# Efficient Secure and Verifiable Outsourcing of Matrix Multiplications

Yihua Zhang and Marina Blanton

Department of Computer Science and Engineering, University of Notre Dame
yzhang16@nd.edu, mblanton@nd.edu

**Abstract.** With the emergence of cloud computing services, a resource-constrained client can outsource its computationally-heavy tasks to cloud providers. Because such service providers might not be fully trusted by the client, the need to verify integrity of the returned computation result arises. The ability to do so is called verifiable delegation or verifiable outsourcing. Furthermore, the data used in the computation may be sensitive and it is often desired to protect it from the cloud throughout the computation. In this work, we put forward solutions for verifiable outsourcing of matrix multiplications that favorably compare with the state of the art. Our goal is to minimize the cost of verifying the result without increasing overhead associated with other aspects of the scheme. In our scheme, the cost of verifying the result of computation uses only a single modulo exponentiation and the number of modulo multiplications linear in the size of the output matrix. This cost can be further reduced to avoid all cryptographic operations if the cloud is rational. A rational cloud is neither honest nor arbitrarily malicious, but rather economically motivated with the sole purpose of maximizing its monetary reward. We extend our core constructions with several desired features such as data protection, public verifiability, and computation chaining.

## 1 Introduction

The emergence of cloud computing technologies enables clients who are unable to procure and maintain their own computing infrastructure to resort to convenient on-demand computing resources. Despite the paradigm being economically sensible for resource-limited clients, it comes with new security and privacy concerns. One of them is the lack of transparency and control over the outsourced computation, which necessitates the need to verify the result to guarantee integrity of the computation. Another is the need to protect confidentiality of the data used in the computation. Addressing these security objectives is the focus of this work.

Computation outsourcing to a cloud computing provider is common today and can take different forms. In particular, in addition to the conventional scenario when a computationally-limited client outsources its computation to the cloud and receives the result, there are many uses of cloud computing that involve multiple entities. For example, a doctor's office might send computation associated with a patient's test to a cloud provider, while the patient in question is also entitled to access to the result of the computation and thus should be able to verify integrity of the returned result. This calls for solutions where the integrity of the result can be verified by entities who do not have access to secret keys thus achieving public verifiability. Furthermore, if the task result may be verified by different entities or verified by the same entity multiple times over time (e.g., every time the result is used), then it is desirable to lower the overhead associated with verifying the result of the outsourced computation without increasing other costs.

The specific problem that we treat here is matrix multiplication outsourcing. Because of popularity of matrix multiplication in a number of different domains and relatively high cost of this operation on large inputs, secure matrix multiplication outsourcing has received attention [4, 5, 18, 30]. We continue this line of research and show below how our results compare to the state of the art.

A novel feature that we propose in this work and which is not present in publications on the same topic is as follows: we divide the overall computation in multiple stages and associate a key with each of them. Only if the computation in the current stage is performed correctly and the correct key is recovered, the server can learn the computation used in the next stage. Without the correct key, it is computationally infeasible to proceed to the next stage and pass verification in any of the stages that follow. This feature allows us to achieve several goals:

1. Chaining of computation from one stage to another allows for more efficient verification of the overall task. That is, corrupting a single cell of the product matrix invalidates the values in all other cells that follow, and verifying the result of the final stage is sufficient in ensuring that the entire task was performed correctly. Other publications (such as [18, 9]), on the other hand, require verification of every matrix cell to ensure correctness of the output.
2. If the server misbehaves during the computation and produces incorrect values for one or more cells of the product matrix, in order to proceed with the computation, it has to invest into substantially larger computation. In other words, the effect of the server's misbehavior is enlarged to the maximum extent, where any deviation from the computation substantially increases the computation cost. Thus, this mechanism is designed to deter the server from deviating from the correct computation.
3. When the result is returned to the client and does not pass verification, the client can efficiently identify the first stage during which the server deviated from the prescribed computation and ask the same or different server to rerun the computation starting from that stage.
4. If the server carries out the computation honestly, but gets compromised or infected by malware that corrupts the computation, the server can use the checkpoints between the stages to efficiently determine that corruption took place and quickly recover from it. That is, if the server is unaware of the compromise and continues with the task, the outcome will not pass verification and the computational effort becomes wasted. With the checkpoints, on

the other hand, the server will identify the problem, stop the computation, resolve the problem, and resume the task without wasting its efforts.

**Our contributions** can be summarized as follows:

- We present the core construction for verifiable outsourcing in presence of malicious adversaries who arbitrarily deviates from the prescribed computation (Section 4). In our scheme, delegating a task requires work linear in the input size, carrying out the computation itself has the same complexity as that of conventional matrix multiplication algorithm (i.e., $O(n^3)$ for matrices of dimension $n \times n$), and verification of the resulting matrix product involves only a single modulo exponentiation and the number of modulo multiplications linear in the output size.
- We present another construction that assumes rational adversaries who are neither honest nor malicious, but rather economically motivated with the sole purpose of maximizing the reward (Section 5.1). Under this adversarial model, verification of the returned result involves only a single comparison.
- We extend the construction in the rational setting to incorporate the chaining feature described above without compromising other properties (Section 5.2). In particular, this has no impact on the complexity of the resulting scheme.
- We also sketch how data privacy and public verifiability, in which any entity with access to public verification key can assess the validity of the returned result, can be added to the constructions in both malicious and rational settings (Section 6). This does not increase asymptotic complexities of the scheme, but the cost of recovering the output or verification time, resp., may increase. Note that when public verifiability is combined with data protection, access to the verification key does not allow for data recovery.

All our schemes achieve public delegatability, which means the entity who runs system setup can be different from the entities who form a task to be outsourced.

In reducing the cost associated with verifiable computation schemes for matrix multiplication outsourcing, our focus was on reducing the cost of verifying the result as this may be a more frequently used operation or an operation performed by weaker clients. The cost of task preparation in our and other schemes requires $O(n^2)$ cryptographic operations for input matrices of size $n \times n$, i.e., linear in the size of input. The server's work for carrying out matrix multiplication is $O(n^3)$ cryptographic operations, i.e., uses the conventional matrix multiplication algorithm.

We note that the cost of $O(n^2)$ cryptographic operations used in task preparation is rather high and will exceed the cost of computing matrix multiplication locally for small matrices. In particular, matrix multiplication algorithms of asymptotic complexity as low as $O(n^{2.373})$ are known, but the huge constants hidden behind the big-O notation prevent most of them from being used in practice (i.e., they require more than $2n^3$ work) [3]. In particular, for matrices of dimension $n < 10^{20}$, only the algorithm by Strassen 1969 and Winograd 1971 of complexity $O(n^{2.807})$ and the technique of trilinear aggregating of complexity

$O(n^{2.775})$ result in implementations of matrix multiplications that outperform the conventional $O(n^3)$ algorithm [3, 28]. This means that our and related constructions reduce the cost of matrix multiplication for the client only for large matrices when performing $n^2$ cryptographic operations is below $O(n^{2.775})$ work.

Before we proceed with the description of our schemes, we discuss related work (Section 2) and provide background information and definitions (Section 3).

## 2 Related Work

**Verifiable Computation**. In verifiable computation [21, 20, 16, 2, 32, 18, 13, 15], a client outsources a computationally intensive task to a server and verifies its results in an efficient manner upon its completion. The basic question was proposed in work on interactive proofs (IP) [8, 22], efficient arguments based on probabilistically checkable proofs (PCP) [26, 27], and computationally sound (CS) proofs [29]. Such schemes are not generally suitable for our goal (due to, e.g., vast client's storage requirements or the need for interactive verification). Parno et al. recently introduced Pinocchio [31], which allows execution of a general function represented as a circuit to be delegated to an untrusted worker. The cost of output verification in [31] is linear in the size of the function's input and output, but requires only a constant number of most expensive operations (pairing evaluation), which resembles similarities to our scheme in presence of malicious adversaries. Our verification cost is still lower in practical terms and is further reduced in the schemes with rational adversaries. The asymptotic complexity of the server's computation in [31] is the same as in our scheme, but our solution offers faster performance. Lastly, the setup of [31] uses a different key generation phase from our problem generation phase, which is applied to a function instead of function's input in our solution. The cost of key generation in [31] is higher than the cost of problem generation in our solution, but the key generation algorithm in [31] may be executed less frequently.

**Homomorphic MAC and Signature**. A homomorphic MAC [1] allows an entity to use a secret key $sk$ to produce a tag $\sigma$ that authenticates message $m$ with an additional property that, given a set of authenticators $\sigma_1, \sigma_2, \ldots, \sigma_n$ for messages $m_1, m_2, \ldots, m_n$, any entity with possession of public parameters can homomorphically evaluate a function $P$ on $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ to produce a short tag $\sigma'$ that authenticates correctness of $m' = P(m_1, m_2, \ldots, m_n)$. In the public-key setting, signatures are used to replace MACs and achieve similar functionality [25]. While homomorphic MACs or signatures can be used to realize verifiable computation for problems of certain structure, with such solutions the cost of verification is not smaller than the cost of executing the task. Furthermore, it is not intuitive as to how to protect privacy of the underlying messages using these techniques because neither MACs nor signatures are designed for this purpose.

