$ and outputs the encoding $\sigma_y = (\vec{y}, \Pi)$.

Verification Given encoding $\sigma_y = (\vec{y}, \Pi)$ and verification key VK_x, verifier $\mathcal{V}$ executes algorithm Verify which upon invocation checks whether the following equation holds:

$$e(\Pi, h) \stackrel{?}{=} e(g, \tilde{h})^{\vec{\gamma} \cdot \vec{y}^\intercal} \mathsf{VK}_x \tag{3}$$

where $\vec{\gamma} \cdot \vec{y}^\intercal$ denotes the dot product of row vectors $\vec{\gamma}$ and $\vec{y}^\intercal$.

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

4.4 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 server $\mathcal{S}$ correctly performs the **Computation** phase when queried with vector $\vec{x} = (x_1, x_2, ..., x_n)^\intercal$, then Equation 3 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 and $\tilde{g}_i$ is p, it also implies that:

$$\begin{aligned}
\Pi &= \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}_{ij}^{x_j} = \prod_{i=1}^{n} \prod_{j=1}^{m} \left(\tilde{g}_i^{M_{ij}} g_i^{R_{ij}} \right)^{x_j} \\
&= \prod_{i=1}^{n} \prod_{j=1}^{m} \left(\tilde{g}_i^{M_{ij}} g_i^{a_i b_j} \right)^{x_j} = \prod_{i=1}^{n} \prod_{j=1}^{m} \tilde{g}_i^{M_{ij} x_j} g_i^{a_i b_j x_j} \\
&= \prod_{i=1}^{n} \tilde{g}_i^{\sum_{j=1}^{m} M_{ij} x_j} \prod_{i=1}^{n} g_i^{a_i \sum_{j=1}^{m} b_j x_j} = \prod_{i=1}^{n} \tilde{g}_i^{y_i} \prod_{i=1}^{n} g_i^{a_i (\vec{b} \cdot \vec{x}^\intercal)}
\end{aligned}$$

Since for all $1 \leqslant i \leqslant n$, $g_i = g^{\gamma_i}$ and $\tilde{g}_i = \tilde{g}^{\gamma_i} = g^{\delta \gamma_i}$, then:

$$\begin{aligned}
\Pi &= \prod_{i=1}^{n} g^{\delta \gamma_i y_i} \prod_{i=1}^{n} g^{\gamma_i a_i (\vec{b} \cdot \vec{x}^\intercal)} = g^{\delta \sum_{i=1}^{n} \gamma_i y_i} g^{(\vec{b} \cdot \vec{x}^\intercal) \sum_{i=1}^{n} \gamma_i a_i} \\
&= g^{\delta (\vec{\gamma} \cdot \vec{y}^\intercal)} g^{(\vec{b} \cdot \vec{x}^\intercal)(\vec{\gamma} \cdot \vec{a}^\intercal)} = g^{\delta (\vec{\gamma} \cdot \vec{y}^\intercal)} g^{(\vec{b} \cdot \vec{x}^\intercal) \nu} = g^{\delta (\vec{\gamma} \cdot \vec{y}^\intercal)} g^{\tau_x} \\
e(\Pi, h) &= e(g^{\delta (\vec{\gamma} \cdot \vec{y}^\intercal)} g^{\tau_x}, h) = e(g^{\delta (\vec{\gamma} \cdot \vec{y}^\intercal)}, h) e(g^{\tau_x}, h) \\
&= e(g, h^{\delta})^{\vec{\gamma} \cdot \vec{y}^\intercal} e(g, h)^{\tau_x}
\end{aligned}$$

Protocol Phase	Computation cost	Client's storage	Server's storage
Setup	$(2n+m)$ prng in $\mathbb{F}_p$ $n(3m+1)$ mul in $\mathbb{F}_p$ nm exp in $\mathbb{G}_1$	$\mathcal{O}(1)$	$\mathcal{O}(nm)$
Problem Generation	m prng and $(m+1)$ mul in $\mathbb{F}_p$ 1 exp in $\mathbb{G}_T$	–	–
Computation	nm mul in $\mathbb{F}_p$ $(n-1)(m-1)$ mul and nm exp in $\mathbb{G}_1$	–	–
Verification	n prng and n mul in $\mathbb{F}_p$ 1 exp and 1 mul in $\mathbb{G}_T$ 2 pairings	–	–

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

As $\tilde{h} = h^{\delta}$ and $\mathsf{VK}_x = e(g,h)^{\tau_x}$, we get: $e(\Pi, h) = e(g,\tilde{h})^{\vec{\gamma}\cdot\vec{y}^{\intercal}}\mathsf{VK}_x$ □

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

For ease of exposition, the proof of this theorem is deferred to Appendix B.

4.5 Performance Analysis

To delegate the operations involving an (n,m)-matrix M of elements M_{ij} in $\mathbb{F}_p$, client $\mathcal{C}$ first generates three random vectors $\vec{a} = (a_1, a_2, ..., a_n)^{\intercal} \in \mathbb{F}_p^n$, $\vec{b} = (b_1, b_2, ..., b_m) \in \mathbb{F}_p^m$ and $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n) \in \mathbb{F}_p^n$, which calls for the generation of $2n+m$ random numbers in $\mathbb{F}_p$. Then it evaluates the dot product $\nu = \vec{\gamma}\cdot\vec{a}^{\intercal}$ by performing n multiplications and $(n-1)$ additions in $\mathbb{F}_p$. Next client $\mathcal{C}$ computes the pair $(g_i, \tilde{g}_i) = (g^{\gamma_i}, g^{\delta\gamma_i})$ $(1 \leqslant i \leqslant n)$ and uses these pairs to calculate $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{a_i b_j}$ $(1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m)$. Notice that one can reduce the cost of the **Setup** phase by directly computing $\mathcal{N}_{ij}$ as $g^{\gamma_i(\delta M_{ij}+a_i b_j)} = \tilde{g}_i^{M_{ij}} g_i^{a_i b_j}$. This requires $3nm$ multiplications and nm additions in $\mathbb{F}_p$, and nm exponentiations in $\mathbb{G}_1$. It should be noted that while the **Setup** phase involves expensive operations such as exponentiations, it is executed only once by client $\mathcal{C}$, and consequently, its cost is *amortized* over the large number of verifications that verifier $\mathcal{V}$ can perform.

At the end of the **Setup** phase, server $\mathcal{S}$ stores the (n,m)-matrix M of elements $M_{ij} \in \mathbb{F}_p$ and the (n,m)-matrix $\mathcal{N}$ of elements $\mathcal{N}_{ij} \in \mathbb{G}_1$; whereas client $\mathcal{C}$ stores the secret key $\mathsf{SK}_M = (\delta, \nu, \vec{b})$. Note that instead of storing $(\delta, \nu, \vec{b})$, client $\mathcal{C}$ may only store (δ, ν, K_b), where K_b is the key employed by client $\mathcal{C}$ to generate vector $\vec{b}$. In the same manner, instead of publishing vector $\vec{\gamma}$, client $\mathcal{C}$ may choose to publish the key K_γ used to generate $\vec{\gamma}$. These two optimizations lead to a *constant storage* cost at the client, however, it demands that **i.)** client $\mathcal{C}$ generates $\vec{b}$ whenever it wants to multiply a vector $\vec{x}$ with matrix M and that **ii.)** verifier $\mathcal{V}$ generates $\vec{\gamma}$ whenever it would like to verify the server's results. This yields a slightly more expensive **Problem Generation** and **Verification** phases than the ones described in the original protocol.

