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. The cost of verifying the result of computation consists of a single modulo exponentiation and can be further reduced if the cloud is rational (or lazy). A lazy cloud tries to minimize its work by skipping the computation, but does not maliciously corrupt the data. Our solutions achieve 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 resource 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 one if the need to protect confidentiality of the data used in the outsourced computation, which can be proprietary, personal, or otherwise sensitive. Addressing these security objectives is thus 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, when the delegator runs multiple computations on the same setup, prior literature calls for further separation of the delegator's tasks into the initial setup that produces the common parameters and generates keys and the task of generating an instance of the computation to be placed on the cloud. For example, using the previous example of doctor's office, the setup can be performed by the doctor while each individual task is generated by a lab assistant. In this setting, it is obvious that lab assistant should not have access to the doctor's master key if present.

The specific problem that we treat in this work is that of matrix multiplication outsourcing. Because of popularity of matrix multiplication computation in a number of different domains and relatively high cost of this operation on large inputs, secure matrix multiplication outsourcing has received attention in research literature [3, 4, 17, 31]. We continue this line of research and show below how our results compare to the state of the art.

A novel feature that we put forward in this work and which is not present in publications that treat the same topic can be described 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. And 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. The fact that the computation is chained 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. Contrast this with other results that require verification of every matrix cell to ensure that the result of the computation is correct.

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 design 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. In the event that the server carries out the computation honestly, but get 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 is not going to 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 putting effort in the computation that will later have to be disregarded.

Our contributions can be summarized as follows:

- We present the core construction for verifiable outsourcing in presence of malicious adversaries who can arbitrarily deviate from the prescribed computation (Section 4). In our scheme, delegating a task requires work linear in the input size, carrying out the computation itself is also optimal, and verification of the resulting matrix product involves only a single modulo exponentiation.

- We present another construction that assumes rational, or lazy, adversaries who would like to conserve resources by performing the minimal amount of work that allows them to pass the verification test, but otherwise do not maliciously modify the returned output (Section 5.1). Verification of the returned result now involves only a single comparison.
- We extend the construction in the lazy setting to incorporate the chaining feature as described above without compromising any other properties (Section 5.3). In particular, this has no impact on the complexity of the resulting scheme.
- We next show how data privacy can be added to the constructions in both (malicious and lazy) settings (Section 6). This increases the cost of recovering the output, but it is still linear in the output size.
- Lastly, we extend our constructions to achieve public verifiability, in which any entity with access to public verification key can assess the validity of the returned result (Section 7). When such solutions are combined with data protection, access to the verification key does not allow for recovering the output of the computation. The complexities of these solutions also remain low, as in the previous constructions.

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

2 Related Work

We divide all related work in categories and discuss each category in turn.

Verifiable Computation. In verifiable computation [23, 19, 15, 2, 33, 32, 17, 12, 14], 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 formulated in work on interactive proofs (IP) [5, 24], efficient arguments based on probabilistically checkable proofs (PCP) [27, 28], and computationally sound (CS) proofs [30]. We highlight some of the more relevant results below.

A typical PCP-based scheme is normally not applicable to our scenario because in such schemes the client needs to maintain a large amount of data in order to verify the results of the computation, while in our setting the client is only capable of storing data linear in the input size. Recently, argument systems based on the work of Kilian [27, 28] have been built [35, 34, 38]. They require the prover to make a commitment to the computation that should be consistent with the queries that will be issued afterwards by the verifier. The client therefore needs to engage in an interactive protocol to carry out the verification procedure. This is also different from our setup, where the client is not involved in interactive computation besides submitting the problem to the server and receiving the results.

Succinct Non-Interactive Arguments of Knowledge (SNARKs) [7] are CS proofs of knowledge for NP that can potentially be utilized to realize verifiable computation. SNARKs represents a succinct argument system that allows a prover to prove the knowledge of NP statements by performing work independent of the size of the problem instance and its witness. The problem was motivated in early work of Kilian [27] and uses a Merkle hash tree and PCPs to achieve a four-message interactive argument for NP. It was later enhanced in [30] to achieve a one-message protocol by additionally incorporating Private Information Retrieval (PIR). As SNARKs are intended for succinct verification using either a designated or public verifier [8], they are considered as an ideal candidate for implementing verifiable computation. However, it is still unknown whether there exists a concrete SNARK construction that relies on standard cryptographic assumptions [20].

Scheme	Computation Verification	Privacy Protection	Public Verifiability	Output-Independent Verification	Deterministic Verification
Atallah et al. [3]	✓	✓	✗	✗	✗
Mohassel [31]	✓	✓	✗	✗	✗
Fiore et al. [17]	✓	✗	✓	✗	✓
Backes et al. [6]	✓	✗	✗	✗	✓
This work	✓	✓	✓	✓	✓

Table 1: Comparisons with related work.

Another line of work [19, 15, 14] uses fully homomorphic encryption [21] in combination with verifiable computation to additionally achieve data protection. Unfortunately, these solutions necessitate an expensive offline phase that requires the client to invest $poly(k, T)$ or $poly(n, k)$ time, where T is the time it takes to compute the function, n is the memory size, and k is a security parameter, to generate public keys or a commitment used for verification. Our solution does not require any large one-time overhead, which is of benefit to resource-constrained clients who lack sufficient resources, but still would like to delegate the computation.

Homomorphic MAC and Signature Schemes. A homomorphic Message Authentication Code (MAC) scheme [1] allows an entity with possession of a secret key sk to produce a tag σ that authenticates message m with a property that allows someone with access to public evaluation key pvk to combine multiple authenticators together. That is, given a set of authenticators $\sigma_1, \sigma_2, \ldots, \sigma_n$ for messages $m_1, m_2, \ldots, m_n$, any entity with pvk is able to homomorphically evaluate a function P on $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ and produce a short tag σ' that authenticates correctness of $m' = P(m_1, m_2, \ldots, m_n)$. In the public-key setting, signature schemes are used in place of MAC schemes to result in similar functionality [26]. To realize verifiable computation using a homomorphic MAC (or signature) scheme, a client first sends function P and (m_i, σ_i) for $1 \leq i \leq n$ as its inputs to a server. The server computes $m' = P(m_1, m_2, \ldots, m_n)$ as a result and the corresponding authenticator σ' and the returns the values to the client. The client next uses the MAC (or signature) authentication procedure on (m', σ') to verify correctness of computational result m'. In earlier schemes [26, 10, 1, 13], homomorphic authenticators were mainly used to verify evaluation results of a linear function. They were later extended in [9, 11, 18] to support more general functions such as constant-degree polynomials and arithmetic circuits with polynomially-bound degrees. In order to apply solutions based on homomorphic authenticators to verifiable computation, the verification mechanism should be much more efficient than executing the function P itself, which is unfortunately not the case for most available schemes. Furthermore, it is not clear as to how to protect secrecy of the underlying messages using these techniques as neither the MAC nor the signature schemes serve this purpose.

Matrix Multiplication. The problem of verifiable matrix multiplication has been studied in recent literature [17, 31, 3, 6]. In addition to computation verification, existing solutions offer additional important security features that are: (i) *data protection*, i.e., protection of both input and output matrices throughout the computation; (ii) *public verifiability*, i.e., the ability of any entity to verify the result of outsourced computation; and (iii) *deterministic verification*, i.e., the ability to detect faulty cells in an output matrix with probability 1. Unlike prior work, our scheme achieves all of these features. Additionally, we achieve another desirable property called *output-independent efficiency* allows for a constant number of cryptographic operations (modulo exponentiation or pairing operation) to be used during verification independent of the size of the output matrix. Table 1 summarizes features of our solution and other schemes.

Scheme	Client's Preparation	Server's Computation	Client's Verification
Fiore et al. [17] (priv. verif.)	$O(n^2)$	$O(n^3)$	$O(n^2)$
Our's (priv. verif.)	$O(n^2) \mid O(n^2)$	$O(n^3) \mid O(n^3)$	$O(1) \mid 0$
Fiore et al. [17] (pub. verif.)	$O(n^2)$	$O(n^3)$	$O(n^2)$
Our's (pub. verif.)	$O(n^2) \mid O(n^2)$	$O(n^3) \mid O(n^3)$	$O(n^2) \mid O(1)$
Our's (priv. verif. + privacy)	$O(n^2) \mid O(n^2)$	$O(n^3) \mid O(n^3)$	$O(n^2) \mid 0$
Our's (pub. verif. + privacy)	$O(n^2) \mid O(n^2)$	$O(n^3) \mid O(n^3)$	$O(n^2) \mid O(1)$

Table 2: Comparisons with most closely related schemes.

Because [17] has similar security features to our scheme, we compare computational complexities of these two works in terms of client's and server's computation. The results are presented in Table 2, where the time is measured in the number of cryptographic operations (modular exponentiation and pairing operations). Compared to Fiore et al. [17] that only deals with a malicious adversary, our work defends against both malicious and rational adversaries and therefore in the table we list complexities for both malicious and lazy settings separated by | (i.e., malicious | rational). Note that the constants hidden behind the big-O notations are the same or even slightly better in our schemes.

Hourglass Scheme. The hourglass scheme [37] is a technique employed by a tenant for enforcing an economically rational cloud provider to keep the user's data at rest encrypted (even when it possesses decryption keys). The basic idea consists of encoding the data after it has been encrypted in such a way that applying the encoding on the fly upon receiving the tenant's challenges imposes significant resource constraints on the provider. Therefore, by observing the time that it takes to serve the tenant's request, the tenant will know whether the provider is compiling with the data storage policy. Note that the scheme does not work in presence of fully malicious adversaries (who, e.g., can store both plaintext files and their encoded versions) and this is why the approach only targets economically rational adversaries who adopt a strategy that maximizes their profits. Our work applies a similar idea to computation of matrix multiplication instead of storage (i.e., if the server deviates from the prescribed computation, it will have to invest larger computation into passing verification).

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 κ 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.

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 κ, $|\Pr[\mathcal{A}^{F_k}(1^\kappa) - \Pr[\mathcal{A}^R(1^\kappa)]|$ is negligible* *in κ, 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.*

We often use notation PRF to refer to a pseudo-random function family.

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 that there is a trusted setup algorithm Set that, on input a security parameter 1^κ, 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 $\mathbb{G}_T$ (which also has 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 that we use in this work is the Multiple Decisional Diffie-Hellman Assumption m-M-DDH, 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 $m \geq 2$. Given $D = (g^{x_1}, \ldots, g^{x_m}, \{g^{x_i x_j}\}_{1 \leq i < j \leq m})$ for random $x_1, \ldots, x_m \in \mathbb{Z}_p$, and random tuple $D_{rand} = (g_1, \ldots, g_m, \{g_{ij}\}_{1 \leq i < j \leq m})$ in $\mathbb{G}$, we define the advantage of adversary $\mathcal{A}$ in solving the M-DDH problem as*

$$\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa) = |\Pr[\mathcal{A}(g, p, m, D) = 1] - \Pr[\mathcal{A}(g, p, m, D_{rand}) = 1]|$$

We say that the M-DDH assumption holds if for every PPT algorithm $\mathcal{A}$ $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa)$ is negligible.

One of our schemes is 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.

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

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

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

$\mathsf{Compute}(\mathsf{EK}_x, \sigma_x) \rightarrow \sigma_y$**:** Given an encoded input σ_x and an evaluation key EK_x, this algorithm is run by the worker to produce an encoded outcome σ_y, where $y = f(x)$.

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

Talk about the difference between this definition and, for instance, the one used in [17].

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 EK_{x_i} and input encoding σ_{x_i}. Eventually, $\mathcal{A}$ outputs the input x^* on which it would like to be challenged, obtains evaluation key EK_{x^*} and encoding σ_{x^*}, and produces output encoding σ'_y. The adversary succeeds if the output is different from $f(x^*)$ and the verification algorithm does not output an error $\perp$. 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
        $x_i \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_{i-1}}, \mathsf{EK}_{i-1})$
        $(\mathsf{SK}_i, \mathsf{EK}_i, \sigma_{x_i}) \leftarrow \mathsf{ProbGen}(x_i, \mathsf{params})$
    $x^* \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q)$
    $(\mathsf{SK}_{x^*}, \mathsf{EK}_{x^*}, \sigma_{x^*}) \leftarrow \mathsf{ProbGen}(x^*, \mathsf{params})$
    $\sigma'_y \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q, \sigma_{x^*}, \mathsf{EK}_{x^*})$
    $y' \leftarrow \mathsf{Verify}(\mathsf{SK}_{x^*}, \sigma_y)$
    if $y' \neq \perp$ 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 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 *We then say that a verifiable computation scheme* VC *is secure if for any PPT adversary* $\mathcal{A}$*, any* κ*, and any* $f \in F$*, it holds that* $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}, f, q, \kappa)$ *is negligible in* κ*.*

In this work we consider two types of adversaries: the first type is the traditional type of adversary that can arbitrarily deviate from the prescribed computation as defined by $\mathsf{Compute}$ functionality. We denote this type of adversary as malicious. The second type, which we denote as rational, is interested in investing minimal resources into the computation to ensure that the verification passes, but will not invest resources into intentionally modifying the computed results. The rationale behind including the second type of the adversary is that it allows us to design more efficient solutions if the server can be assumed not to intentionally corrupt the result.