**Matrix Computation**. The problem of verifiable matrix computation has been studied in recent literature [18, 30, 4, 9]. In addition to computation verification,

| Scheme | Verifiable Computation | Data Privacy | Public Verifiability | Output-Indep. Verification | Deterministic Verification |
|---|---|---|---|---|---|
| Atallah et al. [4] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Mohassel [30] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Fiore et al. [18] | ✓ | ✗ | ✓ | ✗ | ✓ |
| Backes et al. [9] | ✓ | ✗ | ✗ | ✗ | ✓ |
| This work | ✓ | ✓ | ✓ | ✓ | ✓ |

**Table 1.** Comparison with related work.

existing solutions offer other important security features that are: 1) *data protection*, i.e., protection of both input and output matrices throughout the computation; 2) *public verifiability*, i.e., the ability of any entity to verify the result of outsourced computation; and 3) *deterministic verification*, i.e., the ability to detect faulty cells in an output matrix with probability 1 (minus a negligible function of the security parameter due to computational assumptions). Unlike prior work, our scheme achieves all these features. Additionally, we achieve another property called *output-independent efficiency* that allows for a constant number of cryptographic operations (modulo exponentiations or pairing operations) to be used during verification independent of the size of the output matrix. Table 1 summarizes features of our solution and other schemes. Note that [18] has similar security features to ours, and although not specified in that work, it is feasible to incorporate privacy protection into their scheme as realized in [33]. We, however, found the scheme cannot be adjusted to efficiently handle rational adversaries.

Next, we provide a more detailed comparison of our work with closely related schemes. Note that we treat deterministic verification as a property that is nontrivial to achieve and thus consider only the work of Fiore et al. [18] and Backes et al. [9]. The construction of [9], however, is based on the scheme of [18] and would offer the same performance as that of [18] in our setting (the advantage of [9] is that it allows for more flexible function specification and scheduling). Thus, we list only performance of [18] as the representative of both [18] and [9].

The computational overhead for the client and the server is presented in Table 2, where $n$ represents the size of each dimension of input matrices, and $c_m$, $c_e$, and $c_p$ denote the time to carry out a modular multiplication, exponentiation, and a pairing operation, respectively. In the table, we use notation $\mathsf{VC}_m$ and $\mathsf{VC}_r$ to denote our verifiable computation schemes for malicious and rational adversaries, respectively. Note that some of the constructions for rational adversary do not involve any cryptographic operations for verification (and rather perform a single comparison) and that work is listed as $O(1)$. Finally, note that in some of our constructions for the rational adversary, the work for task preparation or server's computation is increased compared to the equivalent constructions for the malicious adversary, but the verification cost is substantially (and asymptotically) reduced.

| Scheme | Client's Preparation | Server's Computation | Client's Verification |
|---|---|---|---|
| Fiore et al. [18] (priv. ver.) | $(4c_e + 3c_m)n^2$ | $c_e n^3$ | $(c_e + c_m)n^2$ |
| $\mathsf{VC}_m$ (priv. ver.) | $(2c_e + 4c_m)n^2$ | $c_p n^3$ | $c_e + c_m n^2$ |
| $\mathsf{VC}_r$ (priv. ver.) | $(4c_e + 6c_m)n^2$ | $c_p n^3 + (c_e + c_m)n^2$ | $O(1)$ |
| Fiore et al. [18] (pub. ver.) | $(4c_e + 3c_m)n^2$ | $c_e n^3$ | $(c_p + c_e + c_m)n^2$ |
| $\mathsf{VC}_m$ (pub. ver.) | $(2c_e + 4c_m)n^2$ | $c_e n^3$ | $c_p n + (c_e + c_m)n^2$ |
| $\mathsf{VC}_r$ (pub. ver.) | $(4c_e + 6c_m)n^2$ | $c_p n^3 + (c_e + c_m)n^2$ | $c_e$ |
| $\mathsf{VC}_m$ (priv. ver. + privacy) | $(2c_e + 6c_m)n^2$ | $2c_p n^3$ | $c_e + c_m n^2$ |
| $\mathsf{VC}_r$ (priv. ver. + privacy) | $(5c_e + 8c_m)n^2$ | $2c_p n^3 + c_m n^2$ | $O(1)$ |
| $\mathsf{VC}_m$ (pub. ver. + privacy) | $(8c_e + 16c_m)n^2$ | $4c_e n^3$ | $4c_p n + 4(c_e + c_m)n^2$ |
| $\mathsf{VC}_r$ (pub. ver. + privacy) | $(5c_e + 8c_m)n^2$ | $2c_p n^3 + c_m n^2$ | $c_e$ |

**Table 2.** Computation in our and mostly closely related schemes.

| Scheme | Client's Storage | Server's Storage | Communication |
|---|---|---|---|
| Fiore et al. [18] (priv. ver.) | $n^2\kappa$ | $n^2\kappa + 2n^2$ | $2n^2\kappa + 3n^2$ |
| $\mathsf{VC}_m$ (priv. ver.) | $3n\kappa$ | $2n^2\kappa + 2n^2$ | $(2n^2 + 1)\kappa + 3n^2$ |
| $\mathsf{VC}_r$ (priv. ver.) | $\kappa$ | $2n^2\kappa + 2n^2$ | $(2n^2 + 1)\kappa + 3n^2$ |
| Fiore et al. [18] (pub. ver.) | $n^2\kappa$ | $n^2\kappa + 2n^2$ | $2n^2\kappa + 3n^2$ |
| $\mathsf{VC}_m$ (pub. ver.) | $(n^2 + n)\kappa$ | $n^2\kappa + 2n^2$ | $(n^2 + n)\kappa + 3n^2$ |
| $\mathsf{VC}_r$ (pub. ver.) | $\kappa$ | $2n^2\kappa + 2n^2$ | $(2n^2 + 1)\kappa + 3n^2$ |
| $\mathsf{VC}_m$ (priv. ver. + privacy) | $3n\kappa$ | $4n^2\kappa$ | $(5n^2 + 1)\kappa$ |
| $\mathsf{VC}_r$ (priv. ver. + privacy) | $(n^2 + 2n)\kappa$ | $4n^2\kappa$ | $(5n^2 + 1)\kappa$ |
| $\mathsf{VC}_m$ (pub. ver. + privacy) | $4(n^2 + n)\kappa$ | $4n^2\kappa + 8n^2$ | $4(n^2 + n)\kappa + 12n^2$ |
| $\mathsf{VC}_r$ (pub. ver. + privacy) | $(n^2 + 2n)\kappa$ | $4n^2\kappa$ | $(5n^2 + 1)\kappa$ |

**Table 3.** Storage and communication in our and mostly closely related schemes.

The timings of elementary cryptographic operations can be inferred from the benchmarks in [14]. For groups that admit an asymmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which offer faster performance than groups that admit a symmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ and are used in this work (as detailed later in the paper) and can be used in the solution of [18], the timings are as follows: for 128-bit security, a modulo exponentiation in $\mathbb{G}_1$ or $\mathbb{G}_2$ can be performed in 0.1–0.4ms on a single core of a conventional 2.4GHz desktop machine, a modulo exponentiation in $\mathbb{G}_T$ in 0.4–0.9ms, and the pairing operation in 2.1–2.3ms. The cost of modulo multiplications is at least two orders of magnitude smaller than that of performing modulo exponentiations.

The storage and communication requirements of our constructions and those of [18] are listed in Table 3. In the table, the security parameter $\kappa$ denotes the bitlength of group elements. The client's storage is computed as the amount of information the client needs to maintain in order to be able to verify the result (i.e., the size of the key). Then the task delegator and each task verifier will require additional storage for their respective input and output matrices. The server's storage corresponds to the amount of storage the server needs to maintain in order to carry out the task. Lastly, communication corresponds to both sending the task to the server and returning the result and the proof of computation to the client.

**Rational Computation**. In recent years, game theory has been used in cryptographic research [17, 24] to develop a new adversarial model – a rational adversary who is no longer treated as arbitrarily malicious, but who is motivated by some utility function with the sole purpose of maximizing its utility. It is known that under this model protocols can be designed with better efficiency than that of traditional counterparts [19]. As our problem deals with verifiable computation, we are interested in rational proof systems that have been recently studied in [6, 7, 23]. The merits of rational proof systems are that they allow for extremely low communication and verification time [6, 7] and can achieve single-round proofs if the prover is computationally bounded [23]. The basic idea of this line of work is that the prover will send the result of computation to the verifier who will compute the corresponding reward based on the "quality" of prover's result, and the reward will be maximized only if the result is correct. The publications focus on general complexity classes such as uniform $TC^0$ (polynomial-time, constant-depth threshold circuits) and decision problems in $P^{||NP}$ (polynomial time with access to parallel queries to an NP oracle), while in our case, we target specific matrix computation with the goal of achieving even better efficiency.