To multiply a vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ with matrix M, client $\mathcal{C}$ first generates vector $\vec{b}$ using key K_b, then evaluates the dot product $\tau_x = \nu(\vec{b} \cdot \vec{x}^\intercal)$, computes $\mathsf{VK}_x = e(g, h)^{\tau_x}$, and finally queries server $\mathcal{S}$ with vector $\vec{x}$. This entails that the **Problem Generation** phase involves the generation of m random numbers, $m - 1$ additions and $m + 1$ multiplications in $\mathbb{F}_p$, and one exponentiation in $\mathbb{G}_T$. This is considerably cheaper than the solutions presented in [3] and [7].

After receipt of a multiplication query $\sigma_x = \vec{x}$, server $\mathcal{S}$ proceeds with the **Computation** phase which 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 $\mathbb{G}_1$.

In order to verify the validity of the result $\sigma_y = (\vec{y}, \Pi)$ output by server $\mathcal{S}$, verifier $\mathcal{V}$ generates vector $\vec{\gamma}$ using the key K_γ, evaluates the dot product $\vec{\gamma} \cdot \vec{y}^\intercal$ in $\mathbb{F}_p$, and computes one bilinear pairing, one exponentiation, and one multiplication in $\mathbb{G}_T$. As a result, the **Verification** phase of our protocol is much more efficient than the solutions introduced in [3] and [7], since it only requires a constant number of exponentiations and bilinear pairings.

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

5 Related Work

Outsourced verifiable computation has recently spurred the interest of the research community. Early work on verifiable computation propose protocols for interactive proofs [8, 9] such as probabilistically checkable proofs (PCP) [10], efficient arguments [11, 12] or muggle proofs [13, 14]. In such protocols, a prover interacts with a verifier as long as necessary, such that the former convinces the latter about the evaluation of arbitrary functions. In contrast with these proposals, our paper focuses on non-interactive solutions that are more practical in real-world scenarios.

Non-Interactive Proofs for Arbitrary Functions. Micali defines *computationally sound proofs* [15] that enable a prover to output a short certificate of the correctness of computation of a statement. This construction relies on the random oracle model and uses PCP and Merkle hash trees [16] as building blocks.

Gennaro et al. [1] first formalize the notion of non-interactive verifiable computation in the *amortized model*. Their solution combines the use of garbled boolean circuits with fully-homomorphic encryption (FHE). Other work described in [17, 18] also propose FHE-based schemes. However, FHE does not have any practical relevance yet and [1] only allows private verification. In comparison, we propose two efficient protocols for public verification that can be implemented in current settings inducing lower computational and storage costs than the ones incurred by FHE-based solutions.

In [6], Parno et al. proposes 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.

Pinocchio [5] applies succinct non-interactive arguments of knowledge (SNARK) [19] to the problem of public verifiable computation of arbitrary functions. Pinocchio converts the outsourced function into an arithmetic circuit which is then translated into a Quadratic Arithmetic Program. This yields a verification that is linear in the number of inputs and outputs of the outsourced function. In our protocol however, the setup and verification costs are lower than the one induced by the protocol in [5]. Furthermore, their construction relies on non-standard cryptographic assumptions, whereas in this paper we use weaker standard assumptions (XDH and co-CDH).

Another type of solutions use homomorphic MACs [20–22] or homomorphic signatures [23, 24]. These solutions generally induce a verification as costly as the computation of the outsourced function itself. Homomorphic MACs proposed by Backes et al. [22] take advantage of algebraic PRFs inducing efficient verification provided that the data is indexed and labeled. This solution however is suitable only for quadratic functions. Similarly, Catalano et al. [24] propose homomorphic signatures with efficient verification for a publicly verifiable computation scheme. Nevertheless, their construction uses expensive multilinear pairings and requires, as in [22], the indexation of data.

Verifiable Polynomial Evaluation. Benabbas et al. [2] propose two protocols for private verification based on small-domain algebraic PRFs, which render their solutions suitable for small inputs only. The first solution is secure under d-Strong Diffie-Hellman assumption, whereas the second is sound under DDH but is much less efficient. In comparison, our solution allows public verification and is secure under standard assumptions (co-CDH). In the same line of work, Fiore and Gennaro [3] combine new algebraic PRFs and bilinear pairings to design a publicly verifiable solution. Compared to [3], our protocol induces less computation both at the client during the setup phase and at the server for the evaluation of the polynomial. As a follow-up to the work of [3], the authors in [7] propose a solution that trades off the server's storage and the client's verification computation: They use algebraic PRFs (similar to those introduced in [3]) and break the delegated computation into several sub-computations verifiable with a single proof. The storage overhead is reduced, but the verification remains more expensive than our scheme. Another solution for public verification considers signatures for correct computation [4], and uses polynomial commitments [25] to construct these signatures. The cost of the verification depends on the degree d of the outsourced

	Client		Server	Verifier	Hardness
	Setup	Problem Generation	Computation	Computation	Assumptions
Fiore and Gennaro [3]	1 pairing $2(d+1)$ exp	1 pairing 1 exp	$d+1$ exp	1 pairing 1 exp	co-CDH DLin
Our scheme	$d-1$ exp	2 exp	$d-1$ exp	1 pairings 1 exp	co-CDH

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

	Client		Server	Verifier	Hardness
	Setup	Problem Generation	Computation	Computation	Assumptions
Fiore and Gennaro [3]	$3nm$ exp	n pairings $2(n+m)$ exp	nm exp	n pairings n exp	co-CDH DLin
Our scheme	nm exp	1 exp	nm exp	2 pairings 1 exp	co-CDH DDH

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

polynomial whereas our verification requires a constant amount of computation. Besides, the solution is secure under the d-Strong Diffie-Hellman assumption. In Table[3] 3, we compare our work with the only solution we know is sound under standard assumptions. That is, the construction proposed in [3], which relies on the decisional linear (DLin) assumption and the co-computational Diffie-Hellman (co-CDH) assumption.

Verifiable Matrix Multiplication. Fiore and Gennaro [3] propose algebraic PRFs for publicly verifiable delegation of matrix multiplications. The problem generation and the verification in [3] perform a linear number of exponentiations and bilinear pairings. In contrast, our protocol for verifiable matrix multiplication only computes dot products and a constant number of exponentiations. The authors in [7] propose a publicly verifiable scheme for vector-matrix multiplication: It tailors an algebraic PRFs (also used in [3]) and divides the delegated computation to a subset of smaller computations, all verifiable with a single proof. However, the computation costs for the problem generation and the verification are higher than the ones induced by our scheme. Table[3] 4 depicts a comparison of our proposal with the solution proposed in [3] which as ours is secure under standard assumptions.

6 Conclusion

In this paper, we introduced two protocols for publicly verifiable delegation of computation which enable a client to outsource securely the evaluation of arbitrary degree univariate polynomials and the multiplication of large matrices. Instead

[3]Tables 3 and 4 compare the computational complexity of our solution and the solution in [3] only in terms of exponentiations and bilinear pairings which are the most computationally prohibitive operations.

of employing algebraic pseudo-random functions, we built our protocols upon the *mathematical* properties of polynomials and matrices. This paved the way for efficient and practical solutions that are provably secure against adaptive adversaries under the co-CDH and the DDH assumptions.

References

[1] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computation: Outsourcing computation to untrusted workers," in *In Proceedings of CRYPTO*. Citeseer, 2010.

[2] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable Delegation of Computation over Large Datasets," in *Advances in Cryptology – CRYPTO 2011*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed. Springer Berlin Heidelberg, 2011, vol. 6841, pp. 111–131.