3.4 Data Protection in Outsourced Computation

When the data used in an outsourced task is sensitive (e.g., private, proprietary, etc.), it is desirable to additionally guarantee secrecy of the data. To formulate *secrecy* protection of data used in verifiable computation, we define an interactive security experiment described next. In the experiment, the adversary $\mathcal{A}$ is allowed to query $\mathsf{ProbGen}$ on inputs x_i of its choice a polynomial number of times, obtain the corresponding evaluation keys EK_{x_i} and the encoded inputs σ_{x_i}, and carry out arbitrary computations including $\mathsf{Compute}$ that produces σ_{y_i}. Eventually, $\mathcal{A}$ outputs two strings $x^{(1)}$ and $x^{(2)}$ on which it would like to be challenged. A random bit b is drawn. $\mathcal{A}$ is given encoding of $x^{(b)}$ $\sigma_{x^{(b)}}$ together with the evaluation key $\mathsf{EK}_{x^{(b)}}$ produced using $\mathsf{ProbGen}$. The adversary $\mathcal{A}$ outputs its guess for b and wins if it could correctly determine the value of b, i.e., distinguish between inputs $x^{(0)}$ and $x^{(1)}$.

```
Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(\mathsf{VC}, f, \kappa)$
    $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, f)$
    for $i = 1$ to $q$ do
        $x_i \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_{i-1}}, \mathsf{EK}_{i-1})$
        $(\mathsf{SK}_i, \mathsf{EK}_i, \sigma_{x_i}) \leftarrow \mathsf{ProbGen}(x_i, \mathsf{params})$
    $\{x^{(0)}, x^{(1)}\} \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q)$
    $b \overset{R}{\leftarrow} \{0, 1\}$
    $(\mathsf{SK}_{x^{(b)}}, \mathsf{EK}_{x^{(b)}}, \sigma_{x^{(b)}}) \leftarrow \mathsf{ProbGen}(x^{(b)}, \mathsf{params})$
    $\hat{b} \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q, \sigma_{x^{(b)}}, \mathsf{EK}_{x^{(b)}})$
    if $\hat{b} = b$ 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 VC as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Priv}}(\mathsf{VC}, f, q, \kappa) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(\mathsf{VC}, f, \kappa) = 1] - 1/2$$

Definition 6 *We say that a verifiable computation scheme* VC *ensures data secrecy if for any PPT adversary* $\mathcal{A}$*, any* κ*, and any* $f \in F$*, it holds that the advantage* $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{Priv}}(\mathsf{VC}, f, q, \kappa)$ *is negligible.*

When we talk about verifiable computation schemes that provide data protection, we separate the algorithms for computation verification and output recovery. In particular, Verify now only assesses the validity of the returned (protected) output, while new algorithm Output recovers the data itself. More formally, we have:

$\mathsf{Verify}(\mathsf{SK}_x, \sigma_y) \rightarrow b$**:** Given an encoded output σ_y and the secret key SK_x, this algorithm outputs a bit b which is set to 1 iff verification was successful.

$\mathsf{Output}(\mathsf{SK}_x, \sigma_y) \to y \cup \perp$**:** Given an encoded output σ_y and the secret key SK_x, this algorithm returns the output y or $\perp$ if the output could not be recovered.

This separation is particularly useful for solutions with public verifiability (where the verifier may not have the ability to recover the output) as described next.

3.5 Public Verifiability

Recall that public verifiability allows any entity with access to public verification key assess correctness of the result of outsourced computation. To model this formally, we modify the $\mathsf{ProbGen}$ algorithm to produce public verification key, denoted by PVK_x, and have Verify take PVK_x instead of SK_x as before. More precisely, the interface for $\mathsf{ProbGen}$ is now

$$\mathsf{ProbGen}(x, \mathsf{params}) \to (\mathsf{SK}_x, \mathsf{PVK}_x, \mathsf{EK}_x, \sigma_x),$$

and the interface for Verify is either

$$\mathsf{Verify}(\mathsf{PVK}_x, \sigma_y) \to y \cup \perp \quad \text{or} \quad \mathsf{Verify}(\mathsf{PVK}_x, \sigma_y) \to b$$

without and with data protection, respectively. Note that $\mathsf{ProbGen}$ may still produce SK_x, which in particular is used to recover the output in schemes that support data protection.

Incorporating public verifiability into a verifiable computation scheme affect the security experiment in which the adversary participates, and we denote the new security experiment by $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathsf{VC}, f, \kappa)$. The difference from the original definition is that now the adversary is given public verification keys corresponding to all problems that it queried. For completeness of this work, we provide the full modified definition in Appendix A. As before, the advantage of an adversary $\mathcal{A}$ making at most $q = poly(\kappa)$ queries in the security experiment against some scheme VC is defined as

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathsf{VC}, f, q, \kappa) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathsf{VC}, f, \kappa) = 1]$$

We then obtain the following modified security definition:

Definition 7 *We say that a verifiable computation scheme with public verifiability* VC *is secure if for any PPT adversary* $\mathcal{A}$*, any* κ*, and any* $f \in F$*, it holds that* $\boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathsf{VC}, f, q, \kappa)$ *is negligible in* κ*.*

4 Matrix Multiplication in Presence of Malicious Adversary

4.1 Problem Formulation

The delegator would like to multiply two matrices A and B of dimensions $d_1 \times d_2$ and $d_2 \times d_3$, 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 a malicious adversary who tampers with the computation or its results regardless of operation costs and attempts to pass the verification test. Also, this basic construction does not achieve public verifiability: only the entity who possesses the secret key is able to attest the correctness of outsourced computation. In a consecutive solution in section 7 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 VC_{mal} to denote this scheme.

$\mathsf{Setup}(1^\kappa, f)$**:** Given f that indicates matrix multiplication, using the security parameter κ 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 $d_1 \times d_2$ and $d_2 \times d_3$, perform:

1. Choose $r_i \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le i \le d_1$, $d_j \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le j \le d_2$, and $c_k, f_k \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le k \le d_3$.
2. Compute $t_i = \sum_{k=1}^{d_2} A_{ik} d_k$ for $1 \le i \le d_1$, $f = \sum_{j=1}^{d_3} f_j$, and $vk_i = t_i f$ for $1 \le i \le d_1$.
3. Compute $X_{ij} = g_1^{r_i A_{ij}}$ for $1 \le i \le d_1$ and $1 \le j \le d_2$.
4. Compute $Y_{ij} = g_2^{c_j B_{ij} + f_j d_i}$ for $1 \le i \le d_2$ and $1 \le j \le d_3$.
5. Set $\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{vk_i\}_{i=1}^{d_1}, \{r_i\}_{i=1}^{d_1})$, $\mathsf{EK}_x = \mathsf{params}$, and $\sigma_x = (A, B, X, Y)$.

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

1. Compute $C = A \times B$.
2. Compute $s = \prod_{i=1}^{d_1} \prod_{j=1}^{d_3} \prod_{k=1}^{d_2} e(X_{ik}, Y_{kj})$.
3. Set $\sigma_y = (C, s)$.

$\mathsf{Verify}(\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{vk_i\}_{i=1}^{d_1}, \{r_i\}_{i=1}^{d_1}), \sigma_y = (C, s))$**:** Verify whether $g_T^{\sum_{i=1}^{d_1} r_i (\sum_{j=1}^{d_3} c_j C_{ij} + vk_i)} = s$. If the check succeeds, output C; otherwise, output $\perp$.

Figure 1: Description of the core scheme VC_{mal} in presence of malicious adversaries.

4.2 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 $X \times Y$. The client then verifies correctness of C by using secret relationship between the elements embedded in the product $X \times Y$.

In more detail, the intuition behind our construction is as follows: The delegator produces X as a randomized version $\hat{A}$ of A and produces Y as $\hat{B} + T$, where $\hat{B}$ is a randomized version of B and T is a random matrix. The delegator sends matrices $\{A, B, X, Y\}$ to the server who is instructed to compute product $C = A \times B$ and the sum of the elements in the product $D = X \times Y$ and return both results to the delegator. Note that matrix D that the server computes can be written as $X \times (\hat{B} + T) = \hat{A} \times \hat{B} + \hat{A} \times T$.

The secret key SK_x that the delegator forms at problem generation time consists of information about the product $A \times T$ and randomness used to encode A into X and B into $\hat{B}$. An important property is that T is formed in a special way so that in order to obtain $A \times T$ the delegator does not have to perform matrix multiplication, but instead does work linear in its input size. When the server returns C and $\sum_i \sum_j D_{ij}$, the delegator uses randomness used in producing $\hat{A}$ and $\hat{B}$ and information about the product $A \times T$ to verify that the sum of the elements in C after proper randomization matches the returned $\sum_i \sum_j D_{ij}$. If the verification succeeds, the delegator uses C as the correct output. A detailed description is 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 $O(d_1 d_2 + d_2 d_3)$. $\mathsf{Compute}$ involves the execution of two matrix multiplications resulting in complexity $O(d_1 d_2 d_3)$. Verify consists of ensuring the validity of C by checking its elements against s that $\mathsf{Compute}$ produces and has complexity $O(d_1 d_3)$, i.e. linear in the size of 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.

4.3 Analysis of the Scheme

To demonstrate correctness, we show that if the computation was performed correctly, Verify outputs product $A \times B$. In Verify, we have:

$$\begin{aligned}
g_T^{\sum_{i=1}^{d_1} r_i(\sum_{j=1}^{d_3} c_j C_{ij} + v k_i)} &= \prod_{i=1}^{d_1} g_T^{r_i \sum_{j=1}^{d_3} c_j C_{ij} + r_i v k_i} = \prod_{i=1}^{d_1}\prod_{j=1}^{d_3} g_T^{r_i c_j C_{ij} + r_i t_i f_j} \\
&= \prod_{i=1}^{d_1}\prod_{j=1}^{d_3} g_T^{\sum_{k=1}^{d_2} r_i A_{ik}(c_j B_{kj} + f_j d_k)} = \prod_{i=1}^{d_1}\prod_{j=1}^{d_3}\prod_{k=1}^{d_2} e(g_1^{r_i A_{ik}}, g_2^{c_j B_{kj} + f_j d_k}) \\
&= \prod_{i=1}^{d_1}\prod_{j=1}^{d_3}\prod_{k=1}^{d_2} e(X_{ik}, Y_{kj}) = s
\end{aligned}$$

The security property that this scheme achieves can be stated as follows:

Theorem 1 *Assuming that the M-DDH and XDH problems are hard, the verifiable computation scheme* VC_{mal} *is secure according to Definition 5.*

Proof Our proof follows the hybrid argument. We start with the security experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}, f, \kappa)$ and devise a sequence of security games, where the adversary's view in one game is indistinguishable from its view in another game. We consequently analyze the adversary's advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}, f, q, \kappa)$ 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 $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}, f, \kappa)$.

Game $\mathbf{G_1}$. The game is identical to game $\mathbf{G_0}$, except that when generating Y_{ij}, the delegator will use random value $r_{ij}^{(1)}$ in $\mathbb{Z}_p$ instead of $f_j d_i$. In other words, each Y_{ij} is formed as ${g_2}^{c_j B_{ij} + r_{ij}^{(1)}}$ as opposed to $g_2^{c_j B_{ij} + f_j d_i}$ in game $\mathbf{G_0}$.

Comparing the adversary's view in games $\mathbf{G_0}$ and $\mathbf{G_1}$, we have that values of the form $g_1^{f_j d_i}$ are replaced with random group elements. Now notice that the sequence of values $g_1^{f_j d_i}$ collectively form a partial $(d_2 + d_3)$-$\mathsf{M\text{-}DDH}$ tuple. This gives us that the advantage the adversary 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 $|\Pr[T_1] - \Pr[T_0]| \le \mathbf{Adv}_{\mathcal{A}}^{\mathsf{M\text{-}DDH}}(\kappa)$ and based on our assumption that the M-DDH problem is hard the difference in the adversary's view between the games $\mathbf{G_0}$ and $\mathbf{G_1}$ is negligible.

Game $\mathbf{G_2}$. The game is identical to $\mathbf{G_1}$ except that the delegator removes information about c_j and B_{ij} from the elements of Y. That is, instead of having each Y_{ij} equal to $g_2^{c_j B_{ij} + r_{ij}^{(1)}}$, it is now formed as $g_2^{r_{ij}^{(1)}}$. Because the $r_{ij}^{(1)}$'s are completely random, the distributions of the Y_{ij}'s in $\mathbf{G_1}$ and the Y_{ij}'s in $\mathbf{G_2}$ are identical and thus $\Pr[T_2] = \Pr[T_1]$.

Now observe that we completely removed any information about the c_j's from the adversary's view and next analyze its success of winning $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}, f, \kappa)$ in $\mathbf{G_2}$. Assuming that the XDH assumption is true, the M-DDH problem is hard in our setting that uses bilinear maps, i.e., it is hard in $\mathbb{G}_T$. Thus, while the adversary can compute ${g_T}^{r_i}$ for each i, the sequence

$({g_T}^{r_1}, \ldots, {g_T}^{r_{d_1}}, \{{g_T}^{r_i c_j}\}_{1 \le i \le d_1, 1 \le j \le d_3})$ is a partial $(d_1 + d_3)$-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.

Now suppose that the adversary is able to return a tuple $(\hat{C}, \hat{s})$ that differs from the correct one (C, s), but nevertheless passes the verification test. Then the returned value should satisfy the equation ${g_T}^{\sum_{i=1}^{d_1} \sum_{j=1}^{d_3} r_i c_j (C_{ij} - \hat{C}_{ij})} = s/\hat{s}$. Since the server has no information about the c_j's and furthermore is unable to distinguish the ${g_T}^{r_i c_j}$'s from random group elements, the only way for the adversary to create simultaneously valid $C_{ij} - \hat{C}_{ij}$ and $s/\hat{s}$ that pass the verification is to correctly guess the value of ${g_T}^{r_i c_j}$. The probability of this happening is, however, negligible in the security parameter and we obtain that $\Pr[T_2]$ 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 $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}, f, \kappa)$ as desired. $\square$

5 Matrix Multiplication in Presence of Lazy Adversary