## 3 Background and Definitions

### 3.1 Basic Definitions

Throughout this work we use notation $x \overset{R}{\leftarrow} S$ to denote that $x$ is chosen uniformly at random from the set $S$. A function $\epsilon(n)$ is said to be negligible if for sufficiently large $n$ its value is smaller than the inverse of any polynomial $poly(n)$. Let $F$ be a family of functions and $\mathsf{Dom}(f)$ denote the domain of function $f \in F$. Also, let $\kappa$ denote a security parameter. We use $H : \{0,1\}^* \to \{0,1\}^{\ell_1(\kappa)}$ to denote a collision resistant hash function that takes as input a string $x$ and outputs an $\ell_1(\kappa)$-bit hash $y$. We also use notation $||$ to denote string concatenation. For matrix $A$, notation $A_{ij}$ refers to the element of $A$ at row $i$ and column $j$. We use notation $\mathsf{PRF}$ to refer to a pseudo-random function family defined as follows.

**Definition 1.** *Let $F : \{0,1\}^\kappa \times \{0,1\}^{\ell_2(\kappa)} \to \{0,1\}^{\ell_2(\kappa)}$ be a family of functions. For $k \in \{0,1\}^\kappa$, the function $f_k : \{0,1\}^{\ell_2(\kappa)} \to \{0,1\}^{\ell_2(\kappa)}$ is defined as $F_k(x) = F(k, x)$. $F$ is said to be a family of pseudo-random functions (PRF) if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$ with oracle access to a function $F_k$ and all sufficiently large $\kappa$, $|\Pr[\mathcal{A}^{F_k}(1^\kappa) - \Pr[\mathcal{A}^R(1^\kappa)]|$ is negligible in $\kappa$, where $k \overset{R}{\leftarrow} \{0,1\}^\kappa$ and $R$ is a function chosen at random from all possible functions mapping $\ell_2(\kappa)$-bit inputs to $\ell_2(\kappa)$-bit outputs.*

**Definition 2 (Bilinear map).** *A one-way function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map if the following conditions hold:*

- *(Efficient)* $\mathbb{G}_1$, $\mathbb{G}_2$, *and* $\mathbb{G}_T$ *are groups of the same prime order* $p$ *and there exists an efficient algorithm for computing* $e$.
- *(Bilinear) For all* $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, *and* $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
- *(Non-degenerate) If* $g_1$ *generates* $\mathbb{G}_1$ *and* $g_2$ *generates* $\mathbb{G}_2$, *then* $e(g_1, g_2)$ *generates* $\mathbb{G}_T$.

Throughout this work, we assume there exists a trusted setup algorithm *Set* that, on input a security parameter $1^\kappa$, outputs the setup for groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ of prime order $p$ that have a bilinear map $e$, and $e(g_1, g_2)$ generates $\mathbb{G}_T$ of order $p$. That is, $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow Set(1^\kappa)$.

## 3.2 Computational Assumptions

The first computational assumption used in this work is the Multiple Decisional Diffie-Hellman Assumption ($m$-M-DDH) [12], which can be stated as follows:

**Definition 3 ($m$-M-DDH assumption).** *Let* $\mathbb{G}$ *be a group of prime order* $p$, $g \in \mathbb{G}$ *is its generator, and* $m \geq 2$. *Also let* $D = (g^{x_1}, \ldots, g^{x_m}, \{g^{x_i x_j}\}_{1 \leq i < j \leq m})$ *for random* $x_1, \ldots, x_m \in \mathbb{Z}_p$, *and define random tuple as* $D_{rand} = (g_1, \ldots, g_m, \{g_{ij}\}_{1 \leq i < j \leq m})$ *in* $\mathbb{G}$. *Adversary* $\mathcal{A}$, *whose task is to distinguish an M-DDH tuple from a random tuple, outputs a bit. We define the advantage of adversary* $\mathcal{A}$ *in solving the M-DDH problem as* $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa) = |\Pr[\mathcal{A}(g, p, m, D) = 1] - \Pr[\mathcal{A}(g, p, m, D_{rand}) = 1]|$. *The M-DDH assumption holds if for every PPT algorithm* $\mathcal{A}$, $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa)$ *is negligible.*

Some of our schemes are built using subgroups of elliptic curves with pairings where the decisional Diffie-Hellman (DDH) problem is hard. The use of DDH-hard pairing groups requires the External Diffie-Hellman (XDH) assumption [10].

**Definition 4 (XDH assumption).** *Let* $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow Set(1^\kappa)$. *We define the advantage of adversary* $\mathcal{A}$ *in solving the DDH problem in* $\mathbb{G}_1$ *as* $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{XDH}}(\kappa) = |\Pr[\mathcal{A}(p, g_1, g_2, g_1^a, g_1^b, g_1^{ab}) = 1] - \Pr[\mathcal{A}(p, g_1, g_2, g_1^a, g_1^b, g_1^c) = 1]|$, *where* $a, b, c \overset{R}{\leftarrow} \mathbb{Z}_p$. *We say that the XDH assumption holds if for every PPT algorithm* $\mathcal{A}$ $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{XDH}}(\kappa)$ *is negligible.*

The XDH assumption implies that there is no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$. This assumption is also necessary for the M-DDH assumption to hold in groups that admit a bilinear map.

## 3.3 Verifiable Computation

A verifiable computation scheme $\mathsf{VC}$ is a 4-tuple of polynomial-time algorithms ($\mathsf{Setup}$, $\mathsf{ProbGen}$, $\mathsf{Compute}$, $\mathsf{Verify}$) that allows a user to outsource the computation of function $f \in F$ to an untrusted worker. $\mathsf{VC}$ is defined as follows:

$\mathsf{Setup}(1^\kappa, f) \rightarrow \mathsf{params}$: On input a security parameter $\kappa$ and function $f$ to be outsourced, it produces public parameters $\mathsf{params}$.

$\mathsf{ProbGen}(x, \mathsf{params}) \to (\mathsf{SK}_x, \mathsf{EK}_x, \sigma_x)$: Given an input $x \in \mathsf{Dom}(f)$, this algorithm is run by the delegator to produce a secret key $\mathsf{SK}_x$ associated with the problem instance for computation outsourcing and output verification, an evaluation key $\mathsf{EK}_x$ given to the worker to carry out the outsourced computation, and an encoding $\sigma_x$ of input $x$.

$\mathsf{Compute}(\mathsf{EK}_x, \sigma_x) \to \sigma_y$: On an encoded input $\sigma_x$ and $\mathsf{EK}_x$, the worker runs the algorithm to produce an encoded outcome $\sigma_y$, where $y = f(x)$.

$\mathsf{Verify}(\mathsf{SK}_x, \sigma_y) \to y \cup \bot$: Given an encoded output $\sigma_y$ and the secret key $\mathsf{SK}_x$, this algorithm outputs $y$ or an error $\bot$ upon result verification.

The *correctness* requirement is such that the values produced by the algorithms will allow any honest worker who faithfully executes $\mathsf{Compute}$ to pass verification of the output it produces. More formally, for any $f \in F$, any $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, f)$, and any $x \in \mathsf{Dom}(f)$, if $(\mathsf{SK}_x, \mathsf{EK}_x, \sigma_x) \leftarrow \mathsf{ProbGen}(x, \mathsf{params})$, $\sigma_y \leftarrow \mathsf{Compute}(\mathsf{EK}_x, \sigma_x)$, and $y \leftarrow \mathsf{Verify}(\mathsf{SK}_x, \sigma_y)$, then $\Pr[y = f(x)] = 1$.

To formulate *security* of a verifiable computation scheme, we define an interactive security experiment described next. In the experiment, the adversary $\mathcal{A}$ is allowed to query $\mathsf{ProbGen}$ algorithms on inputs of its choice $x_i$ and obtain the corresponding evaluation key $\mathsf{EK}_{x_i}$ and input encoding $\sigma_{x_i}$. In the private key setting, it is also granted oracle access to $\mathsf{Verify}$ algorithm, where $\mathcal{O}_{\mathsf{Verify}}(x_i, \sigma_y)$ runs $y \leftarrow \mathsf{Verify}(\mathsf{SK}_i, \sigma_y)$ and returns $y$. Eventually, $\mathcal{A}$ outputs the input $x^*$ on which it would like to be challenged, obtains evaluation key $\mathsf{EK}_{x^*}$ and encoding $\sigma_{x^*}$, and produces output encoding $\sigma'_y$. The adversary succeeds if the output is different from $f(x^*)$ and the verification algorithm does not output an error $\bot$. Note that this definition captures full adaptive security as opposed to weaker selective security where the adversary is required to commit to the challenge input $x^*$ in the beginning of the game.

```
Experiment $\mathbf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\mathsf{VC}, f, \kappa)$
    $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, f)$
    for $i = 1$ to $q$ do
        $x_i \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Verify}}(\cdot,\cdot)}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_{i-1}}, \mathsf{EK}_{i-1})$
        $(\mathsf{SK}_i, \mathsf{EK}_i, \sigma_{x_i}) \leftarrow \mathsf{ProbGen}(x_i, \mathsf{params})$
    $x^* \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q)$
    $(\mathsf{SK}_{x^*}, \mathsf{EK}_{x^*}, \sigma_{x^*}) \leftarrow \mathsf{ProbGen}(x^*, \mathsf{params})$
    $\sigma'_y \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q, \sigma_{x^*}, \mathsf{EK}_{x^*})$
    $y' \leftarrow \mathsf{Verify}(\mathsf{SK}_{x^*}, \sigma'_y)$
    if $y' \neq \bot$ and $y' \neq f(x^*)$ return 1
    else return 0
```

For any $\kappa \in \mathbb{N}$ and any function $f \in F$, we define the advantage of an adversary $\mathcal{A}$ making at most $q = poly(\kappa)$ queries in the above security game against $\mathsf{VC}$ as $\mathbf{Adv}^{\mathsf{Ver}}_{\mathcal{A}}(\mathsf{VC}, f, q, \kappa) = \Pr[\mathbf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\mathsf{VC}, f, \kappa) = 1]$.