[3] D. Fiore and R. Gennaro, "Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. ACM, 2012, pp. 501–512.

[4] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography*. Springer, 2013, pp. 222–242.

[5] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 238–252.

[6] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, R. Cramer, Ed. Springer Berlin Heidelberg, 2012, vol. 7194, pp. 422–439.

[7] L. F. Zhang and R. Safavi-Naini, "Verifiable delegation of computations with storage-verification trade-off," in *Computer Security - ESORICS 2014*, ser. Lecture Notes in Computer Science, M. Kutyłowski and J. Vaidya, Eds. Springer International Publishing, 2014, vol. 8712, pp. 112–129.

[8] L. Babai, "Trading group theory for randomness," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85. ACM, 1985, pp. 421–429. [Online]. Available: http://doi.acm.org/10.1145/22145.22192

[9] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989. [Online]. Available: http://dx.doi.org/10.1137/0218012

[10] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 71–84.

[11] J. Kilian, "A note on efficient zero-knowledge proofs and arguments," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. ACM, 1992, pp. 723–732.

[12] ——, "Improved efficient arguments," in *Advances in Cryptology–CRYPTO'95*. Springer, 1995, pp. 311–324.

[13] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *Proceedings of the 40th annual ACM symposium on Theory of computing*. ACM, 2008, pp. 113–122.

[14] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 71–89.

[15] S. Micali, *Computationally-Sound Proofs*, ser. Lecture Notes in Logic. Berlin: Springer-Verlag, 1998, vol. Volume 11, pp. 214–268. [Online]. Available: http://projecteuclid.org/euclid.lnl/1235415908

[16] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology–CRYPTO'87*. Springer, 1988, pp. 369–378.

[17] K.-M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 483–501.

[18] M. Barbosa and P. Farshim, "Delegatable homomorphic encryption with applications to secure outsourcing of computation," in *Topics in Cryptology–CT-RSA 2012*. Springer, 2012, pp. 296–312.

[19] N. Bitansky, R. Canetti, A. Chiesa, and E. 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*. ACM, 2012, pp. 326–349.

[20] D. Catalano and D. Fiore, "Practical homomorphic macs for arithmetic circuits." in *EUROCRYPT*. Springer, 2013, pp. 336–352.

[21] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *Advances in Cryptology-ASIACRYPT 2013*. Springer, 2013, pp. 301–320.

[22] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM, 2013, pp. 863–874.

[23] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *Advances in Cryptology–EUROCRYPT 2011*. Springer, 2011, pp. 149–168.

[24] D. Catalano, D. Fiore, and B. Warinschi, "Homomorphic signatures with efficient verification for polynomial functions," in *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, pp. 371–389.

[25] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology-ASIACRYPT 2010*. Springer, 2010, pp. 177–194.

A Proof of Theorem 2

Theorem. *The solution proposed in Section 3.3 for publicly verifiable polynomial evaluation 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 polynomial evaluation with a non-negligible advantage ϵ. We demonstrate in what follows that there exists another adversary $\mathcal{B}$ that breaks the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage ϵ'.

The proof of soundness of our solution for publicly verifiable polynomial evaluation involves three games:

Game 0 This game corresponds to the soundness game (cf. Section 2.2) of our protocol for verifiable polynomial evaluation.

Game 1 In this game, adversary $\mathcal{B}$ simulates the soundness game to adversary $\mathcal{A}$ as follows:

When adversary $\mathcal{A}$ calls the 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 $\mathcal{O}_{\mathsf{Setup}}$ as follows:

- **i.)** It sets the public parameters of the evaluation of A to $\widehat{\mathsf{param}}_A = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, h)$ as in *Game 0*;
- **ii.)** it picks a random polynomial $B(X) = b_1 X + b_0$ that does not divide A and performs the Euclidean division of A by B. This yields a quotient polynomial $Q(X) = \sum_{i=0}^{d-1} q_i X^i$ and a remainder polynomial $R \neq 0$;
- **iii.)** it computes $\mathbf{q}_i = g^{q_i}$ for all $0 \leqslant i \leqslant d-1$;
- **iv.)** it defines the keys $\widehat{\mathsf{SK}}_A = (g, A, B, \{\mathbf{q}_i\}_{i=0}^{d-1})$ and $\widehat{\mathsf{EK}}_A = (A, \{\mathbf{q}_i\}_{i=0}^{d-1})$, then returns $\widehat{\mathsf{param}}_A$ and $\widehat{\mathsf{EK}}_A$ to adversary $\mathcal{A}$.

We point out here that the output of oracle $\mathcal{O}_{\mathsf{Setup}}$ in this game is identical to the output of $\mathcal{O}_{\mathsf{Setup}}$ in *Game 0*.

If adversary $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with input $x \in \mathbb{F}_p$ and evaluation key $\widehat{\mathsf{EK}}_A$, adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ by first computing $B(x) = b_1 x + b_0$. If $B(x) = 0$, then adversary $\mathcal{B}$ aborts the game; otherwise adversary $\mathcal{B}$ proceeds as follows:

- **i.)** It randomly selects $r \in \mathbb{F}_p^*$ and computes $\widehat{\mathsf{VK}}_{(x,B)} = e(g,h)^r$;
- **ii.)** it computes $A(x) = \sum_{i=0}^{d} a_i x^i \mod p$ and sets $\widehat{\mathsf{VK}}_{(x,R)} = \frac{\widehat{\mathsf{VK}}_{(x,B)}^{A(x)}}{e(\prod_{i=0}^{d-1} \mathbf{q}_i^{x^i}, h)}$;
- **iii.)** it provides adversary $\mathcal{A}$ with the encoding $\sigma_x = x$ and the verification key $\widehat{\mathsf{VK}}_x = (\widehat{\mathsf{VK}}_{(x,B)}, \widehat{\mathsf{VK}}_{(x,R)})$.

Note that if adversary $\mathcal{B}$ does not abort the game, then the output of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in this game is statistically indistinguishable from the output of $\mathcal{O}_{\mathsf{ProbGen}}$ in *Game 0*.

Indeed, given that $e(g,h)^{A(x)} = e(g,h)^{Q(x)B(x)} e(g,h)^R$, whereby $e(g,h)^R$ is unknown and B is a random polynomial, adversary $\mathcal{A}$ cannot distinguish between $e(g,h)^{B(x)^{-1}}$ and $e(g,h)^r$ for some randomly generated r. Therefore, adversary $\mathcal{A}$ cannot tell whether verification key $\widehat{\mathsf{VK}}_x = (\widehat{\mathsf{VK}}_{(x,B)}, \widehat{\mathsf{VK}}_{(x,R)})$ is computed correctly or not.

Game 2 The goal of adversary $\mathcal{B}$ in this game is to break the co-CDH assumption in $\mathbb{G}_2$ using adversary $\mathcal{A}$.

Let $\mathcal{O}_{\mathsf{co-cdh}}$ be an oracle which when queried returns the pair (g, g^α) in $\mathbb{G}_1$ and the pair (h, h^β) in $\mathbb{G}_2$ for randomly generated α, β in $\mathbb{F}_p$.