Our next construction aims at defending against an adversary who attempts to pass verification by performing less work than what the computation prescribes. We term such an adversary as lazy. Unlike a malicious adversary who can tamper with the computation regardless of the cost, a lazy adversary returns correct output for the computation that it performed, but tries to do the minimal amount of work that passes the verification. If the adversary carried out the requested workload exactly, we assume the adversary returns the correct outcome to obtain compensation for its work. In this regard, the adversary's behavior is rational. In our solution against lazy adversaries VC_{lazy}, we would like to achieve two features: (i) to force a lazy 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 VC_{lazy} is similar to that of VC_{mal}: 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 VC_{mal}, correctness of $A \times B$ is verified by checking a secret relationship between the two matrix products. However, unlike VC_{mal}, where the delegator performs the verification itself, in VC_{lazy}, 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 the input and output. This scheme can be particularly suitable in the setting with three distinct 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 $\hat{A}$ of A and produces Y of the form $\hat{B} + T$, where $\hat{B}$ is a randomized version of B and T is a random matrix of special form, but additional randomization is used in $\hat{A}$ and $\hat{B}$ to ensure security in the modified setting. The delegator also releases a new matrix Z that includes some of the randomization used

$\mathsf{Setup}(1^\kappa, f)$**:** Given f that indicates matrix multiplication, using the security parameter κ 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 $d_1 \times d_2$ and $d_2 \times d_3$, perform:

1. Choose $r_i \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le i \le d_1$, $d_j, m_j \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le j \le d_2$, and $c_k, f_k \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le k \le d_3$.
2. Compute $X_{ij} = g_1^{r_i/m_j A_{ij}}$ for $1 \le i \le d_1$ and $1 \le j \le d_2$.
3. Compute $Y_{ij} = g_2^{c_j m_i B_{ij} + f_j d_i}$ for $1 \le i \le d_2$ and $1 \le j \le d_3$.
4. Compute $Z_{ij} = g_T^{c_j r_i}$ for $1 \le i \le d_1$ and $1 \le j \le d_3$.
5. Compute $t_i = \sum_{k=1}^{d_2} A_{ik} d_k / m_k$ for $1 \le i \le d_1$, and $vk_{ij} = t_i r_i f_j$ for $1 \le i \le d_1$ and $1 \le j \le d_3$.
6. Compute $\hat{sk}_{ij} = g_T^{vk_{ij}}$ for $1 \le i \le d_1$ and $1 \le j \le d_3$, set $sk_j = H(\hat{sk}_{1j} || \hat{sk}_{2j} || \ldots || \hat{sk}_{d_1 j})$ for $1 \le j \le d_3$, and $sk = H(sk_1 || sk_2 || \ldots || sk_{d_3})$.
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. For $j = 1, \ldots, d_3$ do:
   (a) For $1 \le i \le d_1$, compute
       • $V_{ij}^{(1)} = \sum_{k=1}^{d_2} A_{ik} B_{kj}$
       • $V_{ij}^{(2)} = \prod_{k=1}^{d_2} e(X_{ik}, Y_{kj}) = g_T^{\sum_{k=1}^{d_2} A_{ik}(c_j r_i B_{kj} + f_j d_k r_i / m_k)}$
       • $\Delta_{ij} = V_{ij}^{(2)} / Z_{ij}^{V_{ij}^{(1)}}$
2. Compute $\hat{sk}_j = H(\Delta_{1j} || \Delta_{2j} || \ldots || \Delta_{d_1 j})$ and $\hat{sk} = H(\hat{sk}_1 || \hat{sk}_2 || \ldots || \hat{sk}_{d_3})$.
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 $\perp$.

Figure 2: Description of the core scheme VC_{lazy} in presence of lazy adversaries.

in $\hat{A}$ and $\hat{B}$ to aid the server in producing a proof of correct computation. As before, the server is instructed with computing $A \times B$ and $X \times Y = X \times \hat{B} + X \times T$. This time the delegator precomputes $X \times T$ and sets its secret verification key to be the result of hashing of the individual elements of $X \times T$. Thus, to pass verification, the server has to recover the elements of $X \times T$ correctly. Now 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. Because this scheme is designed for lazy adversaries who will return correct information if it had to compute it, the server returns $A \times B$ that had to be computed to recover the correct verification key. The scheme is given in Figure 2.

The complexity of $\mathsf{ProbGen}$ is $O(d_1 d_2 + d_1 d_3 + d_2 d_3)$, i.e., linear in the size of the input and output. The complexity of $\mathsf{Compute}$ is dominated by two matrix multiplication resulting in $O(d_1 d_2 d_3)$ time. Lastly, the Verify algorithm performs a single string comparison of complexity $O(1)$ and outputs the matrix of size $d_1 \times d_3$.

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 matrix multiplication.

Matrix encoding: Given M of dimensions $d_1 \times d_2$, choose any value ℓ such that $A_{ij} + \ell \neq 0$ for $1 \leq i \leq d_1$ and $1 \leq j \leq d_2$. To form M', add ℓ to each element of M, i.e., $M'_{ij} = M_{ij} + \ell$, and store ℓ for future reference.

Matrix decoding: Let $C' = M' \times N'$, where M' (N') is an encoded version of matrix M (resp., N) using value ℓ_1 (resp., ℓ_2) and has dimensions $d_1 \times d_2$ (resp., $d_2 \times d_3$). Observe that each element $C'_{ij} = \sum_{k=1}^{d_2} (M_{ik} + \ell_1)(N_{kj} + \ell_2)$. To recover $C = M \times N$, compute the offset Δ_{ij} for each element, which equals to $\ell_2 \sum_{k=1}^{d_2} M_{ik} + \ell_1 \sum_{k=1}^{d_2} N_{kj} + d_2 \ell_1 \ell_2$, and set $C_{ij} = C'_{ij} - \Delta_{ij}$. Note that the value $\ell_2 \sum_{k=1}^{d_2} A_{ik}$ ($\ell_1 \sum_{k=1}^{d_2} B_{kj}$) 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 d_1 rows of matrix M (resp., d_3 columns of matrix N). The overall complexity of computing all offsets is therefore $O(d_1 d_2 + d_2 d_3)$ and the complexity of computing C from C' is $O(d_1 d_3)$.

The decoding computation is simplified when only one of the matrices used in the product was encoded to eliminate zero entries (as in VC_{lazy}). In that case, the offset becomes $\Delta_{ij} = \ell_1 \sum_{k=1}^{d_2} N_{kj}$ assuming that M was the encoded matrix, and the overall complexity of decoding is $O(d_1 d_3 + d_2 d_3)$.

5.2 Analysis

To demonstrate correctness, we show that if the computation was performed correctly, the server is able to recover the key sk. That is, an honest server will be able pass verification when $\hat{sk}$ is honestly computed as $\hat{sk} = H(\hat{sk}_1 || \hat{sk}_2 || \ldots || \hat{sk}_{d_3})$ and for each $j = 1, \ldots, d_3$, we have:

$$\begin{aligned}
\hat{sk}_j &= H(\Delta_{1j} || \Delta_{2j} || \ldots || \Delta_{d_1 j}) = H(V_{1j}^{(2)} / Z_{1j}{}^{V_{1j}^{(1)}} || V_{2j}^{(2)} / Z_{2j}{}^{V_{2j}^{(1)}} || \ldots || V_{d_1 j}^{(2)} / Z_{d_1 j}{}^{V_{d_1 j}^{(1)}}) \\
&= H(g_T{}^{\sum_{k=1}^{d_2} A_{1k}(f_j d_k / m_k) r_1} || g_T{}^{\sum_{k=1}^{d_2} A_{2k}(f_j d_k / m_k) r_2} || \ldots || g_T{}^{\sum_{k=1}^{d_2} A_{d_1 k}(f_j d_k / m_k) r_{d_1}}) \\
&= H(\hat{sk}_{1j} || \hat{sk}_{2j} || \ldots || \hat{sk}_{d_1 j}) = sk_j
\end{aligned}$$

That is, since $\hat{sk}_j = sk_j$ for each j, $\hat{sk} = sk$. Security of our VC_{lazy} scheme can be stated as follows:

Theorem 2 *If the verification of the returned result was successful, H is a collision-resistant hash function, the M-DDH and XDH assumptions hold, and all elements of A are non-zero, the server must have followed the correct protocol and computed the correct matrix products $A \times B$ and $X \times Y$.*

Proof Similar to the proof of Theorem 1, we proceed with a series of games using the hybrid argument and analyze the adversary's capabilities in the final game.

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

Game $\mathbf{G_1}$. The game is identical to game $\mathbf{G_0}$, except that when generating Y_{ij}, the delegator will use random value $r_{ij}^{(1)}$ in $\mathbb{Z}_p$ instead of $f_j d_i$. As before, assuming that $(d_2 + d_3)$-$\mathsf{M\text{-}DDH}$ problem is hard, the difference in the adversary's view between $\mathbf{G_1}$ and $\mathbf{G_0}$ is negligible.

Game $\mathbf{G_2}$. The game is identical to $\mathbf{G_1}$ except that the delegator removes information about c_j, m_i, and B_{ij} from each Y_{ij}, which is now formed as $g_2^{r_{ij}^{(1)}}$ instead of $g_2^{c_j m_i B_{ij} + r_{ij}^{(1)}}$. Because the $r_{ij}^{(1)}$'s are completely random, the distributions of the Y_{ij}'s in $\mathbf{G_1}$ and the Y_{ij}'s in $\mathbf{G_2}$ are identical and the adversary's view does not change.

Game $\mathbf{G_3}$. The game is identical to $\mathbf{G_2}$ except that the delegator replaces the r_i/m_j's used in generating X_{ij}'s with random values $r_{ij}^{(2)}$ and the $c_j r_i$'s used in generating Z_{ij}'s with random values $r_{ij}^{(3)}$. Because anyone with access to the public parameters can compute values equivalent to X_{ij} in $\mathbb{G}_T$ (i.e., $g_T^{r_i A_{ij}/m_j}$), we treat all values as an instance of a single M-DDH problem. Then assuming that the $(d_1+d_2+d_3)$-M-DDH problem defined on $g_T^{r_i}$'s ($g_1^{r_i}$'s), $g_T^{c_j}$'s, and g_T^{1/m_j}'s (g_1^{1/m_j}'s) is hard, the difference in the adversary's view between $\mathbf{G_2}$ and $\mathbf{G_3}$ is negligible.

Game $\mathbf{G_4}$. The game is the same as $\mathbf{G_3}$ except that X_{ij}'s are formed as $g_1^{r_{ij}^{(2)}}$ instead of $g_1^{r_{ij}^{(2)} A_{ij}}$. Assuming that each A_{ij} is non-zero, the adversary's view is unchanged from $\mathbf{G_3}$.

Now observe that we completely removed any information used in constructing the key sk from the adversary's view. This implies that the server is unable to pass the verification with more than negligible probability without performing some computation. To match sk, computed as $H(sk_1||sk_2||\ldots||sk_{d_3})$, the server needs to produce correct sk_i for $i = 1, \ldots, d_3$ because of the collision-resistance of hash function H. Similarly, because each each sk_i is formed as $H(\Delta_{1j}||\Delta_{2j}||\ldots||\Delta_{d_1 j})$, the server has to have Δ_{ij}'s to produce correct sk_i and pass verification.

Next, observe that sk is a function of d_i, f_j, and $1/m_k$ (more precisely, a function of their product) and information about $d_i f_j$ is encoded in Y and information about $1/m_k$ is encoded in X (and no other values contain information about d_i, f_j, or $1/m_k$). Then the only way for the server to derive $d_i f_j/m_k$ is to use X and Y as a whole and apply the pairing operation which will result in the desired product. Now notice that by doing this, the server will introduce the term ${g_T}^{c_j r_i \sum_{k=1}^{d_2} A_{ik} B_{kj}}$ that protects Δ_{ij}. To remove it, the server has to use Z_{ij}, which is the only value that is a function of c_j. Thus, the server is forced to compute ${Z_{ij}}^{V_{ij}^{(1)}}$ to obtain correct Δ_{ij}. This shows that in order to pass the verification, the adversary is forced to follow the computation and compute $V^{(1)}$, which corresponds to the product $A \times B$.

Furthermore, if the adversary is able to use $(\hat{V}_{ij}^{(1)}, \hat{V}_{ij}^{(2)})$ that differ from the correct one $(V_{ij}^{(1)}, V_{ij}^{(2)})$, but nevertheless passes the verification test, then the returned value should satisfy the equation ${g_T}^{r_i c_j (\hat{V}_{ij}^{(1)} - V_{ij}^{(1)})} = \hat{V}_{ij}^{(2)} - V_{ij}^{(2)}$. Because the server has no information about the c_j's and furthermore is unable to distinguish the ${g_T}^{r_i c_j}$'s from random group elements, the only way for the adversary to create simultaneously valid $\hat{V}_{ij}^{(1)} - V_{ij}^{(1)}$ and $\hat{V}_{ij}^{(2)} - V_{ij}^{(2)}$ is to correctly guess the value of ${g_T}^{r_i c_j}$. The probability of this happening is, however, negligible in the security parameter.

Because we assume that lazy adversary will return true information if it went into the effort of computing it, it follows that it will return correct product $A \times B$.

5.3 Description of 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 d_3 sub-computations. The keys sk_i are formed as before, but now the ith 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 at stages 1 through 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, where n is the total number of sub-computations. The server will also be able to examine the 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 notification to the server of the existence of faulty cells and the server can suspend the computation to identify faulty sells among the cells computed so far.

The detail of our solution can be described as follows: The client generates four matrices $\{A, B, X, Y\}$ and forms keys sk_i as in the base scheme. Now the computation of the ith column of matrices $A \times B$ and $X \times Y$ is considered to be the ith 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 pseudo-random string is produced using sk_i as the secret key to a pseudo-random function 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 ith 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 string output by PRF to be longer than the size of the 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_{d_3}, which the client compares to the expected key and accepts the result of the computation if the verification succeeds. The scheme is given in Figure 3.

The complexity of $\mathsf{ProbGen}$ and $\mathsf{Compute}$ algorithms remain the same as the basic scheme, i.e., $O(d_1d_2 + d_1d_3 + d_2d_3)$ and $O(d_1d_2d_3)$, respectively. The Verify algorithm performs a single string comparison of complexity $O(1)$ and outputs a matrix of size $d_1 \times d_3$ in case of no errors. If the check, however, fails, the client retrieves d_3 keys and additionally performs $O(\log d_3)$ work.

5.4 Analysis

To show correctness, it is trivial to demonstrate that the server is able to recover the inputs for the next sub-computation upon computing key sk_j for each j. Because the server can produce each random mask by computing $\mathsf{PRF}_{sk_{j-1}}(i||j||id)$, it is able to obtain the inputs and complete the computation.

We start showing security of the enhanced VC_{lazy} scheme with the following claim:

Lemma 1 *If H is a collision-resistant hash function,* PRF *is a pseudo-random function, the M-DDH and XDH assumptions hold, and all elements of A are non-zero, it is computationally difficult for the server who did not follow the prescribed computation for stages 1 through i to recover the input for $(i+1)$th sub-computation for $i \geq 1$.*

Proof We prove this by induction on i.

$\mathsf{Setup}(1^\kappa, f)$**:** The same as in VC_{lazy}.

$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$**:** On input two matrices A and B, perform:

```
1. Compute $X$, $Y$, $Z$, and $\{sk_i\}_{i=1}^{d_3}$ as in $\mathsf{VC}_{lazy}$.
2. Set $\hat{B}_{i1} = B_{i1}$ and $\hat{Y}_{i1} = Y_{i1}$ for $1 \le i \le d_2$; 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 $1 \le i \le d_2$ and $2 \le j \le d_3$, where $\lambda$ is a correctness parameter.
3. Set $\mathsf{SK}_x = \{sk_i\}_{i=1}^{d_3}$, $\mathsf{EK}_x = (\mathsf{params}, Z)$, 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 the following steps:

```
1. Set $U_{i1}^{(1)} = \hat{B}_{i1}$ and $U_{i1}^{(2)} = \hat{Y}_{i1}$ for $1 \le i \le d_2$.
2. For $j = 1, \ldots, d_3$ do:
   (a) For $1 \le i \le d_1$, compute
       • $V_{ij}^{(1)} = \sum_{k=1}^{d_2} A_{ik} U_{kj}^{(1)}$
       • $V_{ij}^{(2)} = \prod_{k=1}^{d_2} e(X_{ik}, U_{kj}^{(2)})$
   (b) Let $\Delta_{ij} = V_{ij}^{(2)} / {Z_{ij}}^{V_{ij}^{(1)}}$ for $1 \le i \le d_1$. Compute $\hat{sk}_j = H(\Delta_{1j}||\Delta_{2j}||\ldots||\Delta_{d_1 j})$.
   (c) If $j \ne d_3$, for $1 \le i \le d_2$ 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 $\sigma_y = (V^{(1)}, \hat{sk}_{d_3})$.
```

$\mathsf{Verify}(\mathsf{SK}_x = \{sk_i\}_{i=1}^{d_3}, \sigma_y = (V^{(1)}, \hat{sk}_{d_3}))$**:** Verify whether $\hat{sk}_{d_3} = sk_{d_3}$. If the check succeeds, output $V^{(1)}$. Otherwise, retrieve all $\{\hat{sk}_i\}_{i=1}^{d_3}$ from the server and find the smallest index i such that $\hat{sk}_i \ne sk_i$ using binary search.

Figure 3: Description of scheme VC_{lazy} that incorporates chaining and allows for fast location of an error in the computation.

Basic step: Let $i = 1$. The server receives inputs to the first sub-computation in the clear and would like to recover the inputs for the next sub-computation. First, notice that the inputs to the second-computation are protected by the output of PRF evaluated on unique points. This means that a computationally-bounded server will be unable to learn any information about the inputs with greater than negligible probability without access to sk_1. To recover sk_1, however, the only option for the server is to comply with the prescribed computation, using exactly the same argument as in the proof of Theorem 2.

We obtain that an adversary who deviates from the prescribed computation is unable to recover sk_1 and thus the inputs for the second sub-computation, as claimed.

Induction step: Suppose that the claim is true for some $k = i$ and we show that it is true for $k = i + 1$. The argument proceeds similarly to the basic step above. That is, by induction we know that it is infeasible to recover inputs for the ith sub-computation without following the prescribed computation. Therefore, if the server was able to recover the inputs to the ith sub-computation, it must be difficult to recover the inputs to the $(i + 1)$th sub-computation without honestly carrying out the computation for the ith sub-computation. The security argument proceeds in the same way as in the basic step.

If the server was not able to recover the inputs to the ith sub-computation, it is still infeasible to recover the inputs to the $(i+1)$th sub-computation due to the security of the PRF function used. This completes the proof.

Theorem 3 *If the verification of the returned result was successful, H is a collision-resistant hash function, PRF is a pseudo-random function, the M-DDH and XDH assumptions hold, and all elements of A are non-zero, the server must have followed the correct protocol and computed the correct matrix products $A \times B$ and $X \times Y$.*

Proof Because the server passed the verification, we know that it must have computed the last key sk_{d_3}. According to the argument used in the proof of Theorem 2, we know that the server must have carried out the d_3th sub-computation as prescribed. Also, according to Lemma 2, the server must have computed all prior sub-computations 1 through $d_3 - 1$ in order to recover the inputs for the d_3th sub-computation. This means that the server must have computed the products $A \times B$ and $X \times Y$.

Corollary 1 *If the result of the returned computation does not verify, the client will be able to identify the first faulty sub-computation using $O(\log(d_3))$ string comparisons.*

Proof Based on Lemma 2, if a key sk_i was not computed correctly, it is computationally difficult to recover all consecutive keys $sk_{i+1}, \ldots, sk_{d_3}$. This means that to identify the first faulty sub-task, the client needs to find i such that the server's $\hat{sk}_i \neq sk_i$, but $\hat{sk}_{i-1} = sk_{i-1}$. Finding such an i using binary search can be accomplished in $O(\log(d_3))$ steps.

6 Support for Matrix Privacy

In this section, we extend both of our previous constructions, VC_{mal} and VC_{lazy}, to incorporate privacy (or data secrecy) protection of input/output matrices. We denote the resulting schemes as VC^p_{mal} and VC^p_{lazy}, respectively. In more detail, we would like to ensure that the server who executes the outsourced task is unable to learn any information regarding input matrices A and B, and consecutively their product $A \times B$, throughout the computation. We thus modify both schemes to achieve this goal, while still preserving their original properties of verifiable computation.

6.1 Support for Matrix Privacy in Malicious Setting

In VC^p_{mal}, we wish to guarantee that the server is unable to infer any information about the input matrices A and B from σ_x and EK_x that $\mathsf{ProbGen}$ outputs. Because EK_x consists of only public parameters, we need to concentrate on σ_x that consists of matrices A, B, X, and Y. To protect A and B, our solution encodes them using a homomorphic encryption scheme that supports one multiplication and an unlimited number of additions on messages in encrypted form. This allows for matrix multiplication $A \times B$ to be privately carried out on encrypted data. With respect to the remaining matrices, notice that no changes to Y are necessary. That is, the term $g_2^{f_j d_i}$ protects each element of matrix B that matrix Y encodes, assuming that the M-DDH assumption holds. Matrix X in its original form, however, can disclose information about the elements of matrix A. To prevent this disclosure, we encode its elements in a form similar to that of matrix Y, by incorporating a random term $g_1^{h_j e_i}$ into each element of Y. The detailed protocol is presented in Figure 4.

$\mathsf{Setup}(1^\kappa, f)$**:** The same as in VC_{mal}.

$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$**:** On input two matrices A and B of respective dimensions $d_1 \times d_2$ and $d_2 \times d_3$, perform:

1. Choose $r_i, e_i \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \leq i \leq d_1$, $d_j, h_j \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \leq j \leq d_2$, and $c_k, f_k \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \leq k \leq d_3$.
2. Compute $t_i = \sum_{k=1}^{d_2} A_{ik} d_k$ for $1 \leq i \leq d_1$, $f = \sum_{j=1}^{d_3} f_j$, and $vk_i = r_i t_i f$ for $1 \leq i \leq d_1$.
3. Compute $m_i = \sum_{k=1}^{d_2} h_k B_{ki}$ for $1 \leq i \leq d_3$, $c = \sum_{i=1}^{d_3} c_i m_i$, and $vk_i = vk_i + e_i c$ for $1 \leq i \leq d_1$.
4. Compute $h = \sum_{k=1}^{d_2} h_k d_k$, $f = \sum_{j=1}^{d_3} f_j$, and $vk_i = vk_i + e_i h f$ for $1 \leq i \leq d_1$.
5. Compute $X_{ij} = g_1^{r_i A_{ij} + h_j e_i}$ for $1 \leq i \leq d_1$ and $1 \leq j \leq d_2$.
6. Compute $Y_{ij} = g_2^{c_j B_{ij} + f_j d_i}$ for $1 \leq i \leq d_2$ and $1 \leq j \leq d_3$.
7. Setup homomorphic encryption scheme Enc; denote by params_E the scheme's public parameters and by dk private decryption key.
8. Encrypt each element of A and B to produce $\mathsf{Enc}(A_{ik})$ and $\mathsf{Enc}(B_{kj})$ for $1 \leq i \leq d_1$, $1 \leq k \leq d_2$, and $1 \leq j \leq d_3$.
9. Set $\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{vk_i\}_{i=1}^{d_1}, \{r_i\}_{i=1}^{d_1}, dk)$, $\mathsf{EK}_x = (\mathsf{params}, \mathsf{params}_E)$, and $\sigma_x = (\mathsf{Enc}(A), \mathsf{Enc}(B), X, Y)$.

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

1. Compute $\mathsf{Enc}(C) = \mathsf{Enc}(A) \times \mathsf{Enc}(B)$.
2. Compute $s = \prod_{i=1}^{d_1} \prod_{j=1}^{d_3} \prod_{k=1}^{d_2} e(X_{ik}, Y_{kj})$.
3. Set $\sigma_y = (\mathsf{Enc}(C), s)$.

$\mathsf{Verify}(\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{vk_i\}_{i=1}^{d_1}, \{r_i\}_{i=1}^{d_1}, dk), \sigma_y = (\mathsf{Enc}(C), s))$**:** Decrypt $\mathsf{Enc}(C)$ using dk to recover C and verify whether $g_T^{\sum_{i=1}^{d_1} r_i (\sum_{j=1}^{d_3} c_j C_{ij}) + vk_i} = s$. If the check succeeds, output 1; otherwise, output 0.