**Definition 5.** *A verifiable computation scheme* $\mathsf{VC}$ *is secure if for any PPT adversary* $\mathcal{A}$*, any* $\kappa$*, and any* $f \in F$*,* $\boldsymbol{Adv}^{\mathsf{Ver}}_{\mathcal{A}}(\mathsf{VC}, f, q, \kappa)$ *is negligible in* $\kappa$*.*

In this work we consider two types of adversaries: the first type is the traditional adversary that can arbitrarily deviate from the prescribed computation as defined by $\mathsf{Compute}$ functionality. We denote this type of adversary as malicious. While the malicious adversary model leads to strong security guarantees, it has been criticized as overly pessimistic due to neglecting the incentive that could potentially cause computational entities to deviate from the prescribed behavior. We therefore consider the second type of adversary which we denote as rational. A rational adversary is neither honest nor malicious, but only interested in maximizing its reward attained during computation. The rationale behind including this type of adversary is that it allows us to design more efficient solutions if the server can be assumed not to intentionally corrupt the result. Next, we formally define the rational adversary model in verifiable computation, which was initially proposed in [6] and later refined for rational argument systems in [7, 23].

**Definition 6.** *A function $f$ admits a rational argument with security parameter $\kappa$ if there exists a protocol $(P, V)$ and a randomized reward function* $\mathsf{reward}\colon \{0,1\}^\star \to \mathbb{R}_{\geq 0}$ *such that for any prover $\hat{P}$ of size $\leq 2^{\kappa(|x|)}$ and input $x \in \{0,1\}^\star$, the following three properties hold:*

- $\Pr[\mathsf{output}(P, V)(x) = f(x)] = 1$.
- *There exists a negligible function $\mu(\cdot)$ such that* $E[\mathsf{reward}((P, V)(x))] + \mu(|x|) \geq E[\mathsf{reward}((\hat{P}, V)(x))]$.
- *If there exists a polynomial $p(\cdot)$ such that* $\Pr[\mathsf{output}((\hat{P}, V)(x)) \neq f(x)] \geq p(|x|)^{-1}$, *then there exists a polynomial $q(\cdot)$ such that* $E[\mathsf{reward}((P, V)(x))] \geq E[\mathsf{reward}((\hat{P}, V)(x))] + q(|x|)^{-1}$.

The first property refers to completeness of the protocol, which says prover $P$ is able to return a correct answer $f(x)$ by following the prescribed protocol. The second property ensures that by deviating from the prescribed protocol in a computationally bounded manner, a dishonest prover $\hat{P}$ will achieve at most negligibly larger gain than a faithful prover $P$. The last property guarantees if $\hat{P}$ does not report a correct answer $f(x)$ with a noticeable probability, he has to bear a noticeable utility loss. A rational argument system ensures that a rational adversary will maximize the reward if and only if he honestly follows the protocol to report the correct answer. Therefore, to prove security of a protocol, we need to show that it conforms to Definition 6 under reasonable assumptions on cost and utility, which we formulate in the server-client setting as follows:

**Assumption 1**
- *For each outsourced task, both the monetary reward the client compensates and the computational cost the server bears are polynomial to the size of the input.*
- *As the server aims to make profits by devoting his resources to clients' specific tasks, the monetary reward he gains from a client should be larger than the cost it bears, which also conforms to the business practice for cloud service.*
- *In the event of any inconsistency between the answers the server returns and the answers the client expects, the server will not receive any reward or even undertake utility loss resulted from the violation of Service Level Agreement.*

All additional definitions for data protection and public verifiability are omitted due to space constraints, but can be found in the full version of this work [34].

## 4 Matrix Multiplication For Malicious Adversary

**Problem Formulation.** The delegator would like to multiply matrices $A$ and $B$ of dimensions $n_1 \times n_2$ and $n_2 \times n_3$, respectively. It is assumed that the elements of $A$ and $B$ are not sensitive and do not require protection. In our solution, the delegator's work is linear in the size of the input and output, which is optimal.

Our first scheme aims to defend against malicious adversary who tampers with the computation or its results regardless of costs and attempts to pass verification. Also, this basic construction does not achieve public verifiability: only the entity who possesses the secret key is able to attest correctness of returned result. In section 6 we extend the scheme to support public verifiability that allows any entity with access to public key assess the validity of computation results. For notational simplicity, we use $\mathsf{VC}_m$ to denote this scheme.

**Description of the Scheme.** The main idea used in our solution is that the delegator encodes matrix $A$ into matrix $X$ and matrix $B$ into matrix $Y$. The delegator sends matrices $\{A, B, X, Y\}$ to the server who computes $C = A \times B$ and $D = X \times Y$. The client then verifies correctness of $C$ by checking a secret relationship between the elements of $C$ and $D$.

In more detail, the secret relationship is formed by generating secret random group elements $R_{ij}$ such that $D_{ij} = R_{ij}^{C_{ij}}$, which are of the form $g_T^{r_i c_j}$, where $\{r_i\}_{i=1}^{n_1}$ and $\{c_j\}_{j=1}^{n_3}$ are two vectors of random elements. Moreover, in order to satisfy the relationship, we need to embed $r_i$ and $c_j$ into each $D_{ij}$, and this is realized by forming $X_{ij}$ and $Y_{ij}$ to be of the form $g_1^{r_i A_{ij}}$ and $g_2^{c_j B_{ij}}$, respectively. We aim to rely on the M-DDH assumption to show that, given $g_1^{r_i}$ and $g_2^{c_j}$, $g_T^{r_i c_j}$ is indistinguishable from a random value, which is however difficult due to the existence of bilinear pairing operation. To remedy the problem, we completely hide all information about $g_2^{c_j}$'s from the server. As a result, if we rely on the XDH assumption, $(g_T^{r_1}, \ldots, g_T^{r_{n_1}}, \{g_T^{r_i c_j}\}_{1 \le i \le n_1, 1 \le j \le n_3})$ is a partial $(n_1 + n_3)$-M-DDH tuple and the adversary can have only a negligible advantage in distinguishing the ${g_T}^{r_i c_j}$'s from random elements of the group. The hiding is achieved by further masking each $Y_{ij}$ by a random value $T_{ij}$, which should be also in a special form; otherwise, the client will have to compute $X \times T$ to satisfy the relationship, which is the exact workload the client wants to avoid. Therefore, $T_{ij}$'s are formed as $g_2^{w_j d_i}$ using two random vectors $\{d_i\}_{i=1}^{n_2}$ and $\{w_j\}_{j=1}^{n_3}$, and the client only needs to perform $O(n_1 n_3)$ work to compute $X \times T$.

When the server returns $C$ and $\sum_i \sum_j D_{ij}$, the delegator uses $\{r_i c_j\}_{1 \le i \le n_1, 1 \le j \le n_3}$ and information about the product $X \times T$ to verify that the sum of the elements in $C$ after proper randomization matches $\sum_i \sum_j D_{ij}$. If the verification succeeds, the delegator uses $C$ as the correct output. The details are given in Figure 1.

The complexity of $\mathsf{ProbGen}$ run by the delegator is dominated by computing key $vk$ and matrices $X$ and $Y$, and is therefore $O(n_1 n_2 + n_2 n_3)$. $\mathsf{Compute}$ involves the execution of two matrix multiplications resulting in complexity $O(n_1 n_2 n_3)$. $\mathsf{Verify}$ consists of ensuring the validity of $C$ by checking its elements against $s$ that $\mathsf{Compute}$ produces and has complexity $O(n_1 n_3)$, i.e., linear in the size of

$\mathsf{Setup}(1^\kappa, f)$: Given $f$ that indicates matrix multiplication, using the security parameter $\kappa$ run $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow Set(1^\kappa)$ and set Set $\mathsf{params} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T = e(g_1, g_2), f)$. Matrix elements should be representable as values in $\mathbb{Z}_p$.

$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$: On input two matrices $A$ and $B$ of respective dimensions $n_1 \times n_2$ and $n_2 \times n_3$, perform:

1. Choose $r_i \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \leq i \leq n_1$, $d_j \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \leq j \leq n_2$, and $c_k, w_k \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \leq k \leq n_3$.
2. Compute $t_i = \sum_{k=1}^{n_2} A_{ik} d_k$ for $1 \leq i \leq n_1$, $f = \sum_{j=1}^{n_3} w_j$, and $vk_i = r_i t_i f$ for $1 \leq i \leq n_1$.
3. Compute $X_{ij} = g_1^{r_i A_{ij}}$ for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$.
4. Compute $Y_{ij} = g_2^{c_j B_{ij} + w_j d_i}$ for $1 \leq i \leq n_2$ and $1 \leq j \leq n_3$.
5. Set $\mathsf{SK}_x = (\{c_i\}_{i=1}^{n_3}, \{vk_i\}_{i=1}^{n_1}, \{r_i\}_{i=1}^{n_1})$, $\mathsf{EK}_x = \mathsf{params}$, and $\sigma_x = (A, B, X, Y)$.