To break co-CDH, adversary $\mathcal{B}$ first calls oracle $\mathcal{O}_{\mathsf{co-cdh}}$ to obtain a tuple $(g, g^\alpha, h, h^\beta)$; then simulates the soundness game to adversary $\mathcal{A}$ as depicted below:

Learning Phase When adversary $\mathcal{A}$ calls the 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 $\mathcal{O}_{\mathsf{Setup}}$ as in *Game 1* as follows:

- **i.)** It sets the public parameters $\widetilde{\mathsf{param}}_A$ to $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, h)$;
- **ii.)** it computes $\breve{\mathbf{q}}_i = \breve{g}^{q_i}$ for all $0 \leqslant i \leqslant d-1$, where $\breve{g} = g^\alpha$;
- **iii.)** it defines the keys $\breve{\mathsf{SK}}_A = (\breve{g}, A, B, \{\breve{\mathbf{q}}_i\}_{i=0}^{d-1})$ and $\breve{\mathsf{EK}}_A = (A, \{\breve{\mathbf{q}}_i\}_{i=0}^{d-1})$;
- **iv.)** it returns the public parameters $\widetilde{\mathsf{param}}_A$ and $\breve{\mathsf{EK}}_A$ to adversary $\mathcal{A}$.

The distribution of the public parameters $\widetilde{\mathsf{param}}_A$ and the evaluation key $\breve{\mathsf{EK}}_A$ returned by adversary $\mathcal{B}$ is *statistically indistinguishable* to the distribution of $\widehat{\mathsf{param}}_A$ and $\widehat{\mathsf{EK}}_A$ in *Game 1*.

If adversary $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with input $x \in \mathbb{F}_p$ and evaluation key $\breve{\mathsf{EK}}_A$, then adversary $\mathcal{B}$ first checks whether $B(x) = 0 \mod p$. If so, adversary $\mathcal{B}$ aborts the soundness game; otherwise it simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ as in *Game 1* as depicted below:

i.) It computes $\breve{\mathsf{VK}}_{(x,B)}$ as $e(\breve{g}, h^\beta)^r$ for some randomly generated $r \in \mathbb{F}_p^*$;

ii.) it sets $\breve{\mathsf{VK}}_{(x,R)}$ to $\frac{\breve{\mathsf{VK}}_{(x,B)}^{A(x)}}{e(\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^i}, h)}$.

Notice that for all $x \in \mathbb{F}_p$ such that $B(x) \neq 0$, $\breve{\mathsf{VK}}_{(x,B)} = e(\breve{g}, h^\beta)^{B(x)^{-1}}$ is statistically indistinguishable from $\widehat{\mathsf{VK}}_{(x,B)}$ from *Game 1*. In addition,$\{\breve{\mathbf{q}}_i\}_{i=0}^{d-1}$ are statistically indistinguishable from $\{\mathbf{q}_i\}_{i=0}^{d-1}$ defined in *Game 1*. Hence, $\breve{\mathsf{VK}}_{(x,R)}$ as computed in this game is also statistically indistinguishable from $\widehat{\mathsf{VK}}_{(x,R)}$ computed in *Game 1*.

Challenge Phase Adversary $\mathcal{A}$ first picks a challenge input x^* and queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$. Accordingly, adversary $\mathcal{B}$ simulates $\mathcal{O}_{\mathsf{ProbGen}}$ by first evaluating polynomial B at point x^*. If $B(x^*) = 0$, then adversary $\mathcal{B}$ aborts the game; otherwise it proceeds as following:

i.) It computes $\breve{\mathsf{VK}}_{(x^*,B)}$ as $e(\breve{g}, h^\beta)^{B(x^*)^{-1}}$;

ii.) it sets $\breve{\mathsf{VK}}_{(x^*,R)}$ to $\frac{\breve{\mathsf{VK}}_{(x^*,B)}^{A(x^*)}}{e(\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}}, h)}$.

Thereafter, adversary $\mathcal{A}$ outputs a response $\sigma_{y^*} = (y^*, \pi^*)$ such that $y^* \neq A(x^*)$.

Upon receipt of pair $\sigma_{y^*} = (y^*, \pi^*)$, adversary $\mathcal{B}$ checks whether the following equality holds:

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

If so, then adversary $\mathcal{B}$ breaks co-CDH in $\mathbb{G}_1$ by computing:

$$g^{\alpha\beta} = \left(\frac{\pi^{*B(x^*)}}{(\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}})^{B(x^*)}} \right)^{(y^* - A(x^*))^{-1}}$$

Here we show that $\left(\frac{\pi^*}{\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}}} \right)^{B(x^*)(y^* - A(x^*))^{-1}}$ is indeed equal to $g^{\alpha\beta}$.

We remark that if $\sigma_{y^*} = (y^*, \pi^*)$ passes the verification then:

$$\breve{\mathsf{VK}}_{(x^*,B)}^{y^*} = e(\pi^*, h)\breve{\mathsf{VK}}_{(x^*,R)} \tag{4}$$

By construction, we also have:

$$\breve{\mathsf{VK}}_{(x^*,B)}^{A(x^*)} = e(\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}}, h)\breve{\mathsf{VK}}_{(x^*,R)} \tag{5}$$

By dividing Equation 4 with Equation 5, we obtain:

$$\breve{\mathsf{VK}}_{(x^*,B)}^{(y^*-A(x^*))} = e\left(\frac{\pi^*}{\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}}}, h\right)$$

As $\breve{\mathsf{VK}}_{(x^*,B)} = e(\breve{g}, h^\beta)^{B(x^*)^{-1}} = e(g^\alpha, h^\beta)^{B(x^*)^{-1}}$, we get:

$$e(g^{\alpha\beta}, h)^{B(x^*)^{-1}} = e(\frac{\pi^*}{\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}}}, h)$$

$$e(g^{\alpha\beta}, h) = e\left(\frac{\pi^{*B(x^*)}}{(\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}})^{B(x^*)}}, h\right)$$

and thus:

$$g^{\alpha\beta} = \left(\frac{\pi^*}{\prod_{i=0}^{d-1} \breve{\mathbf{q}}_i^{x^{*i}}}\right)^{B(x^*)(y^*-A(x^*))^{-1}}$$

In conclusion, if there is an adversary $\mathcal{A}$ that breaks the soundness of our protocol for publicly verifiable computation of polynomials with a non-negligible advantage ϵ, then there exists an adversary $\mathcal{B}$ that breaks the co-CDH assumption in $\mathbb{G}_1$ with some advantage ϵ' as long as adversary $\mathcal{B}$ does not abort the soundness game.

Here we quantify the advantage ϵ' of adversary $\mathcal{B}$ in breaking the co-CDH assumption in $\mathbb{G}_1$:

Let E_B denote event that adversary $\mathcal{B}$ breaks the co-CDH assumption and E_{abort} the event that adversary $\mathcal{B}$ aborts the soundness game.

$$\begin{aligned}\epsilon' = \Pr(E_B) &= \Pr(E_B \mid E_{\mathsf{abort}}).Pr(E_{\mathsf{abort}}) + \Pr(E_B \mid \overline{E_{\mathsf{abort}}}).\Pr(\overline{E_{\mathsf{abort}}}) \\ &\geqslant \Pr(E_B \mid \overline{E_{\mathsf{abort}}}).\Pr(\overline{E_{\mathsf{abort}}})\end{aligned}$$

If ϵ is the advantage of adversary $\mathcal{A}$ in breaking the soundness of our protocol for verifiable delegation of polynomial evaluation, then $\Pr(E_B \mid \overline{E_{\mathsf{abort}}}) = \epsilon$ and therewith $\epsilon' \geqslant \epsilon \Pr(\overline{E_{\mathsf{abort}}})$.

We recall that adversary $\mathcal{B}$ aborts the soundness game if and only if adversary $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with x such that $B(x) = 0 \mod p$. If we assume that adversary $\mathcal{A}$ makes at most $t+1$ queries to $\mathcal{O}_{\mathsf{ProbGen}}$ during the soundness game and given that B has one root in $\mathbb{F}_p$, then the probability that adversary $\mathcal{B}$ does not abort the soundness game is $\Pr(\overline{E_{\mathsf{abort}}}) = (1-1/p)^{t+1} \simeq 1 - t/p$.