$\mathsf{Output}(\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{vk_i\}_{i=1}^{d_1}, \{r_i\}_{i=1}^{d_1}, dk), \sigma_y = (\mathsf{Enc}(C), s))$**:** Decrypt $\mathsf{Enc}(C)$ using dk to recover and output C or $\perp$ in case of failure.

Figure 4: Description of scheme VC_{mal}^p for malicious adversaries that achieves data privacy.

To protect the original elements of matrix X, the delegator chooses additional random values e_i and h_j in $\mathsf{ProbGen}$, which are used incorporated into the elements of X. Because the format of the elements in X is changed from $g_1^{r_i A_{ij}}$ to $g_1^{r_i A_{ij} + h_j e_i}$, the key vk should also be updated accordingly to satisfy the verification equation. This change does not affect the complexity of $\mathsf{ProbGen}$, which remains $O(d_1 d_2 + d_2 d_3)$. In $\mathsf{Compute}$, the server executes the protocol in the same way as in VC_{mal} except that the server now performs matrix multiplication on ciphertexts $\mathsf{Enc}(A_{ij})$ and $\mathsf{Enc}(B_{ij})$ rather than cleartext and returns the encrypted matrix $\mathsf{Enc}(C)$ to the client. The complexity of $\mathsf{Compute}$ is still dominated by two matrix multiplication resulting in $O(d_1 d_2 d_3)$ time. In Verify, the client decrypts $\mathsf{Enc}(C)$ to obtain C in the clear and then follows the verification procedure of VC_{mal}. As before, the procedure has complexity $O(d_1 d_3)$ (i.e., linear in the size of the output). Output has the same complexity $O(d_1 d_3)$.

6.1.1 Analysis

To demonstrate correctness, we show that if the computation was performed correctly, Verify output 1, the delegator recovers product $A \times B$. We have:

$$
\begin{aligned}
g_T^{\sum_{i=1}^{d_1} r_i(\sum_{j=1}^{d_3} c_j C_{ij}) + v k_i} &= \prod_{i=1}^{d_1} g_T^{r_i \sum_{j=1}^{d_3} c_j C_{ij} + v k_i} = \prod_{i=1}^{d_1}\prod_{j=1}^{d_3} g_T^{r_i c_j C_{ij}} \times \prod_{i=1}^{d_1} g_T^{v k_i} \\
&= \prod_{i=1}^{d_1}\prod_{j=1}^{d_3} g_T^{\sum_{k=1}^{d_2} r_i c_j A_{ik} B_{kj}} \times \prod_{i=1}^{d_1}\prod_{j=1}^{d_3} g_T^{\sum_{k=1}^{d_2} (A_{ik} d_k r_i f_j + h_k B_{kj} e_i c_j + h_k d_k f_j e_i)} \\
&= \prod_{i=1}^{d_1}\prod_{j=1}^{d_3}\prod_{k=1}^{d_2} e(g_1^{r_i A_{ik} + e_i h_k}, g_2^{c_j B_{kj} + f_j d_k}) = \prod_{i=1}^{d_1}\prod_{j=1}^{d_3}\prod_{k=1}^{d_2} e(X_{ik}, Y_{kj}) = s
\end{aligned}
$$

The data protection property of VC^p_{mal} construction can be stated as follows:

Theorem 4 *Assuming that the M-DDH and XDH problems are hard and* Enc *is a semantically secure encryption scheme, the verifiable computation scheme* VC^p_{mal} *achieves data secrecy according to Definition 6.*

Proof First, notice that the input matrices A and B are protected with secure encryption. This means that we only need to concentrate on the remaining matrices X and Y that the server receives. The goal is to ensure that no information about the inputs is revealed from X or Y throughout the computation. Towards the goal, we proceed with a series of games using the hybrid argument and analyze the adversary's capabilities in the final game.

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

Game $\mathbf{G_1}$. The game is identical to game $\mathbf{G_0}$, except that when generating X_{ij} and Y_{ij}, the delegator will use random values $r^{(1)}_{ij}$ and $r^{(2)}_{ij}$ from $\mathbb{Z}_p$ instead of $h_j e_i$ and $f_j d_i$, respectively. As before, assuming that $(d_1 + d_2)$-$\mathsf{M\text{-}DDH}$ and $(d_2 + d_3)$-$\mathsf{M\text{-}DDH}$ problems are hard, the difference in the adversary's view between $\mathbf{G_1}$ and $\mathbf{G_0}$ is negligible.

Game $\mathbf{G_2}$. The game is identical to $\mathbf{G_1}$ except that the delegator removes information about r_i and A_{ij} from each X_{ij}, which is now formed as $g_1^{r^{(1)}_{ij}}$ instead of $g_1^{r_i A_{ij} + r^{(1)}_{ij}}$. Similarly, the delegator removes information about c_j and B_{ij} from each Y_{ij}, which is now formed as $g_2^{r^{(2)}_{ij}}$ instead of $g_2^{c_j B_{ij} + r^{(2)}_{ij}}$. Because the $r^{(1)}_{ij}$'s ($r^{(2)}_{ij}$'s) are completely random, the distributions of X_{ij}'s (resp., Y_{ij}'s) in $\mathbf{G_1}$ and the X_{ij}'s (resp., Y_{ij}'s) in $\mathbf{G_2}$ are identical and the adversary's view does not change.

Now observe that we completely eliminated any information about the inputs A and B from the adversary's view, which makes the probability of the server winning game $\mathbf{G_2}$ exactly 1/2. Combined with the previous analysis of differences in the adversarial success between games $\mathbf{G_0}$ and $\mathbf{G_2}$, we obtain the adversary's advantage is negligible in winning the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(\mathsf{VC}^p_{mal}, f, \kappa)$.

Theorem 5 *Assuming that the M-DDH and XDH problems are hard, the verifiable computation scheme* VC^p_{mal} *is secure according to Definition 5.*

Proof There are two differences in the adversary's view between VC_{mal} and VC^p_{mal}: (i) in VC^p_{mal}, the input matrices A and B are sent to the server encrypted and (ii) in VC^p_{mal}, each element of matrix X includes an additional random term $g_1^{h_j e_i}$. These differences, however, do not give any advantage to the adversary in winning the security game $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\mathsf{VC}_{mal}^{p}, f, \kappa)$. Therefore, the proof proceeds similar to the proof of Theorem 1 and the claim still holds.

6.2 Support for Matrix Privacy in Lazy Setting

Now we would like to modify VC_{lazy} to add data protection. Unlike VC_{mal}, where the computation of product matrix C and verification value s can be carried out independently, in VC_{lazy} the server needs to perform these two computations together in order to produce correct key $\hat{sk}$ used for verification purposes. A direct consequence of the difference is that we can no longer apply 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. The format of these matrices protects information about A and B, assuming each element of A takes a non-zero value. Description of how to modify input matrices to contain only non-zero values was given in Section 5.1. Our VC_{lazy}^{p} construction is given in Figure 5.

The differences from VC_{lazy} is that in $\mathsf{ProbGen}$ the client now generates additional random values and uses them in combination with other values to form new matrices X' and Y' from input matrices A and B in a similar way X and Y are formed. Furthermore, there is now no need for the client to produce matrix Z as part of the evaluation key because secret random values c_j's and r_i's are now incorporated into X' and Y'. The overall complexity of $\mathsf{ProbGen}$ is still $O(d_1d_2 + d_1d_3 + d_2d_3)$. In $\mathsf{Compute}$, the server performs almost the same computation as in VC_{lazy}, except that in order to produce $V^{(1)}$, the server needs to perform $d_1d_2d_3$ pairing operations instead of modular exponentiations. The overall complexity of $\mathsf{Compute}$ is still $O(d_1d_2d_3)$. In Verify, the verification procedure is a single comparison and therefore has complexity $O(1)$. If the verification passes, the client calls Output and uses c_j's, r_i's, and $\hat{vk}_{ij}$'s from its secret key to recover product $C = A \times B$ from $V^{(1)}$. In particular, to obtain each matrix element C_{ij}, the client needs to compute the discrete logarithm of $(V_{ij}^{(1)}/g_T^{\hat{vk}_{ij}})^{(c_j r_i)^{-1}}$ to the base g_T, which in general is hard. However, if we restrict matrix elements to a small range, the discrete logarithm can be computed by trying all values in the range or precomputing all powers of g_T. The complexity of Output is thus $O(d_1d_3)$ cryptographic operations due to the need to recover C.

6.2.1 Analysis

To demonstrate correctness, we show that if the computation was performed correctly, the server is able to recover the key sk. For each $j = 1, \ldots, d_3$, we have:

$$\begin{aligned}
\hat{sk}_j &= H(\Delta_{1j}||\ldots||\Delta_{d_1 j}) = H(V_{1j}^{(1)}/V_{1j}^{(2)}||\ldots||V_{d_1 j}^{(1)}/V_{d_1 j}^{(2)}) \\
&= H(g_T{}^{\sum_{k=1}^{d_2} A_{1k}(f_j d_k/m_k) r_1 - \sum_{k=1}^{d_2} A_{1k}(u_j s_k/n_k) r_1}||\ldots||g_T{}^{\sum_{k=1}^{d_2} A_{d_1 k}(f_j d_k/m_k) r_{d_1} - \sum_{k=1}^{d_2} A_{d_1 k}(u_j s_k/n_k) r_{d_1}}) \\
&= H(\hat{sk}_{1j}||\ldots||\hat{sk}_{d_1 j}) = sk_j
\end{aligned}$$

Then since $\hat{sk}_j = sk_j$ for each j, $\hat{sk} = sk$. The data protection property of VC_{lazy}^{p} construction can be stated as follows:

Theorem 6 *Assuming that the M-DDH and XDH problems are hard and A does not contain zero elements, the verifiable computation scheme* VC_{lazy}^{p} *achieves data secrecy according to Definition 6.*

$\mathsf{Setup}(1^\kappa, f)$**:** The same as in VC_{lazy}.

$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$**:** On input two matrices A and B of respective dimensions $d_1 \times d_2$ and $d_2 \times d_3$, perform:

1. Choose $r_i \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le i \le d_1$, $d_j, n_j, m_j, s_j \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le j \le d_2$, and $c_k, f_k, u_k \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le k \le d_3$.
2. Compute $X_{ij} = g_1^{r_i/m_j A_{ij}}$ for $1 \le i \le d_1$ and $1 \le j \le d_2$.
3. Compute $Y_{ij} = g_2^{c_j m_i B_{ij} + f_j d_i}$ for $1 \le i \le d_2$ and $1 \le j \le d_3$.
4. Compute $X'_{ij} = g_1^{r_i/n_j A_{ij}}$ for $1 \le i \le d_1$ and $1 \le j \le d_2$.
5. Compute $Y'_{ij} = g_2^{c_j n_i B_{ij} + u_j s_i}$ for $1 \le i \le d_2$ and $1 \le j \le d_3$.
6. Compute $t_i = \sum_{k=1}^{d_2} A_{ik} d_k / m_k$ for $1 \le i \le d_1$, and $\hat{vk}_{ij} = t_i r_i f_j$ for $1 \le i \le d_1$ and $1 \le j \le d_3$.
7. Compute $t'_i = \sum_{k=1}^{d_2} A_{ik} s_k / n_k$ for $1 \le i \le d_1$, and $vk_{ij} = \hat{vk}_{ij} - t'_i r_i u_j$ for $1 \le i \le d_1$ and $1 \le j \le d_3$.
8. Compute $\hat{sk}_{ij} = g_T^{vk_{ij}}$ for $1 \le i \le d_1$ and $1 \le j \le d_3$, set $sk_j = H(\hat{sk}_{1j} || \hat{sk}_{2j} || \ldots || \hat{sk}_{d_1 j})$ for $1 \le j \le d_3$, and $sk = H(sk_1 || sk_2 || \ldots || sk_{d_3})$.
9. Set $\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{r_i\}_{i=1}^{d_1}, \{\hat{vk}_{ij}\}_{i=1,j=1}^{d_1,d_3}, sk)$, $\mathsf{EK}_x = (\mathsf{params})$, and $\sigma_x = (X, Y, X', Y')$.

$\mathsf{Compute}(\mathsf{EK}_x, \sigma_x = (X, Y, X', Y'))$**:** Execute the following steps:

1. For $1 \le j \le d_3$ and $1 \le i \le d_1$, compute
   - $V_{ij}^{(1)} = \prod_{k=1}^{d_2} e(X_{ik}, Y_{kj}) = g_T{}^{\sum_{k=1}^{d_2} A_{ik}(c_j r_i B_{kj} + f_j d_k r_i / m_k)}$
   - $V_{ij}^{(2)} = \prod_{k=1}^{d_2} e(X'_{ik}, Y'_{kj}) = g_T{}^{\sum_{k=1}^{d_2} A_{ik}(c_j r_i B_{kj} + u_j s_k r_i / n_k)}$
   - $\Delta_{ij} = V_{ij}^{(1)} / V_{ij}^{(2)}$
2. Compute $\hat{sk}_j = H(\Delta_{1j} || \Delta_{2j} || \ldots || \Delta_{d_1 j})$ and $\hat{sk} = H(\hat{sk}_1 || \hat{sk}_2 || \ldots || \hat{sk}_{d_3})$.
3. Set $\sigma_y = (V^{(1)}, \hat{sk})$.