$\mathsf{Compute}(\mathsf{EK}_x, \sigma_x = (A, B, X, Y))$: Given $\sigma_x$, execute:

1. Compute $C = A \times B$.
2. Compute $s = \prod_{i=1}^{n_1} \prod_{j=1}^{n_3} \prod_{k=1}^{n_2} e(X_{ik}, Y_{kj})$.
3. Set $\sigma_y = (C, s)$.

$\mathsf{Verify}(\mathsf{SK}_x = (\{c_i\}_{i=1}^{n_3}, \{vk_i\}_{i=1}^{n_1}, \{r_i\}_{i=1}^{n_1}), \sigma_y = (C, s))$: If $g_T^{\sum_{i=1}^{n_1} r_i \sum_{j=1}^{n_3} c_j C_{ij} + vk_i} = s$, output $C$; otherwise, output $\perp$.

**Fig. 1.** Description of the core scheme $\mathsf{VC}_m$ in presence of malicious adversaries.

the output. The value of $s$ is computed in such a way that a malicious adversary is unable to construct an incorrect $s$ that passes the verification test.

Security of the scheme can be stated as follows:

**Theorem 1.** *Assuming that the M-DDH and XDH problems are hard, the verifiable computation scheme* $\mathsf{VC}_m$ *is secure according to Definition 5 in presence of malicious adversaries.*

The proof and correctness analysis of this construction can be found in Appendix A.

## 5 Matrix Multiplication For Rational Adversary

Our next construction aims at defending against a rational adversary. Because a rational adversary behaves in the most profitable manner by considering both the compensation paid by the client and the cost endured during the computation, it would be to the adversary's advantage to honestly report all computed results to obtain compensation for the work (rather than report a bogus result that could be detected with overwhelming probability and hence yield a lower reward).

In our solution against rational adversaries $\mathsf{VC}_r$, we achieve two features: (i) to force a rational adversary who wishes to maximize its profits to conform to the prescribed protocol and (ii) in case of faulty computation, to pinpoint location of faulty cells by both the server and the client. Realizing both features is achieved by requiring the client to perform only work sublinear in the size of the matrices at the time of computation verification. For the ease of presentation, we describe our solution in a modular manner, where the first scheme support only the first feature and the second presented scheme enhances it with the second feature.

### 5.1 Description of the Base Scheme

The main idea behind $\mathsf{VC}_r$ is similar to that of $\mathsf{VC}_m$: as before, the delegator encodes $A$ into $X$ and $B$ into $Y$ and asks the server to compute products $A \times B$ and $X \times Y$. Similar to $\mathsf{VC}_m$, correctness of $A \times B$ is verified by checking a secret relationship between the two matrix products. However, unlike $\mathsf{VC}_m$, where the delegator performs the verification itself, in $\mathsf{VC}_r$, the delegator further outsources the verification task to the server and only performs one string comparison to confirm correctness of the verification process. The saving in the verification cost comes with slightly increased work during problem generation, but this work is still linear in the size of input and output. This scheme can be suitable in the setting with three entities (besides the server) such as a doctor who delegates problem generation to lab assistants and patients who verifies the result of the computation returned by the server. The entity performing problem generation (i.e., lab assistant in the above example) is willing to put in additional (one-time) work to benefit routine operations (i.e., verification) by end users (patients).

In our solution, the delegator, as before, produces $X$ as a randomized version of $A$ with its elements of the form $g_1^{r_i A_{ij}}$ and $Y$ as a randomized version of $B$ with its elements of the form $g_2^{c_j B_{ij} + w_j d_i}$. The delegator also releases a new matrix $Z$ formed as $g_T^{r_i c_j}$ to aid the server in producing a proof of correct computation. Unlike $\mathsf{VC}_m$, security of which relies on the secrecy of $g_T^{r_i c_j}$, in $\mathsf{VC}_r$, we need to incorporate additional secret information in the solution to guarantee security despite public exposure of matrix $Z$. If we treat $Y$ as $\hat{B} + T$, where $\hat{B} = g_2^{c_j B_{ij}}$ and $T = g_2^{w_j d_i}$, we want to use $X \times T$ as secret information. In order to do that, the cells of $X \times T$ should be indistinguishable from random group elements, and can be produced by the server only if it follows the prescribed protocol. Towards the goal, we incorporate additional randomization into $X$ and $\hat{B}$ by representing their elements as $g_1^{r_i/m_j A_{ij}}$ and $g_2^{c_j m_i B_{ij}}$, respectively, using another random vector $\{m_i\}_{i=1}^{n_2}$. The verification key is set to be the result of hashing of all cells of $X \times T$. Therefore, to pass verification, the server has to recover all elements of $X \times T$ correctly. Notice that because the server is unable to separate $\hat{B}$ from $T$ in $Y$ and thus compute $X \times T$ by unintended means, the server is forced to compute $X \times Y$ and $A \times B$, determine $X \times \hat{B}$ from $A \times B$ with the help of $Z$, and remove $X \times \hat{B}$ from $X \times Y$ to recover the key. The scheme is given in Figure 2.

The complexity of $\mathsf{ProbGen}$ is $O(n_1 n_2 + n_1 n_3 + n_2 n_3)$, i.e., linear in the size of the input. The complexity of $\mathsf{Compute}$ is dominated by two matrix multiplications resulting in $O(n_1 n_2 n_3)$ time. Lastly, the $\mathsf{Verify}$ algorithm performs a single string comparison of complexity $O(1)$ and outputs the matrix of size $n_1 \times n_3$.

Security of this solution holds only when each cell $A_{ij}$ of matrix $A$ takes a non-zero value. For that reason, we next describe a mechanism for encoding an arbitrary matrix $M$ into an equivalent matrix $M'$ that contains only non-zero values and decoding the result after $M'$ is used in the computation.

$\mathsf{Setup}(1^\kappa, f)$: The same as in $\mathsf{VC}_m$.

$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$: On input two matrices $A$ and $B$ of respective dimensions $n_1 \times n_2$ and $n_2 \times n_3$, perform:

1. Choose $r_i \xleftarrow{R} \mathbb{Z}_p^*$ for $1 \le i \le n_1$, $d_j, m_j \xleftarrow{R} \mathbb{Z}_p^*$ for $1 \le j \le n_2$, and $c_k, w_k \xleftarrow{R} \mathbb{Z}_p^*$ for $1 \le k \le n_3$.
2. Compute $X_{ij} = g_1^{r_i/m_j A_{ij}}$ for $1 \le i \le n_1$ and $1 \le j \le n_2$.
3. Compute $Y_{ij} = g_2^{c_j m_i B_{ij} + w_j d_i}$ for $1 \le i \le n_2$ and $1 \le j \le n_3$.
4. Compute $Z_{ij} = g_T^{c_j r_i}$ for $1 \le i \le n_1$ and $1 \le j \le n_3$.
5. Compute $t_i = \sum_{k=1}^{n_2} A_{ik} d_k / m_k$ for $1 \le i \le n_1$, and $vk_{ij} = t_i r_i w_j$ for $1 \le i \le n_1$ and $1 \le j \le n_3$.
6. Compute $sk_{ij} = g_T^{vk_{ij}}$ for $1 \le i \le n_1$ and $1 \le j \le n_3$, set $sk_j = H(sk_{1j}||sk_{2j}||\ldots||sk_{n_1 j})$ for $1 \le j \le n_3$, and $sk = H(sk_1||sk_2||\ldots||sk_{n_3})$.
7. Set $\mathsf{SK}_x = sk$, $\mathsf{EK}_x = (\mathsf{params}, Z)$, and $\sigma_x = (A, B, X, Y)$.

$\mathsf{Compute}(\mathsf{EK}_x = (\mathsf{params}, Z), \sigma_x = (A, B, X, Y))$: Execute the following steps:

1. Compute (i) $V_{ij}^{(1)} = \sum_{k=1}^{n_2} A_{ik} B_{kj}$, (ii) $V_{ij}^{(2)} = \prod_{k=1}^{n_2} e(X_{ik}, Y_{kj}) = {g_T}^{\sum_{k=1}^{n_2} A_{ik}(c_j r_i B_{kj} + w_j d_k r_i / m_k)}$, and (iii) $\varDelta_{ij} = V_{ij}^{(2)} / {Z_{ij}}^{V_{ij}^{(1)}}$ for $1 \le j \le n_3$ and $1 \le i \le n_1$.
2. Compute $\hat{sk}_j = H(\varDelta_{1j}||\varDelta_{2j}||\ldots||\varDelta_{n_1 j})$ and $\hat{sk} = H(\hat{sk}_1||\hat{sk}_2||\ldots||\hat{sk}_{n_3})$.
3. Set $\sigma_y = (V^{(1)}, \hat{sk})$.

$\mathsf{Verify}(\mathsf{SK}_x = sk, \sigma_y = (V^{(1)}, \hat{sk}))$: Verify whether $\hat{sk} = sk$. If the check succeeds, output $V^{(1)}$; otherwise, output $\perp$.