Hence, adversary $\mathcal{B}$ breaks co-CDH in $\mathbb{G}_1$ with a non-negligible advantage $\epsilon' \geqslant \epsilon - \frac{\epsilon t}{p}$. □

B Proof of Theorem 4

Theorem. *The solution described in Section 4.3 for publicly verifiable matrix multiplication is sound under the DDH assumption in $\mathbb{G}_1$ and 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 build below another adversary $\mathcal{B}$ that uses adversary $\mathcal{A}$ to break the co-CDH assumption in $\mathbb{G}_1$ with a non-negligible advantage ϵ', provided that the DDH assumption holds in $\mathbb{G}_1$.

The proof of the soundness of our protocol for publicly verifiable matrix multiplication comprises the following sequence of games:

Game 0 This corresponds to the soundness game of the protocol described in Section 4.3.

Game 1 In this game, adversary $\mathcal{B}$ simulates the soundness game to adversary $\mathcal{A}$ as follows:

When adversary $\mathcal{A}$ calls the oracle $\mathcal{O}_{\mathsf{Setup}}$ with some matrix M of elements M_{ij} in $\mathbb{F}_p$, adversary $\mathcal{B}$ proceeds as algorithm Setup in *Game 0* except for the following:

- **i.)** It randomly generates the elements R_{ij} of matrix R, instead of computing them as $R_{ij} = a_i b_j \mod p$;
- **ii.)** it defines the elements in matrix $\widehat{\mathcal{N}}$ as $\widehat{\mathcal{N}}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$;
- **iii.)** it sets the secret key associated with matrix M to $\widehat{\mathsf{SK}}_M = (\delta, M, \widehat{\mathcal{N}})$.

Adversary $\mathcal{B}$ then ends the simulation of oracle $\mathcal{O}_{\mathsf{Setup}}$ by outputting the public parameters $\widehat{\mathsf{param}}_M = (p, \vec{\gamma}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, \tilde{h})$ and the public evaluation key $\widehat{\mathsf{EK}}_M = (M, \widehat{\mathcal{N}})$.

Note that given the public parameters $\widehat{\mathsf{param}}_M$ and evaluation key $\widehat{\mathsf{EK}}_M = (M, \widehat{\mathcal{N}})$, adversary $\mathcal{A}$ cannot distinguish between $\mathcal{N}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{a_i b_j}$ (see Section 4.3) or $\widehat{\mathcal{N}}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$ where R_{ij} are randomly generated as long as the DDH assumption holds in $\mathbb{G}_1$ (cf. Lemma 1). Therefore, adversary $\mathcal{A}$ cannot tell if it is actually interacting with oracle $\mathcal{O}_{\mathsf{Setup}}$ or with a simulation of oracle $\mathcal{O}_{\mathsf{Setup}}$.

Lemma 1. *Under the DDH assumption in $\mathbb{G}_1$, adversary $\mathcal{A}$ cannot distinguish between $\mathcal{N}_{ij} = \tilde{g}^{M_{ij}} g^{a_i b_j}$ and $\widehat{\mathcal{N}}_{ij} = \tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$, where R_{ij} are randomly generated for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$.*

For ease of exposition, the proof of Lemma 1 is deferred to Appendix C.

Now if adversary $\mathcal{A}$ queries the oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with some column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and evaluation key $\widehat{\mathsf{EK}}_M$, then adversary $\mathcal{B}$ (provided with secret key $\mathsf{SK}_M = (\delta, M, \widehat{\mathcal{N}})$) simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ by outputting the public encoding $\sigma_x = \vec{x}$ and the corresponding verification key:

$$\widehat{\mathsf{VK}}_x = \frac{e\left(\prod_{i=1}^{n}\prod_{j=1}^{m}\widehat{\mathcal{N}}_{ij}^{x_j}, h\right)}{e\left(g^{\vec{\gamma}\cdot(M\vec{x})^\intercal}, \tilde{h}\right)} \tag{6}$$

Notice that according to the description of the protocol in Section 4.3, the verification key VK_x that would be output by oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in *Game 0* verifies the following equation for all $\vec{x} \in \mathbb{F}_p^n$:

$$\mathsf{VK}_x = \frac{e\left(\prod_{i=1}^{n}\prod_{j=1}^{m}\mathcal{N}_{ij}^{x_j}, h\right)}{e\left(g^{\vec{\gamma}\cdot(M\vec{x})^\intercal}, \tilde{h}\right)}$$

Since $\mathcal{N}_{ij}$ and $\widehat{\mathcal{N}}_{ij}$ are computationally indistinguishable under the DDH assumption, so are VK_x and $\widehat{\mathsf{VK}}_x$. Thus, the output of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in this game is *computationally indistinguishable* from the output of the actual oracle $\mathcal{O}_{\mathsf{ProbeGen}}$ in *Game 0*, under the DDH assumption in $\mathbb{G}_1$.

Game 2 In this game adversary $\mathcal{B}$ simulates the soundness game to adversary $\mathcal{A}$ as follows:

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 $\mathcal{O}_{\mathsf{Setup}}$ as in *Game 1*, except that it generates a matrix $\check{\mathcal{N}}$ of elements $\check{\mathcal{N}}_{ij} = g^{N_{ij}}$, where N_{ij} are generated randomly in $\mathbb{F}_p$ for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$, and accordingly defines the secret key as $\widecheck{\mathsf{SK}}_M = (\delta, M, \check{\mathcal{N}})$ and the corresponding evaluation key as $\widecheck{\mathsf{EK}}_M = (M, \check{\mathcal{N}})$. Adversary $\mathcal{B}$ concludes its simulation of the oracle $\mathcal{O}_{\mathsf{Setup}}$ by outputting the public parameters $\widecheck{\mathsf{param}}_M = (p, \vec{\gamma}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, \tilde{h})$ and the public evaluation key $\widecheck{\mathsf{EK}}_M$.

We indicate here that matrix $\check{\mathcal{N}}$ is statistically indistinguishable from matrix $\widehat{\mathcal{N}}$ constructed in *Game 1*. As a result, the output of the simulation of oracle $\mathcal{O}_{\mathsf{Setup}}$ in this game is statistically indistinguishable from the output of the simulated oracle $\mathcal{O}_{\mathsf{Setup}}$ in *Game 1*.

If adversary $\mathcal{A}$ invokes oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with some column vector $\vec{x} = (x_1, x_2, ..., x_m)^\intercal$ and evaluation key $\widecheck{\mathsf{EK}}_M$, then adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ by outputting the encoding $\sigma_x = \vec{x}$ and the corresponding verification key:

$$\widecheck{\mathsf{VK}}_x = \frac{e\left(\prod_{i=1}^{n}\prod_{j=1}^{m}\check{\mathcal{N}}_{ij}^{x_j}, h\right)}{e\left(g^{\vec{\gamma}\cdot(M\vec{x})^\intercal}, \tilde{h}\right)} \tag{7}$$

Given that $\widehat{\mathsf{VK}}_x$ from *Game 1* verifies equation 6 for all $\vec{x} \in \mathbb{F}_p^n$ and given that matrix $\widehat{\mathcal{N}}$ has the same statistical distribution as matrix $\check{\mathcal{N}}$ defined in this game, we infer that $\widehat{\mathsf{VK}}_x$ is statistically identical to $\widecheck{\mathsf{VK}}_x$. Hence, the simulated response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in this game is *statistically indistinguishable* from the simulated response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in *Game 1*.