$\mathsf{Verify}(\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{r_i\}_{i=1}^{d_1}, \{\hat{vk}_{ij}\}_{i=1,j=1}^{d_1,d_3}, sk), \sigma_y = (V^{(1)}, \hat{sk}))$**:** Verify whether $\hat{sk} = sk$ and output 1 if the verification was successful and 0 otherwise.

$\mathsf{Output}(\mathsf{SK}_x = (\{c_i\}_{i=1}^{d_3}, \{r_i\}_{i=1}^{d_1}, \{\hat{vk}_{ij}\}_{i=1,j=1}^{d_1,d_3}, sk), \sigma_y = (V^{(1)}, \hat{sk}))$**:** Compute $g_T^{C_{ij}} = (V_{ij}^{(1)} / g_T^{\hat{vk}_{ij}})^{(c_j r_i)^{-1}}$, recover C_{ij} from the result, and output C. If the computation fails, output $\perp$.

Figure 5: Description of scheme VC_{lazy}^p for lazy adversaries that achieves data privacy.

Proof We proceed with a series of games using the hybrid argument and analyze the adversary's capabilities in the final game.

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

Game $\mathbf{G_1}$. The game is identical to game $\mathbf{G_0}$, except that when generating Y_{ij}s and Y'_{ij}s, the delegator will use random values $r_{ij}^{(1)}$ and $r_{ij}^{(2)}$ in $\mathbb{Z}_p$ instead of $f_j d_i$ and $u_j s_i$, respectively. As before, assuming $(d_2 + d_3)$-$\mathsf{M\text{-}DDH}$ problem is hard, the difference in the adversary's view between $\mathbf{G_1}$ and $\mathbf{G_0}$ is negligible.

Game $\mathbf{G_2}$. The game is identical to $\mathbf{G_1}$ except that the delegator removes information about c_j,

B_{ij}, and m_i (n_i) from each Y_{ij} (resp., Y'_{ij}). Each Y_{ij} is now formed as $g_1^{r_{ij}^{(1)}}$ instead of $g_1^{c_j m_i B_{ij} + r_{ij}^{(1)}}$ and each Y'_{ij} is now formed as $g_1^{r_{ij}^{(2)}}$ instead of $g_1^{c_j n_i B_{ij} + r_{ij}^{(2)}}$. Because the $r_{ij}^{(1)}$'s and $r_{ij}^{(2)}$'s are completely random, the distributions of Y_{ij}'s and Y'_{ij}'s in $\mathbf{G_1}$ and their distribution n $\mathbf{G_2}$ are identical and the adversary's view does not change.

Game $\mathbf{G_3}$. The game is identical to $\mathbf{G_2}$ except that when generating X_{ij}'s, the delegator will use random values $r_{ij}^{(3)}$ in $\mathbb{Z}_p$ instead of r_i/m_j's and when generating X'_{ij}'s, the delegator will use random values $r_{ij}^{(4)}$ in $\mathbb{Z}_p$ instead of r_i/n_j's. Because some values (r_i's) are used in both X and X', we treat all r_i/m_j and r_i/n_j as an instance of a single M-DDH problem. Then assuming that $(d_1 + 2d_2)$-$\mathsf{M\text{-}DDH}$ problem is hard, the difference in the adversary's view between $\mathbf{G_2}$ and $\mathbf{G_3}$ is negligible.

Game $\mathbf{G_4}$. The game is identical to $\mathbf{G_3}$ except that the delegator removes information about A_{ij} from each X_{ij} and X'_{ij}, which are now formed as $g_1^{r_{ij}^{(3)}}$ and $g_1^{r_{ij}^{(4)}}$ instead of $g_1^{A_{ij} r_{ij}^{(3)}}$ and $g_1^{A_{ij} r_{ij}^{(4)}}$, respectively. Because the $r_{ij}^{(3)}$'s and $r_{ij}^{(4)}$'s are completely random and all A_{ij}'s are non-zero, the distributions of X_{ij}'s and X'_{ij}'s in $\mathbf{G_3}$ are identical to their distributions in $\mathbf{G_4}$. Therefore, the adversary's view does not change between $\mathbf{G_3}$ and $\mathbf{G_4}$.

Now observe that we completely removed any information about the input matrices A and B from the adversary's view, which implies that the server's probability in winning in $\mathbf{G_4}$ is exactly $1/2$. Combined with the previous analysis of differences in the adversarial success between games $\mathbf{G_0}$ and $\mathbf{G_4}$, we obtain that the adversary's advantage is negligible in winning the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Priv}}(\mathsf{VC}_{lazy}^{p}, f, \kappa)$. $\square$

Theorem 7 *If the verification of the returned result was successful, H is a collision-resistant hash function, the M-DDH and XDH assumptions hold, and all elements of A are non-zero, the server must have followed the correct protocol and computed the correct matrix products $X \times Y$ and $X' \times Y'$.*

Proof The only difference in the adversary's view from VC_{lazy} is that matrices (A, B, Z) that the adversary receives are replaced with matrices (X', Y'). In either scheme, they are used to form $V^{(1)}$ and consequently construct the verification key $\hat{sk}$. Next, note that in both schemes the verification key is computed in the same way, but strictly less information about the data is revealed to the adversary in VC_{lazy}^{p} (i.e., X' and Y' do not reveal any information about the key and input matrices under out computational assumptions). Therefore, the adversary does not gain any advantage in passing verification without following the prescribed computation and the security claim holds in VC_{lazy}^{p} as well. $\square$

It is worth noting that we can easily modify VC_{lazy}^{p} to incorporate key chaining, which allows for efficient location of faulty cells when verification fails, by adding an additional layer of encoding to matrices Y and Y'.

7 Support for Public Verifiability

Recall that public delegatability means that at the time of problem generation the delegator produces a public verification key and after the server executes the computation delegated to it any entity with the knowledge of the public key is able to verify correctness of the outsourced computation. In this section, we incorporate public verifiability into VC_{mal} and VC_{lazy} constructions while preserving their original security properties. In particular, the client will now produce a public verification key PVK_x as a public version of SK_x at $\mathsf{ProbGen}$ time, which is roughly in the form of g^{SK_x}. This key then can be utilized by any auditing entity at Verify time to confirm correctness of outsourced computation upon its completion. We also show how the introduced modifications can be applied to privacy-preserving versions of schemes VC^p_{mal} and VC^p_{lazy} so that all features of public delegatability and verifiability, privacy protection, and computation verification will be supported in a single construction.

7.1 Support for Public Verifiability in Lazy Setting

Recall that in VC_lazy, SK_x produced at problem generation time 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. Thus, our scheme with public verifiability VC^{pv}_{lazy} can be described as follows:

1. In $\mathsf{ProbGen}$, the only difference is that in step 7 we set $\mathsf{PVK}_x = g^{sk}$ and the algorithm returns PVK_x instead of SK_x. Here, g is a generator of a cyclic group $\mathbb{G}$ in which the discrete logarithm problem is hard, and any $\mathbb{G}$ and g will suffice.

2. In Verify, the difference is that the input now includes PVK_x instead of SK_x and the verification consists of checking whether $\mathsf{PVK}_x = g^{\hat{sk}}$, where as before $\hat{sk}$ is part of output produced by $\mathsf{Compute}$.

The remaining algorithms, Setup and $\mathsf{Compute}$, are unchanged from VC_{lazy}.

Theorem 8 *Assuming that the M-DDH and XDH problems are hard, H is a collision-resistant hash function, and matrix A contains only non-zero elements, the verifiable computation scheme VC^{pv}_{lazy} is secure according to Definition 7.*

Proof There is only one difference in the adversary's view between VC_{lazy} and VC^{pv}_{lazy} in accordance to Definition 7: In VC^{pv}_{lazy}, the adversary can observe published $\mathsf{PVK}_x = g^{sk}$ for all of its queries. However, as the discrete logarithm problem is hard in $\mathbb{G}$, the adversary is unable to extract information about sk and therefore will not gain any non-negligible advantage in winning the security game $\mathbf{Exp}^{\mathsf{PubVer}}_{\mathcal{A}}(\mathsf{VC}^{pv}_{lazy}, f, \kappa)$. The proof for this construction proceeds in the same way to the proof of Theorem 2 and the security claim holds. $\square$

Because the release of PVK_x does not reveal enough information to compromise security of the scheme, we can apply the same modification to the privacy-preserving version VC^p_{lazy} of this scheme, without having to make additional adjustments. In the resulting construction, any verifier will be able to assess correctness of the performed computation using PVK_x, but will not be able to recover the output without access to additional information stored in SK_x. The delegator uses procedure Output to recover the produce matrix C.

7.2 Support for Public Verifiability in Malicious Setting

Unlike the previous solution, to achieve public verifiability in the malicious setting setting PVK_x to be g^v for every $v \in \mathsf{SK}_x$ does not work. Doing this would reveal crucial key information, which can be easily exploited by the server to compromise integrity of the computation. In particular, assume that PVK_x now consists $g_T^{r_i c_j}$ for $1 \le i \le d_1$ and $1 \le j \le d_3$ and $g_T^{\sum_{k=1}^{d_1} r_i v k_i}$, and Verify consists of

$\mathsf{Setup}(1^\kappa, f)$**:** The same as in VC_{mal}.

$\mathsf{ProbGen}(x = (A, B), \mathsf{params})$**:** On input two matrices A and B of respective dimensions $d_1 \times d_2$ and $d_2 \times d_3$, perform:

1. Choose $r_i \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le i \le d_1$, $d_j \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le j \le d_2$, and $c_k, f_k \overset{R}{\leftarrow} \mathbb{Z}_p^*$ for $1 \le k \le d_3$.
2. Compute $t_i = \sum_{k=1}^{d_2} A_{ik} d_k$ for $1 \le i \le d_1$, $f = \sum_{j=1}^{d_3} f_j$, $vk_i = t_i f$ for $1 \le i \le d_1$, and $vk = \sum_{i=1}^{d_1} r_i vk_i$.
3. Compute $Y_{ij} = g_2^{c_j B_{ij} + f_j d_i}$ for $1 \le i \le d_2$ and $1 \le j \le d_3$.
4. Set $\mathsf{PVK}_x = (\{g_1^{r_i}\}_{i=1}^{d_1}, \{g_T^{r_i c_j}\}_{i=1,j=1}^{d_1,d_3}, g_T^{vk})$, $\mathsf{EK}_x = \mathsf{params}$, and $\sigma_x = (A, B, Y)$.

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

1. Compute $C = A \times B$.
2. Compute $s_i = \prod_{j=1}^{d_3} \prod_{k=1}^{d_2} Y_{kj}^{A_{ik}}$ for $1 \le i \le d_1$.
3. Set $\sigma_y = (C, \{s_i\}_{i=1}^{d_1})$.

$\mathsf{Verify}(\mathsf{PVK}_x = (\{g_1^{r_i}\}_{i=1}^{d_1}, \{g_T^{r_i c_j}\}_{i=1,j=1}^{d_1,d_3}, g_T^{vk}), \sigma_y = (C, \{s_i\}_{i=1}^{d_1}))$**:** Verify whether $g_T^{vk} \prod_{i=1}^{d_1} \prod_{j=1}^{d_3} (g_T^{r_i c_j})^{C_{ij}} = \prod_{i=1}^{d_1} e(g_1^{r_i}, s_i)$. If the check succeeds, output C; otherwise, output $\perp$.

Figure 6: Description of scheme VC_{mal}^{pv} in malicious setting that supports public verifiability.

checking whether $\prod_{i=1}^{d_1} \prod_{j=1}^{d_3} (g_T^{r_i c_j})^{C_{ij}} \cdot g_T^{\sum_{k=1}^{d_1} r_i vk_i} = s$ holds (where s is returned by $\mathsf{Compute}$).[1] To launch an attack, the adversary first correctly computes matrix C and s, then randomly chooses one matrix element C_{ij}, changes it to $C_{ij} + \delta$, and multiplies s by $g_T^{r_i c_j \delta}$. By doing that, the adversary is able to produce a tuple $(\hat{C}, \hat{s})$ that differs from the correct one (C, s), but nevertheless passes the verification test. Notice that this attack is infeasible in the original VC_{mal} scheme as the adversary has no information about $g_T^{r_i c_j}$ and hence is unable to correctly produce $\hat{s}$. Therefore, to support public verifiability in the malicious setting, we need to introduce further changes to the scheme.