**Fig. 2.** Description of the core scheme $\mathsf{VC}_r$ in presence of rational adversaries.

Matrix encoding: Given $M$ of dimensions $n_1 \times n_2$, choose any value $\ell$ such that $A_{ij} + \ell \neq 0$ for $1 \le i \le n_1$ and $1 \le j \le n_2$. To form $M'$, add $\ell$ to each element of $M$, i.e., $M'_{ij} = M_{ij} + \ell$, and store $\ell$ for future reference.

Matrix decoding: Let $C' = M' \times N'$, where $M'$ ($N'$) is an encoded version of matrix $M$ (resp., $N$) using value $\ell_1$ (resp., $\ell_2$) and has dimensions $n_1 \times n_2$ (resp., $n_2 \times n_3$). Observe that each element $C'_{ij} = \sum_{k=1}^{n_2} (M_{ik} + \ell_1)(N_{kj} + \ell_2)$. To recover $C = M \times N$, compute the offset $\varDelta_{ij}$ for each element, which equals to $\ell_2 \sum_{k=1}^{n_2} M_{ik} + \ell_1 \sum_{k=1}^{n_2} N_{kj} + n_2 \ell_1 \ell_2$, and set $C_{ij} = C'_{ij} - \varDelta_{ij}$. Note that the value $\ell_2 \sum_{k=1}^{n_2} A_{ik}$ ($\ell_1 \sum_{k=1}^{n_2} B_{kj}$) is the same for all elements in a single row of matrix $M$ (resp., single column of matrix $N$). This means that we only need to compute that value for $n_1$ rows of matrix $M$ (resp., $n_3$ columns of matrix $N$). The overall complexity of computing all offsets is therefore $O(n_1 n_2 + n_2 n_3)$ and the complexity of computing $C$ from $C'$ is $O(n_1 n_3)$.
The decoding computation is simplified when only one of the matrices used in the product was encoded to eliminate zero entries (as in $\mathsf{VC}_r$). In that case, the offset becomes $\varDelta_{ij} = \ell_1 \sum_{k=1}^{n_2} N_{kj}$ assuming that $M$ was the encoded matrix, and the overall complexity of decoding is $O(n_1 n_3 + n_2 n_3)$.

Security of our $\mathsf{VC}_r$ scheme can be stated as follows:

**Theorem 2.** *If $H$ is a collision-resistant hash function, the M-DDH and XDH assumptions hold, and all elements of $A$ are non-zero, a server that deviates from the protocol can only pass verification with a probability negligible in $\kappa$.*

**Assumption 2** *If in $\mathsf{VC}_r$ the server returns correct $\hat{sk} = sk$ but incorrect $V^{(1)} \neq C$, the error will be detected by the client with a non-negligible probability.*

This assumption is crucial to our result and can be realized by outsourcing a small (but non-negligible) fraction $0 < \rho < 1$ of all tasks to a second independent server. For example, for each task the client chooses random $v \in [0, 1]$, and if $v \leq \rho$, the client uses two (non-colluding) servers for independently outsourcing the task and compares the returned products $C$ afterwards. If the servers are non-colluding and at least one returned $C$ is incorrect, the client will be able to detect misbehavior with non-negligible probability. This implies that if the server performed the work to pass the verification test, it will be incentivized to return correct $C$.

**Theorem 3.** *If Assumptions 1 and 2 and assumptions of Theorem 2 hold, $\mathsf{VC}_r$ is secure in presence of rational adversaries according to Definition 6.*

Due to space constraints, security proofs and correctness analysis are available only in the full version [34]. The above means that following the protocol and producing correct matrix products $A \times B$ and $X \times Y$ is the most profitable strategy for a rational adversary.

## 5.2 Description of the Enhanced Scheme

In this section, we propose an enhanced scheme that supports chaining and allows honest parties to pinpoint faulty cells in case of computation corruption or intentional deviation from the prescribed computation using a single mechanism. That is, recall that this feature makes it difficult for a dishonest server to continue with the next stage of the computation if it was not carried out correctly at the current stage and allows the client to efficiently identify the first stage at which a fault occurred. It also allows the cloud itself to detect a problem with the computation (in case of compromise or malware infection). The basic idea behind the solution is that the client divides the entire computation into $n_3$ sub-computations. The keys $sk_i$ are formed as before, but now the $i$th key is used to encode the inputs of the $(i + 1)$th sub-computation. The server is able to recover the $(i + 1)$th sub-key only if it executes sub-computations $1, \ldots, i$ correctly. Upon computation completion, the verifier receives the last key from the server and examines its correctness. If the verifier notices a discrepancy between the returned key and its expected value, he will ask the server to return all the keys generated throughout the computation. The verifier then applies a procedure similar to binary search to locate the first incorrect key, which corresponds to the first sub-computation that has been executed incorrectly. This operation can be implemented in $O(\log n)$ steps when the number of sub-computations is $n$. The server will also be able to examine correctness of the first $i$ sub-computations that have been executed so far by verifying that the inputs of $(i + 1)$th sub-computation could be decoded correctly using $sk_i$. If this check fails, this serves as a notice of the existence of faulty cells and the server can suspend the computation to identify faulty sells among the computed cells.

The detail of our solution can be described as follows: The client generates five matrices $\{A, B, X, Y, Z\}$ and forms keys $sk_i$ as in the base scheme. Now the computation of the $i$th column of matrices $A \times B$ and $X \times Y$ is considered to be the $i$th sub-computation. The client then blinds each element of the $(i+1)$th column of matrices $B$ and $Y$ by XORing them with a pseudo-random string. This string is produced using $sk_i$ as the secret key to a pseudo-random function $\mathsf{PRF}$, which is evaluated on the cell's row and column indices together with a unique identifier for each matrix to guarantee uniqueness of the input/output.

In order to remove blinding and recover the next sub-computation, the server needs to produce correct $sk_i$ as before (by computing the $i$th column of two matrix products) and evaluate the PRF on that key to reproduce the random mask. This will allow the server to recover the inputs for the $(i+1)$th column of matrix $B$ and $Y$, i.e., the $(i+1)$th sub-computation, and continue the computation. We make the size of the pseudo-random output of $\mathsf{PRF}$ longer than the size of matrix elements they mask so that the remaining bits can be used to verify that the input to the $(i+1)$th sub-computation was decoded correctly. That is, we append a zero string of a predefined size to each input before encoding, and the server can use it to verify that decoding was successful (and if it was not, investigate the reason for the failure). At the end of the computation, the server recovers and returns the last key $sk_{n_3}$, which the client compares to the expected key and accepts the output if the verification succeeds. The scheme is given in Figure 3.

The complexities of $\mathsf{ProbGen}$ and $\mathsf{Compute}$ are the same as in $\mathsf{VC}_r$, i.e., $O(n_1n_2 + n_1n_3 + n_2n_3)$ and $O(n_1n_2n_3)$, respectively. $\mathsf{Verify}$ now performs one string comparison of complexity $O(1)$ and outputs a matrix of size $n_1 \times n_3$ in case of no errors. Otherwise, the client retrieves $n_3$ keys and additionally performs $O(\log(n_3))$ work. Security is stated below and the proof can be found in [34].

**Theorem 4.** *If $H$ is a collision-resistant hash function, $\mathsf{PRF}$ is a pseudo-random function, the M-DDH and XDH assumptions hold, and all elements of $A$ are non-zero, the server that deviates from the correct protocol can only pass verification with a probability negligible in $\kappa$.*

As the corollary, we have that if the result does not verify, the client can identify the first faulty sub-computation using $O(\log(n_3))$ string comparisons.

## 6 Extensions

In this section, we sketch how to extend our schemes to incorporate privacy protection of input/output matrices and public verifiability. The details of the constructions and their analysis are available in [34].

**Matrix Privacy.** This property must ensure that the server is unable to learn any information about $A$ and $B$ and their product $A \times B$. We thus modify both schemes to achieve this goal, while still preserving computation verifiability.