Game 3 In this game, adversary $\mathcal{B}$ would like to break the co-CDH assumption in $\mathbb{G}_1$ with the help of adversary $\mathcal{A}$. To this end, adversary $\mathcal{B}$ calls the oracle $\mathcal{O}_{\mathsf{co-cdh}}$ which in turn outputs the pair $(g, g^{\alpha}) \in \mathbb{G}_1^2$ and the pair $(h, h^{\beta}) \in \mathbb{G}_2^2$.

To simulate the soundness game to adversary $\mathcal{A}$, adversary $\mathcal{B}$ proceeds as following:

Learning phase When adversary $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Setup}}$ with some (n, m)-matrix M, adversary $\mathcal{B}$ acts as in *Game 2* except for the following:

- **i.)** It computes $(\tilde{g}, \tilde{h}) = ((g^{\alpha})^{\delta}, (h^{\beta})^{\delta})$ and sets the public parameters to $\overline{\mathsf{param}}_M = (p, \vec{\gamma}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \bar{g} = g^{\alpha}, h, \tilde{h})$;
- **ii.)** it generates a matrix $\bar{\mathcal{N}}$ of elements $\bar{\mathcal{N}}_{ij} = (\bar{g})^{N_{ij}}$, where N_{ij} are generated randomly in $\mathbb{F}_p$ for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$.

Adversary $\mathcal{B}$ ends the simulation of oracle $\mathcal{O}_{\mathsf{Setup}}$ by setting the secret key to $\overline{\mathsf{SK}}_M = (\delta, M, \bar{\mathcal{N}})$, and outputting the public parameters $\overline{\mathsf{param}}_M$ and the public evaluation key $\overline{\mathsf{EK}}_M$.

It is clear that the simulations of oracle $\mathcal{O}_{\mathsf{Setup}}$ in *Game 2* and in *Game 3* are indistinguishable.

If adversary $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{ProbGen}}$ with some vector $\vec{x} = (x_1, x_2, ..., x_m)^{\intercal}$ and evaluation key $\overline{\mathsf{EK}}_M$, then similarly to *Game 2* adversary $\mathcal{B}$ outputs the public encoding $\sigma_x = \vec{x}$ and the verification key:

$$\overline{\mathsf{VK}}_x = \frac{e\left(\prod_{i=1}^{n}\prod_{j=1}^{m}\bar{\mathcal{N}}_{ij}^{x_j}, h\right)}{e\left(\bar{g}^{\vec{\gamma}\cdot(M\vec{x})^{\intercal}}, \tilde{h}\right)} \tag{8}$$

Given that verification key $\breve{\mathsf{VK}}_x$ satisfies equation 7 for all $\vec{x} \in \mathbb{F}_p^n$, and that matrix $\breve{\mathcal{N}}$ defined in *Game 2* has the same statistical distribution as matrix $\bar{\mathcal{N}}$, we deduce that $\breve{\mathsf{VK}}_x$ and $\overline{\mathsf{VK}}_x$ are statistically identical, and therewith, the simulated response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in this game is *statistically indistinguishable* from the simulated response of oracle $\mathcal{O}_{\mathsf{ProbGen}}$ in *Game 2*.

Challenge phase Adversary $\mathcal{A}$ first picks a challenge vector $\vec{x}^* = (x_1^*, x_2^*, ..., x_m^*)^{\intercal}$ which it gives to oracle $\mathcal{O}_{\mathsf{ProbGen}}$ along with evaluation key $\overline{\mathsf{EK}}_M$. Adversary $\mathcal{B}$ simulates oracle $\mathcal{O}_{\mathsf{ProbGen}}$ as before by outputting the public encoding $\sigma_{x^*} = \vec{x}^*$ and verification key:

$$\overline{\mathsf{VK}}_{x^*} = \frac{e\left(\prod_{i=1}^{n}\prod_{j=1}^{m}\bar{\mathcal{N}}_{ij}^{x_j^*}, h\right)}{e\left(\bar{g}^{\vec{\gamma}\cdot(M\vec{x}^*)^{\intercal}}, \tilde{h}\right)}$$

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

To break the co-CDH assumption in $\mathbb{G}_1$, adversary $\mathcal{B}$ verifies whether $\vec{\gamma} \cdot \vec{y^*} = \vec{\gamma} \cdot (M\vec{x^*})^\intercal \mod p$. If so, adversary $\mathcal{B}$ aborts the game; otherwise it breaks co-CDH by returning:

$$g^{\alpha\beta} = \left(\frac{\Pi^*}{\prod_{i=1}^{n} \prod_{j=1}^{m} \bar{\mathcal{N}}_{ij}^{x_j^*}} \right)^{(\delta\vec{\gamma}\cdot(\vec{y^*}-M\vec{x^*})^\intercal)^{-1}}$$

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

$$e(\Pi^*, h) = e(\bar{g}^{\vec{\gamma}\cdot\vec{y^*}^\intercal}, \tilde{h})\overline{\mathsf{VK}}_{x^*} \tag{9}$$

Also by construction, we have:

$$e(\prod_{i=1}^{n}\prod_{j=1}^{m} \bar{\mathcal{N}}_{ij}^{x_j^*}, h) = e(\bar{g}^{\vec{\gamma}\cdot(M\vec{x^*})^\intercal}, \tilde{h})\overline{\mathsf{VK}}_{x^*} \tag{10}$$

By dividing Equation 9 with Equation 10, we obtain:

$$e\left(\frac{\Pi^*}{\prod_{i=1}^{n} \prod_{j=1}^{m} \bar{\mathcal{N}}_{ij}^{x_j^*}}, h \right) = \left(\frac{\bar{g}^{\vec{\gamma}\cdot\vec{y^*}^\intercal}}{\bar{g}^{\vec{\gamma}\cdot(M\vec{x^*})^\intercal}}, \tilde{h} \right)$$

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

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

Therefore if $\vec{\gamma} \cdot (\vec{y^*} - M\vec{x^*})^\intercal \neq 0 \mod p$, then $\delta\vec{\gamma} \cdot (\vec{y^*} - M\vec{x^*})^\intercal \neq 0 \mod p$ and we can compute:

$$g^{\alpha\beta} = \left(\frac{\Pi^*}{\prod_{i=1}^{n} \prod_{j=1}^{m} \bar{\mathcal{N}}_{ij}^{x_j^*}} \right)^{(\delta\vec{\gamma}\cdot(\vec{y^*}-M\vec{x^*})^\intercal)^{-1}}$$

Hence, adversary $\mathcal{B}$ breaks the co-CDH assumption in $\mathbb{G}_1$ as long as $\vec{\gamma} \cdot \vec{y^*}^\intercal \neq \vec{\gamma} \cdot (M\vec{x^*})^\intercal \mod p$. Fortunately, as stated in Lemma 2, the probability that $\vec{\gamma} \cdot \vec{y^*}^\intercal = \vec{\gamma} \cdot (M\vec{x^*})^\intercal$ is equal to $\frac{1}{p}$ which is negligible.

Lemma 2. *The probability that $\vec{\gamma} \cdot \vec{y^*}^\intercal = \vec{\gamma} \cdot (M\vec{x^*})^\intercal$ is $\frac{1}{p}$.*

The proof of this lemma can be found in Appendix D.