We refer to the new scheme with public verifiability as VC_{mal}^{pv}. The idea behind the modification is to prevent the server from merging offset $g_T^{r_i c_j \delta}$ with s by making the values belong to two different groups. That is, we design the scheme in a way that the values of the form $g^{r_i c_j}$ are available only in $\mathbb{G}_T$, while s will have to be produced by the server as an element of $\mathbb{G}_2$. Because groups $\mathbb{G}_2$ and $\mathbb{G}_T$ are used in the bilinear function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, there does not exist an efficiently computable conversion from an element g_T^x in $\mathbb{G}_T$ to the corresponding element g_2^x in $\mathbb{G}_2$. Then in the construction the delegator produces PVK_x that consists of $g_1^{r_i}$, $g_T^{r_i c_j}$ and $g_T^{\sum_{i=1}^{d_1} r_i vk_i}$ and provides σ_x that consists of only three input matrices A, B, and Y (rather than four in VC_{mal}) to the server. The server computes each s_i to be an element of $\mathbb{G}_2$ by performing a modulo exponentiation using A and Y (rather than the pairing operation in VC_{mal}) and returns their values 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 $g_T^{\sum_{i=1}^{d_1} r_i vk_i}$ 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. A detailed description of the new scheme VC_{mal}^{pv} is given in Figure 6.

[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 $1 \le i \le d_1$ and $1 \le j \le d_3$ and $g_T^{\sum_{k=1}^{d_1} r_i vk_i}$ represent the minimum information the verifier needs to possess to carry out the verification procedure.

The complexity of $\mathsf{ProbGen}$ is dominated by computing public verification key PVK_x and matrix Y, and is $O(d_1d_2 + d_1d_3 + d_2d_3)$. $\mathsf{Compute}$ involves matrix multiplication and has complexity $O(d_1d_2d_3)$. Verify consists of ensuring the validity of C by checking its elements against s_i that $\mathsf{Compute}$ produces and involves d_1d_3 modulo exponentiations and d_1 pairing operations.

Theorem 9 *Assuming that the M-DDH and XDH problems are hard, the verifiable computation scheme with public verifiability* VC^{pv}_{mal} *is secure according to Definition 7.*

Proof Similar to previous proofs, we proceed with a series of games using the hybrid argument and analyze the adversary's capabilities in the final game.

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

Game $\mathbf{G_1}$. The game is identical to game $\mathbf{G_0}$, except that when generating Y_{ij} and vk in g_T^{vk}, the delegator will use a random value $r^{(1)}_{ij}$ from $\mathbb{Z}_p$ instead of f_jd_i. Assuming that $(d_2 + d_3)$-M-DDH problem is hard, the difference in the adversary's view between $\mathbf{G_1}$ and $\mathbf{G_0}$ is negligible.

Game $\mathbf{G_2}$. We next remove information about c_j's from the elements of Y. That is, we replace each $Y_{ij} = g_2^{r^{(1)}_{ij}+c_jB_{ij}}$ with $g_2^{r^{(1)}_{ij}}$. Because each $r^{(1)}_{ij}$ is random, the distribution of Y_{ij}'s will not be affected. The $r^{(1)}_{ij}$'s, however, are also used in vk and thus we also update g_T^{vk} to be consistent with other information. That is, g_T^{vk} from $\mathbf{G_1}$ is multiplied by $g_T^{-\sum_{i=1}^{d_1}\sum_{j=1}^{d_3}\sum_{k=1}^{d_2} r_ic_jA_{ik}B_{kj}}$ to form $g_T^{vk} = g_T^{\sum_{i=1}^{d_1}\sum_{j=1}^{d_3}\sum_{k=1}^{d_2} r_iA_{ik}(r^{(1)}_{kj}-c_jB_{kj})}$ in $\mathbf{G_2}$.

Game $\mathbf{G_3}$. In this game, we apply another instance of M-DDH problem. This time, we replace all instances of r_ic_j (in $g_T^{r_ic_j}$) with random values from $\mathbb{Z}_p$, which we denote by $r^{(2)}_{ij}$. That is, now each $g_T^{r_ic_j}$ in PVK_x is replaced with $g_T^{r^{(2)}_{ij}}$ and g_T^{vk} is replaced with $g_T^{\sum_{i=1}^{d_1}\sum_{j=1}^{d_3}\sum_{k=1}^{d_2} A_{ik}(r_ir^{(1)}_{kj}-r^{(2)}_{ij}B_{kj})}$. Assuming that $(d_3 + d_1)$-M-DDH problem is hard, the difference in the adversary's view between $\mathbf{G_2}$ and $\mathbf{G_1}$ is negligible. Note that the M-DDH assumption holds when $g_T^{c_i}$ and $g_T^{r_i}$ are available to the adversary. In our case, the adversary has access to $g_1^{r_i}$ and therefore $g_T^{r_i}$, but no information about $g_2^{c_j}$ is available, which means that we can safely apply the M-DDH assumption.

Now, suppose that in game $\mathbf{G_3}$ the server is able to return output $(\hat{C}, \{\hat{s}_i\}_{i=1}^{d_1})$ that differs from the expected $(C, \{s_i\}_{i=1}^{d_1})$, but which nevertheless passes verification. According to the security definition, the adversary wins if $\hat{C} \neq C$, but verification is successful. This means that the returned values must satisfy $\prod_{i=1}^{d_1}\prod_{j=1}^{d_3}(g_T^{r^{(2)}_{ij}})^{C_{ij}-\hat{C}_{ij}} = \prod_{i=1}^{d_1} e(g_1^{r_i}, s_i - \hat{s}_i)$, where $C_{ij} \neq \hat{C}_{ij}$ for at least one value of i and j.

Now note that the server has no information regarding $r^{(2)}_{ij}$'s beyond the released $g_T^{r^{(2)}_{ij}}$'s and thus he has a negligible probability of making $\sum_{i=1}^{d_1}\sum_{j=1}^{d_3} r^{(2)}_{ij}(C_{ij} - \hat{C}_{ij})$ equal to any given value beyond his control. Furthermore, the only information available to the server in $\mathbb{G}_2$ are random elements of the group $g_2^{r^{(1)}_{ij}}$ (which in particular means that the $\hat{s}_i$'s cannot depend on $r^{(2)}_{ij}$'s as the server is unable to use $g_T^{r^{(2)}_{ij}}$'s from $\mathbb{G}_T$ to form $\hat{s}_i$'s in $\mathbb{G}_2$). This means that in this game the server must guess a function of random elements from $\mathbb{Z}_p$ when forming $\hat{C} \neq C$, the probability of which is negligible in the security parameter and thus the probability of winning the game. Combined with the previous analysis of the differences in the adversarial success between games $\mathbf{G_0}$ and $\mathbf{G_3}$, we obtain that $\mathcal{A}$'s advantage is negligible in winning the experiment $\mathbf{Exp}^{\mathsf{PubVer}}_{\mathcal{A}}(\mathsf{VC}^{pv}_{mal}, f, \kappa)$, as desired. $\square$

While the above result already states security of the scheme, we can prove an even stronger property which guarantees that all information returned by the server, i.e., both matrix C and vector s, is correct. This is opposed to only guaranteeing that the output of the computation C is correct. To show the strengthen result, we rely on a new cryptographic assumption that we call k-Computational Linear Aggregation (CLA) assumption and which we prove secure in the generic group model. We obtain the following:

Theorem 10 *Assuming that the M-DDH, XDH, and CLA problems are hard, the verifiable computation scheme with public verifiability* VC_{mal}^{pv} *is secure according to Definition 7 and additionally guarantees that all information returned by malicious server in* σ_y *is correct.*

The details of the k-CLA assumption, its proof of security in the generic model, and the proof of Theorem 10 can be found in Appendix B.

7.3 Support for Public Verifiability and Data Privacy in Malicious Setting

Recall that it was easy to add data privacy to scheme VC_{mal} because the computation consisted of computing products $A \times B$ and $X \times Y$. These products can be computed independently and therefore we could encode matrices A and B using any homomorphic encryption scheme that supports one multiplication and many additions on encrypted data. In VC_{mal}^{pv}, on the other hand, matrix X is not used, but instead both products utilize A. Thus, if we encrypt input matrices A and B in order to protect the data from the computational server and the auditor, the client will need to incorporate randomness used during forming ciphertexts into public verification key PVK_x, which makes the use of encryption more challenging. More importantly, to produce s_i's to be in $\mathbb{G}_2$, we need the elements of A to take values from $\mathbb{Z}_p$ because Y_{ij} (which are also used in the computation of s_i's) are already elements of $\mathbb{G}_2$. To meet the above requirements, instead of using encryption, we additively split each element of matrix A into two shares to obtain matrices A^0 and A^1 that will be distributed to independent and non-colluding servers. The same is applied to matrix B. The total of four servers are used. Each server then performs the same computation as in VC_{mal}^{pv} on its shares of matrices A and B and each result is verified independently. When the client receives the results from all servers, it can reconstruct the output from its shares. More specifically, the solution proceeds as follows:

1. The Setup procedure is the same as in VC_{mal}^{pv}.
2. At $\mathsf{ProbGen}$ time, the client performs the following steps:
   (a) Generate a $d_1 \times d_2$ matrix R with its elements randomly drawn from $\{0,1\}^{\ell+\rho}$, where ℓ is the bitlength of the elements of A and B and ρ is a statistical security parameter. Use matrix R to split matrix A into A^0 and A^1 with their elements computed as $A^0_{ij} = R_{ij}$ and $A^1_{ij} = A_{ij} - R_{ij}$, respectively. Similarly, generate a $d_2 \times d_3$ matrix R' with its elements drawn at random from $\{0,1\}^{\ell+\rho}$ and split matrix B into B^0 and B^1 as $B^0_{ij} = R'_{ij}$ and $B^1_{ij} = B_{ij} - R'_{ij}$.
   (b) Set EK_x computed as in VC_{mal}^{pv}, and $\sigma_x = (A^0, B^0, Y^0)$ for the first server, $\sigma_x = (A^1, B^0, Y^0)$ for the second server, $\sigma_x = (A^0, B^1, Y^1)$ for the third server, and $\sigma_x = (A^1, B^1, Y^1)$ for the fourth server, where Y^i denotes the matrix Y computed according to the elements of B^i. PVK_x is also computed in the same way as in VC_{mal}^{pv} with the exception that g_T^{vk} is computed according to A^0 in PVK_x for the first and third servers and according to A^1 for the second and fourth servers.

3. For $\mathsf{Compute}$, each server performs the same computation as in VC^{pv}_{mal} on its respective inputs.

4. During Verify, an auditor uses σ_y returned by an individual server and the corresponding PVK_x to verify the returned result in the same way as in VC^{pv}_{mal}.

5. During Output, the client receives σ_y from each of the four servers. Let C^i denote the product matrix returned by server i. The client then recovers C by computing $C^1 + C^2 + C^3 + C^4 = A \times B$.

Theorem 11 *Assuming the M-DDH and XDH problems are hard and independent and non-colluding servers and auditors are used, the above verifiable computation scheme with public verifiability and data secrecy* $\mathsf{VC}^{pv,p}_{mal}$ *is a secure according to Definitions 6 and 7.*

The proof of this theorem is straightforward. That is, VC^{pv}_{mal} is used without changes, which means that $\mathsf{VC}^{pv,p}_{mal}$ is also a verifiable computation scheme with public verifiability. Data privacy follows from the fact that a server or an auditor only obtain access to statistically protected data and the unprotected data is never used in the computation. Thus, the probability of the adversary learning any information about the data is negligible in the statistical security parameter ρ.

8 Conclusions

This work presents schemes for verifiable outsourcing of matrix multiplications, where it is assumed that the server to which the computation is being outsourced can either arbitrarily deviate from the prescribed computation or is rational, or lazy, in the sense that it will perform only the minimum amount of work to pass the verification but will not maliciously corrupt the returned result. The complexity and features of our schemes favorably compare to the state of the art, with the solution in the lazy setting having lower 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

We would like to thank Keith Frikken on his comments regarding this work. 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 *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, ACNS '09, pages 292–305, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Proceedings of the 37th International Colloquium Conference on Automata, Languages and Programming*, ICALP'10, pages 152–163, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] M. Atallah and K. Frikken. Securely outsourcing linear algebra computations. In *ASIACCS*, pages 48–59, 2010.

[4] M. Atallah, K. Frikken, and S. Wang. Private outsourcing of matrix multiplication over closed semi-rings. In *SECRYPT*, pages 136–144, 2012.

[5] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, New York, NY, USA, 1985. ACM.

[6] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, CCS '13, pages 863–874, New York, NY, USA, 2013. ACM.

[7] 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*, ITCS '12, pages 326–349, New York, NY, USA, 2012. ACM.

[8] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, STOC '13, pages 111–120, New York, NY, USA, 2013. ACM.

[9] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'11, pages 149–168, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Proceedings of the 14th International Conference on Practice and Theory in Public-Key Cryptography*, PKC'11, pages 1–16, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] D. Catalano and D. Fiore. Practical homomorphic macs for arithmetic circuits. In *Proceedings of the 32nd Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'13, pages 336–352, Berlin, Heidelberg, 2013. Springer-Verlag.

[12] D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography*, TCC'13, pages 680–699, Berlin, Heidelberg, 2013. Springer-Verlag.

[13] D. Catalano, D. Fiore, and B. Warinschi. Efficient network coding signatures in the standard model. In *Proceedings of the 15th International Conference on Practice and Theory in Public-Key Cryptography*, PKC'12, pages 680–696, Berlin, Heidelberg, 2012. Springer-Verlag.