*Malicious Setting.* To protect matrices $A$ and $B$ in $\sigma_x$, our solution encodes them using a homomorphic encryption scheme (e.g., BGN encryption [11]) that

```
$\mathsf{Setup}(1^\kappa, f)$: The same as in $\mathsf{VC}_r$.
$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$: Given matrices $A$ and $B$, perform:
    1. Compute $X$, $Y$, $Z$, and $\{sk_i\}_{i=1}^{n_3}$ as in $\mathsf{VC}_r$.
    2. Set $\hat{B}_{i1} = B_{i1}$ and $\hat{Y}_{i1} = Y_{i1}$ for $1 \leq i \leq n_2$; and also set
        • $\hat{B}_{ij} = \mathsf{PRF}_{sk_{j-1}}(i||j||0) \oplus (B_{ij}||0^\lambda)$
        • $\hat{Y}_{ij} = \mathsf{PRF}_{sk_{j-1}}(i||j||1) \oplus (Y_{ij}||0^\lambda)$
       for $1 \leq i \leq n_2$ and $2 \leq j \leq n_3$, where $\lambda$ is a correctness parameter.
    3. Set $\mathsf{SK}_x = \{sk_i\}_{i=1}^{n_3}$, $\mathsf{EK}_x = (\mathsf{params}, Z)$, and $\sigma_x = (A, \hat{B}, X, \hat{Y})$.
$\mathsf{Compute}(\mathsf{EK}_x = (\mathsf{params}, Z), \sigma_x = (A, \hat{B}, X, \hat{Y}))$: Execute steps:
    1. Set $U_{i1}^{(1)} = \hat{B}_{i1}$ and $U_{i1}^{(2)} = \hat{Y}_{i1}$ for $1 \leq i \leq n_2$.
    2. For $j = 1, \ldots, n_3$ do:
       (a) For $1 \leq i \leq n_1$, compute $V_{ij}^{(1)} = \sum_{k=1}^{n_2} A_{ik} U_{kj}^{(1)}$ and $V_{ij}^{(2)} = \prod_{k=1}^{n_2} e(X_{ik}, U_{kj}^{(2)})$.
       (b) Let $\Delta_{ij} = V_{ij}^{(2)} / Z_{ij}^{V_{ij}^{(1)}}$ for $1 \leq i \leq n_1$. Set $\hat{sk}_j = H(\Delta_{1j}||\Delta_{2j}||\ldots||\Delta_{n_1 j})$.
       (c) If $j \neq n_3$, for $1 \leq i \leq n_2$ compute
           • $U_{ij+1}^{(1)}||W_1 = \hat{B}_{ij+1} \oplus \mathsf{PRF}_{\hat{sk}_j}(i||j+1||0)$
           • $U_{ij+1}^{(2)}||W_2 = \hat{Y}_{ij+1} \oplus \mathsf{PRF}_{\hat{sk}_j}(i||j+1||1)$
           If $W_1$ or $W_2$ is not equal to $0^\lambda$, report an error and abort.
    3. Set $\sigma_y = (V^{(1)}, \hat{sk}_{n_3})$.
$\mathsf{Verify}(\mathsf{SK}_x = \{sk_i\}_{i=1}^{n_3}, \sigma_y = (V^{(1)}, \hat{sk}_{n_3}))$: Verify whether $\hat{sk}_{n_3} = sk_{n_3}$. If the check succeeds, output $V^{(1)}$. Otherwise, retrieve all $\{\hat{sk}_i\}_{i=1}^{n_3}$ from the server and find the smallest index $i$ such that $\hat{sk}_i \neq sk_i$ using binary search.
```

**Fig. 3.** Description of scheme $\mathsf{VC}_r'$ that incorporates chaining and allows for fast location of an error in the computation.

supports one multiplication and an unlimited number of additions on encrypted messages. This allows for matrix multiplication $A \times B$ to be privately carried out on encrypted data. The server also sees $X$ and $Y$, but no changes to $Y$ are necessary. That is, the term $g_2^{w_j d_i}$ protects each element of matrix $B$ that matrix $Y$ encodes, assuming that the M-DDH assumption holds. Matrix $X$, however, in its original form can disclose information about the elements of $A$. To prevent this disclosure, we encode its elements in a form similar to that of matrix $Y$, by incorporating a random term $g_1^{h_j u_i}$ into each element of $X$.

*Rational Setting.* Unlike $\mathsf{VC}_m$, where the computation of product matrix $C$ and verification value $s$ can be carried out independently, in $\mathsf{VC}_r$ the server needs to perform these two computations together in order to produce correct key $\hat{sk}$ used for verification purposes. As a direct consequence of the difference, we no longer can apply an arbitrary encryption algorithm to matrices $A$ and $B$ because randomness used in ciphertexts will lead to the delegator's inability to properly compute $\hat{sk}$. To resolve the issue, we encode $A$ and $B$ using a similar mechanism to that of forming matrices $X$ and $Y$. That is, we use the product of two newly generated random values (such as $w_j d_i$ in $Y_{ij}$) to protect each individual element in $A$ and $B$, and update $vk_{ij}$ accordingly to allow for correct verification.

**Public Verifiability.** Recall that public verifiability allows any entity to verify correctness of the returned result using a public verification key. To incorporate public verifiability into $\mathsf{VC}_m$ and $\mathsf{VC}_r$, the client will now need to produce a public verification key $\mathsf{PVK}_x$ as a public version of $\mathsf{SK}_x$ at $\mathsf{ProbGen}$ time, which will roughly be in the form of $g^{\mathsf{SK}_x}$. This key then can be utilized by any auditor at $\mathsf{Verify}$ time to assess correctness of the result of outsourced computation.

*Rational Setting.* Recall that in $\mathsf{VC}_r$, client's secret key $\mathsf{SK}_x$ consists of only key $sk$. Therefore, all that is needed to convert the solution into a publicly verifiable scheme is to make a public version of the key, $g^{sk}$, publicly available. Then, the verification consists of checking whether $g^{sk} = g^{\hat{sk}}$, where, as before, $\hat{sk}$ is produced by the server during $\mathsf{Compute}$.

*Malicious Setting.* This time, unlike the rational setting, achieving public verifiability by setting $\mathsf{PVK}_x$ to be $g^v$ for every $v \in \mathsf{SK}_x$ does not work. Doing so would reveal crucial key information, which can be easily exploited by the server to compromise integrity of the returned result. In particular, suppose we set $\mathsf{PVK}_x$ to consist of $g_T^{r_i c_j}$ for $1 \le i \le n_1$ and $1 \le j \le n_3$ and $g_T^{\sum_{k=1}^{n_1} r_i v k_i}$, and let $\mathsf{Verify}$ consists of checking whether $\prod_{i=1}^{n_1} \prod_{j=1}^{n_3} (g_T^{r_i c_j})^{C_{ij}} + g_T^{\sum_{k=1}^{n_1} r_i v k_i} = s$ (where $s$ is returned by $\mathsf{Compute}$).[1] To launch an attack, the adversary first correctly computes $C$ and $s$, then changes an arbitrary matrix element from $C_{ij}$ to $C_{ij} + \delta$, and multiplies $s$ by $g_T^{r_i c_j \delta}$. This allows the adversary to produce a tuple $(\hat{C}, \hat{s})$ that differs from the correct $(C, s)$, but nevertheless passes verification. Notice that this attack is infeasible in the original $\mathsf{VC}_m$ scheme as the adversary has no information about $g_T^{r_i c_j}$ and hence is unable to correctly produce $\hat{s}$. Therefore, further changes are needed to support public verifiability in the malicious setting.

The idea behind the modification is to make $g_T^{r_i c_j}$'s and $s$ belong to two different groups. That is, we make the values of the form $g^{r_i c_j}$ available only in $\mathbb{G}_T$, while $s$ will have to be produced as a number of elements $s_i$ in $\mathbb{G}_2$ (and there is no efficiently computable homomorphism from an element of $\mathbb{G}_T$ to an element of $\mathbb{G}_2$). Then in the construction the delegator produces $\mathsf{PVK}_x$ as of $g_1^{r_i}$, $g_T^{r_i c_j}$ and $g_T^{\sum_{i=1}^{n_1} r_i v k_i}$ and provides only three matrices $A$, $B$, $Y$ in $\sigma_x$ to the server. The server computes each $s_i \in \mathbb{G}_2$ by performing a modulo exponentiation using $A$ and $Y$ (instead of the pairing operation in $\mathsf{VC}_m$) and returns them to the client. Because verification of the result now consists of checking whether the product of all $(g_T^{r_i c_j})^{C_{ij}}$'s and $g_T^{\sum_{i=1}^{n_1} r_i v k_i}$ matches the product of $e(g_1^{r_i}, s_i)$'s, it is no longer feasible for the adversary to succeed in the above attack.

The introduced modifications can also be applied to the version of the scheme with data privacy in both adversarial settings to obtain verifiable computation schemes with public delegatability and verifiability and privacy protection. Due to space constraints, detailed constructions of these schemes are provided in [34].

[1] Note that because $s$ is returned as an element of $\mathbb{G}_T$ and each $C_{ij}$ is returned as an element in $\mathbb{Z}_p^*$, the values $g_T^{r_i c_j}$ for each $i$ and $j$ and $g_T^{\sum_{k=1}^{n_1} r_i v k_i}$ represent the minimum information the verifier needs to possess to carry out verification.

## 7 Conclusions

This work presents schemes for verifiable outsourcing of matrix multiplications in both malicious and rational adversary models. The complexity and features of our schemes favorably compare to the state of the art, with the solution in the rational setting having a very low verification cost of only a single comparison. Our basic constructions achieve public delegatability and can be extended with features of data protection, public verifiability and chaining (supporting all or a subset of the features).