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 $\epsilon' \geqslant \epsilon(1-\frac{1}{p})$, provided that the DDH assumption holds in $\mathbb{G}_1$. $\square$

C Proof of Lemma 1

Lemma. *Under the DDH assumption in $\mathbb{G}_1$, adversary $\mathcal{A}$ cannot distinguish between $\tilde{g}^{M_{ij}} g^{a_i b_j}$ and $\tilde{g}_i^{M_{ij}} g_i^{R_{ij}}$, where R_{ij} are randomly generated for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$.*

Proof Sketch. To prove Lemma 1, we proceed in *two steps*. **i.)** We first use a sequence of m games to show that the distribution $\mathbf{D} = \left\{(g^{a_i b_1}, g^{a_i b_2}, ..., g^{a_i b_m})\right\}_{i \in \{1,2\}}$ and the random distribution $\breve{\mathbf{D}} = \left\{(g^{R_{i1}}, g^{R_{i2}}, ..., g^{R_{im}})\right\}_{i \in \{1,2\}}$ are computationally indistinguishable under the DDH assumption in $\mathbb{G}_1$. **ii.)** Then we show that if there exists an adversary $\mathcal{A}$ that distinguishes between distribution $\mathcal{D} = \left\{(g^{a_i b_1}, g^{a_i b_2}, ..., g^{a_i b_m})\right\}_{1 \leqslant i \leqslant n}$ and distribution $\breve{\mathcal{D}} = \left\{(g^{R_{i1}}, g^{R_{i2}}, ..., g^{R_{im}})\right\}_{1 \leqslant i \leqslant n}$, then there exists another adversary $\mathcal{B}$ that distinguishes between $\mathbf{D}$ and $\breve{\mathbf{D}}$, which leads to a contradiction under the DDH assumption.

For ease of exposition, we denote $\hat{g}_i = g^{a_i b_1}$ and we assume w.l.o.g. that $b_1 \neq 0 \mod p$. This leads to the following simplifications:

$$\mathbf{D} = \left\{(\hat{g}_i, \hat{g}_i^{\beta_{11}}, ..., \hat{g}_i^{\beta_{1m-1}})\right\}_{i \in \{1,2\}}$$
$$\mathcal{D} = \left\{(\hat{g}_i, \hat{g}_i^{\beta_{11}}, ..., \hat{g}_i^{\beta_{1m-1}})\right\}_{1 \leqslant i \leqslant n}$$

where $\beta_{1j} = b_{j+1}/b_1$ for all $1 \leqslant j \leqslant m-1$.

Similarly, if we denote $g^{R_{i1}} = \hat{g}_i$ and if we assume that $R_{i1} \neq 0 \mod p$, then $\breve{\mathbf{D}}$ and $\breve{\mathcal{D}}$ can be rewritten as:

$$\breve{\mathbf{D}} = \left\{(\hat{g}_i, \hat{g}_i^{\beta_{i1}}, ..., \hat{g}_i^{\beta_{im-1}})\right\}_{i \in \{1,2\}}$$
$$\breve{\mathcal{D}} = \left\{(\hat{g}_i, \hat{g}_i^{\beta_{i1}}, ..., \hat{g}_i^{\beta_{im-1}})\right\}_{1 \leqslant i \leqslant n}$$

where $\beta_{ij} = R_{ij+1}/R_{i1}$ for all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m-1$.

i.) Distributions $\mathbf{D}$ and $\breve{\mathbf{D}}$ are indistinguishable In the rest of this section, we denote $\Pr(\mathcal{A}(\mathbf{D}_k) = 1)$ the probability that a distinguisher $\mathcal{A}$ outputs 1 on input of a distribution $\mathbf{D}_k$.

Game 0 In this game, we set $\mathbf{D}_0 = \mathbf{D}$.

Game 1 In this game, we define:

$$\mathbf{D}_1 = \left\{(\hat{g}_1, \hat{g}_1^{\beta_{11}}, \hat{g}_1^{\beta_{12}}, ..., \hat{g}_1^{\beta_{1m-1}}), (\hat{g}_2, \hat{g}_2^{\beta_{21}}, \hat{g}_2^{\beta_{12}} ..., \hat{g}_2^{\beta_{1m-1}})\right\}$$

where β_{1j} and β_{21} are randomly generated in $\mathbb{F}_p$.

Under the DDH assumption in $\mathbb{G}_1$, distributions $\mathbf{D}_1$ and $\mathbf{D}_0$ are computationally indistinguishable. More formally, if $\mathcal{A}$ is a distinguisher between $\mathbf{D}_1$ and $\mathbf{D}_0$, then

$$|\Pr(\mathcal{A}(\mathbf{D}_1) = 1) - \Pr(\mathcal{A}(\mathbf{D}_0) = 1)| \leqslant \epsilon$$

where ϵ is the negligible advantage to solve the DDH problem in $\mathbb{G}_1$.

Game k In this game, we set:

$$\begin{aligned}\mathbf{D}_{k-1} &= \left\{(\hat{g}_1, \hat{g}_1^{\beta_{11}}, ..., \hat{g}_1^{\beta_{1m-1}}), (\hat{g}_2, \hat{g}_2^{\beta_{21}}, ..., \hat{g}_2^{\beta_{2k-1}}, \hat{g}_2^{\beta_{1k}}, \hat{g}_2^{\beta_{1k+1}} ..., \hat{g}_2^{\beta_{1m-1}})\right\} \\ \mathbf{D}_k &= \left\{(\hat{g}_1, \hat{g}_1^{\beta_{11}}, ..., \hat{g}_1^{\beta_{1m-1}}), (\hat{g}_2, \hat{g}_2^{\beta_{21}}, ..., \hat{g}_2^{\beta_{2k-1}}, \hat{g}_2^{\beta_{2k}}, \hat{g}_2^{\beta_{1k+1}}, ..., \hat{g}_2^{\beta_{1m-1}})\right\}\end{aligned}$$

Again under the DDH assumption:

$$|\Pr(\mathcal{A}(\mathbf{D}_k) = 1) - \Pr(\mathcal{A}(\mathbf{D}_{k-1}) = 1)| \leqslant \epsilon$$

and accordingly:

$$|\Pr(\mathcal{A}(\mathbf{D}_k) = 1) - \Pr(\mathcal{A}(\mathbf{D}_0) = 1)| \leqslant k\epsilon$$

Game $m-1$ Let in this game

$$\begin{aligned}\mathbf{D}_{m-2} &= \left\{(\hat{g}_1, \hat{g}_1^{\beta_{11}}, ..., \hat{g}_1^{\beta_{1m-1}}), (\hat{g}_2, \hat{g}_2^{\beta_{21}}, ..., \hat{g}_2^{\beta_{2m-2}}, \hat{g}_2^{\beta_{1m-1}})\right\} \\ \mathbf{D}_{m-1} &= \left\{(\hat{g}_1, \hat{g}_1^{\beta_{11}}, ..., \hat{g}_1^{\beta_{1m-1}}), (\hat{g}_2, \hat{g}_2^{\beta_{21}}, ..., \hat{g}_2^{\beta_{2m-2}}, \hat{g}_2^{\beta_{2m-1}})\right\}\end{aligned}$$

(i.e. $\mathbf{D}_{m-1} = \check{\mathbf{D}}$).