[14] K. Chung, Y. T. Kalai, F. Liu, and R. Raz. Memory delegation. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 151–165, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 483–501, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[17] 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*, CCS '12, pages 501–512, New York, NY, USA, 2012. ACM.

[18] R. Gennaro and D. Wichs. Fully homomorphic message authenticators. Cryptology ePrint Archive, Report 2012/290, 2012.

[19] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 465–482, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, STOC '11, pages 99–108, New York, NY, USA, 2011. ACM.

[21] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, STOC'09, pages 169–178, New York, NY, USA, 2009. ACM.

[22] S. Goldberg, S. Halevi, A. D. Jaggard, V. Ramachandran, and R. N. Wright. Rationality and traffic attraction: Incentives for honest path announcements in bgp. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 267–278, New York, NY, USA, 2008. ACM.

[23] 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*, STOC '08, pages 113–122, New York, NY, USA, 2008. ACM.

[24] S. Goldwasser, S. Micali., and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, Feb 1989.

[25] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, STOC '04, pages 623–632, New York, NY, USA, 2004. ACM.

[26] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In *Proceedings of the Cryptographer's Track at the RSA Conference*, CT-RSA '02, pages 244–262, London, UK, UK, 2002. Springer-Verlag.

[27] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, STOC'92, pages 723–732, New York, NY, USA, 1992. ACM.

[28] J. Kilian. Improved efficient arguments. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'95, pages 311–324, London, UK, 1995. Springer-Verlag.

[29] M. H. Manshaei, Q. Zhu, T. Alpcan, T. Bacşar, and J. Hubaux. Game theory meets network security and privacy. *ACM Computing Surveys*, 45(3):25:1–25:39, 2013.

[30] S. Micali. Cs proofs. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS'94, pages 436–453, Washington, DC, USA, 1994. IEEE Computer Society.

[31] P. Mohassel. Efficient and secure delegation of linear algebra. IACR Cryptology ePrint Archive Report 2011/605, 2011.

[32] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 238–252, Washington, DC, USA, 2013. IEEE Computer Society.

[33] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Proceedings of the 9th International Conference on Theory of Cryptography*, TCC'12, pages 422–439, Berlin, Heidelberg, 2012. Springer-Verlag.

[34] S. Setty, , N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 253–268, Berkeley, CA, USA, 2012. USENIX Association.

[35] S. Setty, R. McPherson, A. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *Proceedings of the Network and Distributed Systems Security Symposium*, NDSS'12, 2012.

[36] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 256–266, Berlin, Heidelberg, 1997. Springer-Verlag.

[37] M. D. Van, A. Juels, A. Oprea, R. L. Rivest, E. Stefanov, and N. Triandopoulos. Hourglass schemes: How to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 265–280, New York, NY, USA, 2012. ACM.

[38] V. Vu, S. Setty, A. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP'13, Berkeley, CA, USA, 2013. IEEE Computer Society.

A Additional Definitions

We present the security experiment for a verifiable scheme with public verifiability:

```
Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubVer}}(\mathsf{VC}, f, \kappa)$
    $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, f)$
    for $i = 1$ to $q$ do
        $x_i \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \mathsf{PVK}_1, \ldots, \sigma_{x_{i-1}}, \mathsf{EK}_{i-1}, \mathsf{PVK}_{i-1})$
        $(\mathsf{SK}_i, \mathsf{PVK}_i, \mathsf{EK}_i, \sigma_{x_i}) \leftarrow \mathsf{ProbGen}(x_i, \mathsf{params})$
    $x^* \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \mathsf{PVK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q, \mathsf{PVK}_q)$
    $(\mathsf{SK}_{x^*}, \mathsf{PVK}_{x^*}, \mathsf{EK}_{x^*}, \sigma_{x^*}) \leftarrow \mathsf{ProbGen}(x^*, \mathsf{params})$
```

```
$\sigma'_y \leftarrow \mathcal{A}(\sigma_{x_1}, \mathsf{EK}_1, \mathsf{PVK}_1, \ldots, \sigma_{x_q}, \mathsf{EK}_q, \mathsf{PVK}_q, \sigma_{x^*}, \mathsf{EK}_{x^*}, \mathsf{PVK}_{x^*})$
$y' \leftarrow \mathsf{Verify}(\mathsf{PVK}_{x^*}, \sigma_y)$
if $y' \neq \perp$ and $y' \neq f(x^*)$ return 1
else return 0
```

B Stronger Security of VC^{pv}_{mal} and k-CLA Assumption

In this section, we introduce a new assumption, called k-Computational Linear Aggregation (k-CLA) assumption, show its security in the generic group model, and then show that utilizing it will allow us to prove a stronger security property of the verifiable computation scheme with public verifiability in presence of malicious adversaries VC^{pv}_{mal} as described in Section 7.2.

The k-CLA assumption is defined as follows:

Definition 8 (k-CLA assumption) *Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow Set(1^\kappa)$. Given $R = (g_1^{r_1}, g_1^{r_2}, \ldots, g_1^{r_k})$ with random $r_i \in \mathbb{Z}_p^*$, the adversary $\mathcal{A}$ needs to produce $T = (g_2^{t_1}, g_2^{t_2}, \ldots, g_2^{t_k})$ with $t_i \in \mathbb{Z}_p^*$ that satisfy the equation $\sum_{i=1}^{k} r_i t_i \equiv 0 \pmod{p}$. $\mathcal{A}$'s advantage in solving the k-CLA problem is defined as*

$$\boldsymbol{Adv}_{\mathcal{A}}^{k\text{-}\mathsf{CLA}}(\kappa) = \Pr[\mathcal{A}(p, g_1, g_2, k, e, D) = 1].$$

We say that the k-CLA assumption holds if for every PPT algorithm $\mathcal{A}$, $\boldsymbol{Adv}_{\mathcal{A}}^{k\text{-}\mathsf{CLA}}(\kappa)$ is negligible in κ.

Lemma 2 *In the generic group model, the probability for an adversary who issues q operations to solve the k-CLA problem is at most $(k^2 + 4q^2 + 2kq + k + 4)/2p$.*

Proof Given public information p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, g_1, g_2, e, and k, we keep track of three lists, $L_1 = \{(x_i, \sigma_1(x_i))\}$, $L_2 = \{(y_i, \sigma_2(y_i))\}$, and $L_3 = \{(z_i, \sigma_3(z_i))\}$ with each of them maintaining the queries performed on cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively. Initially, L_1 contains $k+1$ encodings corresponding to the group generator g_1 and inputs $g_1^{r_i}$ for $1 \leq i \leq k$, L_2 contains a single encoding corresponding to the group generator g_2, and L_3 is empty. The three lists will be updated according to the procedures listed below:

- *Group Action in $\mathbb{G}_1$:* The generic algorithm specifies two values x_i and x_j of the current encoding list L_1 and a sign bit, and the oracle computes $x' = x_i + x_j$ or $x' = x_i - x_j$ according to the sign bit. If $(x', \sigma_1(x')) \in L_1$, the oracle will return $\sigma_1(x')$ to the algorithm; otherwise, the oracle will choose a new value for $\sigma_1(x')$, append it to L_1, and return it to the algorithm.
- *Group Action in $\mathbb{G}_2$:* This procedure is the same as for $\mathbb{G}_1$, except that the queried inputs are encoded using function σ_2 and list L_2 is used instead of L_1.
- *Group Action in $\mathbb{G}_T$:* This procedure is the same as for $\mathbb{G}_1$, except that the queried inputs are encoded using function σ_3 and list L_3 is used instead of L_1.
- *Bilinear Map:* The generic algorithm specifies two values x_i and y_j from lists L_1 and L_2, respectively, and the oracle will compute $z_{ij} = x_i \cdot y_j$. If $(z_{ij}, \sigma_3(z_{ij})) \in L_3$, the oracle will return $\sigma_3(z_{ij})$ to the algorithm; otherwise, the oracle will choose a new value for $\sigma_3(z_{ij})$, append it to L_3, and return it to the algorithm.

After performing q operations based on the above rules, the algorithm outputs a list $T = (\sigma_2(y_1), \sigma_2(y_2), \ldots, \sigma_2(y_k))$. Because the encodings for the elements in each cyclic group are represented as random polynomials, their degree will be at most 1 in L_1 and L_2 and at most 2 in L_3. Furthermore, it was shown in [36] that (i) a non-zero polynomial with degree d in the group of order p will be evaluated to 0 with probability at most d/p, and (ii) two unequal polynomials of degree d instantiated with random values for their variables will be evaluated to the same value with probability at most d/p. Next, we analyze the probability of simulation failure that occurs when the oracle outputs two different encodings for the same element in the group, namely two unequal polynomials in L_1, L_2, and L_3 evaluate to the same value when the variables are assigned random values from $\mathbb{Z}_p$. There may occur four inconsistencies:

1. The first inconsistency may occur in the encodings of elements in $\mathbb{G}_1$, i.e., when two distinct polynomials originated from the encoding σ_1 evaluate to the same output on inputs chosen uniformly at random. Based on [36], the probability that this happens is at most $1/p$. Then if τ_1 queries were issued in $\mathbb{G}_1$, there will be $(k + \tau_1 + 1)$ pairs in L_1. The probability that this inconsistency occurs for any element in L_1 is $\binom{\tau_1+k+1}{2} = \frac{(k+\tau_1+1)(k+\tau_1)}{2p} \leq \frac{(k+q+1)(k+q)}{2p}$.

2. The second inconsistency may occur in the encodings of elements in $\mathbb{G}_2$. Similar to the previous case, the probability of this occurring for any two elements in L_2 is $1/p$. If τ_2 queries were issued in $\mathbb{G}_2$, there will be $(\tau_2 + 1)$ pairs in L_2, and the probability of this inconsistency occurring for any element in L_2 is $\binom{\tau_2+1}{2} = \frac{(\tau_2+1)\tau_2}{2p} \leq \frac{q(q+1)}{2p}$.

3. The third inconsistency may occur in the encodings of elements in $\mathbb{G}_T$. Unlike the previous two cases, the degree of polynomials for the encoding σ_3 has degree at most 2, and therefore the probability that two specific polynomials result in this inconsistency is $\leq 2/p$. When τ_3 queries were issued in $\mathbb{G}_T$, there will be (τ_3) pairs in L_3, and the probability of this inconsistency occurring for any element in L_3 is $\binom{\tau_3}{2} = \frac{\tau_3(\tau_3-1)}{p} \leq \frac{q(q-1)}{p}$.

4. Finally, we would like to show that it is unlikely that polynomials originated from encodings σ_1 and σ_2 satisfy the equation $\sum_{i=1}^{k} \sigma_1(x_i)\sigma_2(y_i) = 0 \pmod{p}$, which implies that the answers returned by the adversary do not satisfy the k-CLA problem. In particular, the polynomial $\sum_{i=1}^{k} \sigma_1(x_i)\sigma_2(y_i)$ has degree at most 2 and will not be a zero-polynomial. Therefore, when the variables are instantiated with random values, the probability for the polynomial to be evaluated to zero is $\leq 2/p$. This means that when the adversary returns a single assignment to solve the k-CLA problem, the probability for the occurrence of any inconsistency in the assignment will be $\leq 2/p$.

We obtain that, based on the union bound, the probability for any of the above inconsistencies to occur is $\leq (k^2 + 4q^2 + 2kq + k + 4)/2p$ □.

Proof of Theorem 10 Consider the setup used in the proof of Theorem 9. It was already shown that if the server modifies matrix product C, verification will be successful only with a negligible probability. To prove the stronger statement of this theorem, we now need to show that if the server returns correct matrix C, but incorrect values for s_i's, it is also unable to pass the verification with more than a negligible probability.

Suppose that the server returns correct matrix C. In order for the returned values to deviate from the correct values, there should exist at least a single $\hat{s}_i$ for some $1 \leq i \leq d_1$ that is not equal to s_i. Let the number of $\hat{s}_i$'s that are not not equal to the corresponding s_i's to be k for some $1 \leq k \leq d_1$. Because the verification procedure checks for $g_T^{vk} \prod_{i=1}^{d_1} \prod_{j=1}^{d_3} \left(g_T^{r_i c_j}\right)^{C_{ij}} = \prod_{i=1}^{d_1} e(g_1^{r_i}, s_i)$, we obtain that in order to pass verification, it must hold that $\prod_{i=1}^{k} e(g_1^{r_i}, \Delta s_i) = g_T^0$, where Δs_i is the difference between s_i and $\hat{s}_i$. Recall that each s_i is in the form $g_2^{u_i}$ for some u_i and thus Δs_i can be written as $g_2^{u_i - u'_i} = g_2^{t_i}$. Now, given $g_1^{r_i}$ for $i = 1, \ldots, d_1$, the server needs to compute g^{t_i} for $1 \leq i \leq k$ such that each t_i is non-zero and the equation $\prod_{i=1}^{k} e(g_1^{r_i}, g_2^{t_i}) = g_T^0$ holds. Note that this is exactly what the k-CLA problem is. Thus, assuming that d_1-CLA assumption is hard, we obtain that the server will not be able to corrupt any information in the output without being noticed with more than a negligible probability. $\square$