## Acknowledgments

This work was supported in part by grants CNS-1319090 and CNS-1223699 from the National Science Foundation and FA9550-13-1-0066 from the Air Force Office of Scientific Research. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

## References

1. S. Agrawal and D. Boneh. Homomorphic MACs: MAC-based integrity for network coding. In *ACNS*, pages 292–305, 2009.
2. B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, pages 152–163, 2010.
3. M. Atallah and M. Blanton, editors. *Algorithms and Theory of Computation Handbook. Volume I: General Concepts and Techniques*, chapter 17. CRC Press, 2009.
4. M. Atallah and K. Frikken. Securely outsourcing linear algebra computations. In *ASIACCS*, pages 48–59, 2010.
5. M. Atallah, K. Frikken, and S. Wang. Private outsourcing of matrix multiplication over closed semi-rings. In *SECRYPT*, pages 136–144, 2012.
6. P. D. Azar and S. Micali. Rational proofs. In *STOC*, pages 1017–1028, 2012.
7. P. D. Azar and S. Micali. Super efficient rational proofs. In *EC*, pages 29–30, 2013.
8. L. Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.
9. M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *CCS*, pages 863–874, 2013.
10. L. Ballard, M. Green, B. Medeiros, and F. Monrose. Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive 2005/417, 2005.
11. D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic group diffie-hellman key exchange under standard assumptions. In *EUROCRYPT*, pages 321–336, 2002.
13. D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *TCC*, pages 680–699, 2013.
14. CertiVox. Benchmarks and Subs performance for MIRACL library. https://certivox.org/display/EXT/Benchmarks+and+Subs.
15. K. Chung, Y. T. Kalai, F. Liu, and R. Raz. Memory delegation. In *CRYPTO*, pages 151–165, 2011.

16. K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.
17. Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *CRYPTO*, pages 112–130, 2000.
18. D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *CCS*, pages 501–512, 2012.
19. J. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *FOCS*, pages 648–657, 2013.
20. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.
21. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, pages 113–122, 2008.
22. S. Goldwasser, S. Micali., and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, Feb 1989.
23. S. Guo, S. Hubáček, A. Rosen, and M. Vald. Rational arguments: Single round delegation with sublinear verification. In *ITCS*, pages 523–540, 2014.
24. J. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In *STOC*, pages 623–632, 2004.
25. R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262, 2002.
26. J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.
27. J. Kilian. Improved efficient arguments. In *CRYPTO*, pages 311–324, 1995.
28. J. Landerman, V. Pan, and X.-H. Sha. On practical algorithms for accelerated matrix mutiplication. *Linear Algebra and Its Applications*, 162–164:557–588.
29. S. Micali. CS proofs. In *FOCS*, pages 436–453, 1994.
30. P. Mohassel. Efficient and secure delegation of linear algebra. Cryptology ePrint Archive 2011/605, 2011.
31. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, 2013.
32. B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, 2012.
33. L. Zhang and R. Safavi-Naini. Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. In *CNS*, pages 329–348, 2013.
34. Y. Zhang and M. Blanton. Efficient secure and verifiable outsourcing of matrix multiplications. Cryptology ePrint Archive 2014/133, 2014.

## A Security Analysis of $\mathsf{VC}_m$ Scheme

To demonstrate correctness of $\mathsf{VC}_m$ construction, we show that if the computation was performed correctly, $\mathsf{Verify}$ outputs product $A \times B$. In $\mathsf{Verify}$, we have:

$$g_T^{\sum_{i=1}^{n_1} r_i \sum_{j=1}^{n_3} c_j C_{ij} + vk_i} = \prod_{i=1}^{n_1} g_T^{r_i \sum_{j=1}^{n_3} c_j C_{ij} + vk_i} = \prod_{i=1}^{n_1} \prod_{j=1}^{n_3} g_T^{r_i c_j C_{ij} + r_i t_i w_j}$$
$$= \prod_{i=1}^{n_1} \prod_{j=1}^{n_3} \prod_{k=1}^{n_2} e(g_1^{r_i A_{ik}}, g_2^{c_j B_{kj} + w_j d_k}) = \prod_{i=1}^{n_1} \prod_{j=1}^{n_3} \prod_{k=1}^{n_2} e(X_{ik}, Y_{kj}) = s$$

*Proof (Theorem 1).* Our proof follows the hybrid argument. We start with the security experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}_m, f, \kappa)$ and devise a sequence of security games, where the adversary $\mathcal{A}$'s view in one game is indistinguishable from its view in another game. We analyze $\mathcal{A}$'s advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}_m, f, q, \kappa)$ in winning the experiment. Let $T_i$ denote the event that the security experiment returns 1 in game $\mathbf{G_i}$. The security games are defined as follows:

**Game $\mathbf{G_0}$.** Define $\mathbf{G_0}$ to be the same as $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}_m, f, \kappa)$.

**Game $\mathbf{G_1}$.** The game is identical to $\mathbf{G_0}$, except that when generating $Y_{ij}$, the delegator will use random value $r_{ij}^{(1)}$ in $\mathbb{Z}_p$ instead of $w_j d_i$. In other words, each $Y_{ij}$ is formed as $g_2{}^{c_j B_{ij} + r_{ij}^{(1)}}$ as opposed to $g_2^{c_j B_{ij} + w_j d_i}$ in game $\mathbf{G_0}$. To be able to verify the result of computation, the delegator also changes each $vk_i$ in $\mathsf{SK}_x$ from its original value $r_i \sum_{k=1}^{n_2} \sum_{j=1}^{n_3} A_{ik} d_k w_j$ to $r_i \sum_{k=1}^{n_2} \sum_{j=1}^{n_3} A_{ik} r_{kj}^{(1)}$, while keeping the remaining portion of $\mathsf{VC}_m$ construction unchanged. This will allow the delegator to verify the result of computation without any changes to $\mathsf{Verify}$.

Comparing $\mathcal{A}$'s view in games $\mathbf{G_0}$ and $\mathbf{G_1}$, we have that $g_1^{w_j d_i}$'s are replaced with random group elements. Now notice that all $g_1^{w_j d_i}$'s collectively form a partial $(n_2 + n_3)$-$\mathsf{M\text{-}DDH}$ tuple. This gives us that the advantage $\mathcal{A}$ has in game $\mathbf{G_0}$ is at most $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa)$ larger than in game $\mathbf{G_1}$ and thus any non-negligible difference in the adversary's behavior between the games $\mathbf{G_0}$ and $\mathbf{G_1}$ can be used to break the M-DDH assumption. Therefore, we have that $|\Pr[T_1] - \Pr[T_0]| \leq \mathbf{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa)$ and based on our assumption that the M-DDH problem is hard the difference in the adversary's view between games $\mathbf{G_0}$ and $\mathbf{G_1}$ is negligible.

**Game $\mathbf{G_2}$.** The game is identical to $\mathbf{G_1}$ except the delegator removes information about $c_j$ and $B_{ij}$ from each $Y_{ij}$, i.e., $Y_{ij} = g_2^{r_{ij}^{(1)}}$ instead of $g_2^{c_j B_{ij} + r_{ij}^{(1)}}$. To be able to verify the result of computation, we also update $\mathsf{SK}_x$ to compensate for the difference in $Y_{ij}$'s. We thus add the difference in the $vk_i$'s $\sum_{i=1}^{n_1} \sum_{j=1}^{n_3} r_i c_j (\sum_{k=1}^{n_2} A_{ik} B_{kj})$ to the value of $vk_1$ in $\mathbf{G_1}$ while keeping the remaining $vk_i$'s the same as in $\mathbf{G_1}$ (note that there are other possibilities because only $\sum_i vk_i$'s is used). Because the $r_{ij}^{(1)}$'s are completely random, the distribution of the $Y_{ij}$'s in $\mathbf{G_1}$ and in $\mathbf{G_2}$ is identical and thus $\Pr[T_2] = \Pr[T_1]$. Now observe that we removed any information about the $c_j$'s from $\mathcal{A}$'s view.

Let us next analyze $\mathcal{A}$'s success in winning $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}_m, f, \kappa)$ in $\mathbf{G_2}$. Assuming the XDH assumption is true, the M-DDH problem is hard in our setting with bilinear maps, i.e., it is hard in $\mathbb{G}_T$. Thus, while $\mathcal{A}$ can compute $g_T^{r_i}$ for each $i$, $(\{g_T^{r_i}\}_{1 \leq i \leq n_1}, \{g_T^{r_i c_j}\}_{1 \leq i \leq n_1, 1 \leq j \leq n_3})$ is a partial $(n_1 + n_3)$-M-DDH tuple and $\mathcal{A}$ can have only a negligible advantage in distinguishing the $g_T^{r_i c_j}$'s from random group elements. Now suppose that $\mathcal{A}$ was able to return a tuple $(\hat{C}, \hat{s})$ that differs from correct $(C, s)$, but nevertheless passes verification. Then the returned value satisfies the equation $g_T{}^{\sum_{i=1}^{n_1} \sum_{j=1}^{n_3} r_i c_j (C_{ij} - \hat{C}_{ij})} = s/\hat{s}$. Because the server has no information about the $c_j$'s and furthermore is unable to distinguish $g_T^{r_i c_j}$'s from random group elements, the only way for $\mathcal{A}$ to create simultaneously valid $C_{ij} - \hat{C}_{ij}$ and $s/\hat{s}$ is to correctly guess the value of $g_T^{r_i c_j}$. The probability of this happening is, however, negligible in $\kappa$ and thus $\Pr[T_2]$ is negligible as well.

Combined with the previous analysis of the differences in the adversarial success between games $\mathbf{G_0}$ and $\mathbf{G_2}$, we obtain that $\mathcal{A}$'s advantage is negligible in winning the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}_m, f, \kappa)$ as desired. $\square$