Similarly to GAME 0 and GAME k and under the DDH assumption:

$$|\Pr(\mathcal{A}(\mathbf{D}_{m-1}) = 1) - \Pr(\mathcal{A}(\mathbf{D}_{m-2}) = 1)| \leqslant \epsilon$$

Thus the advantage adv of distinguishing between distribution $\mathbf{D}$ and distribution $\check{\mathbf{D}}$ satisfies the following inequality:

$$\begin{aligned}\mathsf{adv} &= |\Pr(\mathcal{A}(\check{\mathbf{D}}) = 1) - \Pr(\mathcal{A}(\mathbf{D}) = 1)| \\ &= |\Pr(\mathcal{A}(\mathbf{D}_{m-1}) = 1) - \Pr(\mathcal{A}(\mathbf{D}_0) = 1)| \\ &\leqslant (m-1)\epsilon\end{aligned}$$

Hence, we deduce that $\mathbf{D}$ and $\check{\mathbf{D}}$ are computationally indistinguishable under the DDH assumption.

ii.) Distributions $\mathcal{D}$ and $\check{\mathcal{D}}$ are indistinguishable Let $\mathcal{B}$ be a distinguisher between $\mathbf{D}$ and $\check{\mathbf{D}}$ such that when given a distribution $\mathbf{D}' = \left\{(\hat{g}'_i, \hat{g}_i'^{\beta_{i1}}, ..., \hat{g}_i'^{\beta_{im-1}})\right\}_{i \in \{1,2\}}$, $\mathcal{B}$ outputs $b = 1$ if it thinks that $\beta_{1j} = \beta_{2j}$ for all $1 \leqslant j \leqslant m$, and outputs $b = 0$ if it thinks that β_{1j} and β_{2j} are randomly generated for all $1 \leqslant j \leqslant m$.

Suppose that there is a distinguisher $\mathcal{A}$ that distinguishes between $\mathcal{D}$ and $\check{\mathcal{D}}$ with a non-negligible advantage ϵ. More formally, given a distribution $\mathcal{D}' = \left\{(\hat{g}'_i, \hat{g}_i'^{\beta_{i1}}, ..., \hat{g}_i'^{\beta_{im-1}})\right\}_{1 \leqslant i \leqslant n}$, $\mathcal{A}$ outputs $b = 1$ if it believes that $\beta_{1j} = \beta_{ij}$ for all $2 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$, and outputs $b = 0$ otherwise.

In the following, we show how to construct a distinguisher $\mathcal{B}$ that uses $\mathcal{A}$ to distinguish between $\mathbf{D}$ and $\check{\mathbf{D}}$.

Without loss of generality, assume that distinguisher $\mathcal{B}$ is given a distribution $\mathbf{D}' = \left\{(\hat{g}'_i, \hat{g}_i'^{\beta_{i1}}, ..., \hat{g}_i'^{\beta_{im-1}})\right\}_{i \in \{1,2\}}$ and has to determine whether for all $1 \leqslant j \leqslant m$ $\beta_{1j} = \beta_{2j}$.

From distribution $\mathbf{D}'$, adversary $\mathcal{B}$ builds new distribution $\mathcal{D}' = \left\{(\hat{g}'_i, \hat{g}_i'^{\beta_{i1}}, ..., \hat{g}_i'^{\beta_{im-1}})\right\}_{1 \leqslant i \leqslant n}$ as follows:

- for $i \neq 1$ and $i \neq 2$, $\mathcal{B}$ selects two random numbers δ_{i1} and δ_{i2} in $\mathbb{F}_p$ and computes $\hat{g}'_i = \hat{g}_1'^{\delta_{i1}} \hat{g}_2'^{\delta_{i2}}$;
- for $i \neq 1$ and $i \neq 2$, adversary $\mathcal{B}$ computes $\hat{g}_i'^{\beta_{ij}} = (\hat{g}_1'^{\beta_{1j}})^{\delta_{i1}} (\hat{g}_2'^{\beta_{2j}})^{\delta_{i2}} = \hat{g}_1'^{\delta_{i1}\beta_{1j}} \hat{g}_2'^{\delta_{i2}\beta_{2j}}$.

To decide given $\mathbf{D}' = \left\{(\hat{g'}_i, \hat{g}_i'^{\beta_{i1}}, ..., \hat{g}_i'^{\beta_{im-1}})\right\}_{i \in \{1,2\}}$ if $\beta_{1j} = \beta_{2j}$ for all $1 \leqslant j \leqslant m$, adversary $\mathcal{B}$ feeds adversary $\mathcal{A}$ with distribution $\mathcal{D}' = \left\{(\hat{g}'_i, \hat{g}_i'^{\beta_{i1}}, ..., \hat{g}_i'^{\beta_{im-1}})\right\}_{1 \leqslant i \leqslant n}$ defined above. Adversary $\mathcal{A}$ in turn outputs a bit b such that $b = 1$ if it believes that $\beta_{ij} = \beta_{1j}$ for all $2 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$, and $b = 0$ otherwise.

We note now that if we assume that $\hat{g}'_2 = \hat{g}_1'^{\alpha}$ for some $\alpha \in \mathbb{F}_p^*$, then this entails that for all $i \neq 1, 2$ and $1 \leqslant j \leqslant m$:

$$\beta_{ij} = \frac{\delta_{i1}\beta_{1j} + \alpha\delta_{i2}\beta_{2j}}{\delta_{i1} + \alpha\delta_{2i}}$$

By the same token, if $\beta_{1j} = \beta_{2j}$, then this means that $\beta_{ij} = \beta_{1j}$ for all $2 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$[2].

It follows that if adversary $\mathcal{A}$ returns $b = 1$, then this implies that $\beta_{1j} = \beta_{2j}$ ($1 \leqslant j \leqslant m$). Otherwise, adversary $\mathcal{B}$ concludes that β_{1j} and β_{2j} are randomly generated for all $1 \leqslant j \leqslant m$. □

[2]Notice that $\delta_{i1} + \alpha\delta_{2i} \neq 0 \mod p$ with probability $1 - \frac{1}{p}$.

D Proof of Lemma 2

Lemma. *If* $\vec{\gamma} = (\gamma_1, \gamma_2, ..., \gamma_n)$ *is a random vector in* $\mathbb{F}_p^n$*, then for any pair of* distinct *column vectors* $\vec{y}_1$ *and* $\vec{y}_2$ *in* $\mathbb{F}_p^n$ *the probability that* $\vec{\gamma} \cdot (\vec{y}_1 - \vec{y}_2)^\intercal = 0 \mod p$ *is* $\frac{1}{p}$.

Proof. Let $\vec{z}$ denote $\vec{y}_1 - \vec{y}_2$ and $\phi : \mathbb{F}_p^n \to \mathbb{F}_p$ denote the linear form defined as: $\forall \vec{x} \in \mathbb{F}_p^n$, $\phi(\vec{x}) = \vec{x} \cdot \vec{z}^\intercal$.

Let $\mathsf{Ker}_\phi = \{\vec{x} \in \mathbb{F}_p^n, \phi(\vec{x}) = 0\}$ (i.e. Ker_ϕ is the kernel of the linear form ϕ).

Since ϕ is a linear form, the dimension of kernel Ker_ϕ is $n-1$. This means that Ker_ϕ is isomorphic to $\mathbb{F}_p^{n-1}$ and that the cardinality of Ker_ϕ is equal to p^{n-1}.

Now the probability that $\vec{\gamma} \cdot \vec{z}^\intercal = \phi(\vec{\gamma}) = 0 \mod p$ corresponds to the probability that $\vec{\gamma}$ is in Ker_ϕ. Since $\vec{\gamma}$ is a random vector in $\mathbb{F}_p^n$, the probability that $\vec{\gamma} \in \mathsf{Ker}_\phi$ equals:

$$\frac{|\mathsf{Ker}_\phi|}{|\mathbb{F}_p^n|} = \frac{p^{n-1}}{p^n} = \frac{1}{p}$$

□