Private Outsourcing of Polynomial Evaluation and Matrix Multiplication using Multilinear Maps

Liang Feng Zhang and Rehaneh Safavi-Naini
Institute for Security, Privacy and Information Assurance
Department of Computer Science
University of Calgary
{liangf.zhang, rei.safavi}@gmail.com

Abstract

Verifiable computation (VC) allows a computationally weak client to outsource the evaluation of a function on many inputs to a powerful but untrusted server. The client invests a large amount of off-line computation and gives an encoding of its function to the server. The server returns both an evaluation of the function on the client’s input and a proof such that the client can verify the evaluation using substantially less effort than doing the evaluation on its own. We consider how to privately outsource computations using privacy preserving VC schemes whose executions reveal no information on the client’s input or function to the server. We construct VC schemes with input privacy for univariate polynomial evaluation and matrix multiplication and then extend them such that the function privacy is also achieved. Our tool is the recently developed multilinear maps. The proposed VC schemes can be used in outsourcing private information retrieval (PIR).

1 Introduction

The rise of cloud computing in recent years has made outsourcing of storage and computation a reality with many cloud service providers offering attractive services. Large computation can hugely impact resources (e.g. battery) of weak clients. Outsourcing computation removes this bottleneck but also raises a natural question: how to assure the computation is carried out correctly as the server is untrusted. This assurance is not only against malicious behaviors but also infrastructure failures of the server. Verifiable computation (VC) [16] provides such assurances for a large class of computation delegation scenarios. The client in this model invests a large amount of off-line computation and generates an encoding of its function \( f \). Given the encoding and any input \( \alpha \), the server computes and responds with \( y \) and a proof that \( y = f(\alpha) \). The client can verify if the computation has been carried out correctly using substantially less effort than computing \( f(\alpha) \) on its own. In particular, the client’s off-line computation cost is amortized over the evaluations of \( f \) on multiple inputs \( \alpha \).

VC schemes were formally defined by Gennaro, Gentry and Parno [16] and then constructed for a variety of computations [11, 3, 25, 2, 23, 13, 12]. We say that a VC scheme is privacy preserving if its execution reveals no information on the client’s input or function to the server. Protecting the client’s input and function from the server is an essential requirement in many real-life scenarios. For example, a health professional querying a database of medical records may need to protect both the identity and the record of his patient. VC schemes with input privacy have been considered in [16, 2] where a generic function is written as a circuit, and each gate is evaluated using a fully homomorphic encryption scheme (FHE). These VC schemes evaluate the outsourced functions as circuits and are not very efficient. Furthermore, the outsourced functions are given to the server in clear and therefore the function privacy is not achieved. Benabbas, Gennaro and Vahlis [3] and several other works [13, 12, 23] design VC schemes for specific functions without using FHE. Some of them even achieve function privacy. However, they do not consider the input privacy.
1.1 Results and Techniques

In this paper, we consider privacy preserving VC schemes for specific function evaluations without using FHE. The function evaluations we study include univariate polynomial evaluation and matrix multiplication. Our privacy definition is indistinguishability based and guarantees no untrusted server can distinguish between different inputs or functions of the client. In privacy preserving VC schemes both the client’s input and function must be hidden (e.g., encrypted) from the server and the server must evaluate the hidden function on the hidden input and then generate a proof that the evaluation has been carried out correctly. Note that such a proof can be generated using the non-interactive proof or argument systems from [22, 4] but they require the use of either random oracles or knowledge of exponent (KoE) type assumptions, both of which are considered as strong [23] and have been carefully avoided by the VC literatures [16, 3, 25].

We construct VC schemes for univariate polynomial evaluation and matrix multiplication that achieve input privacy and then extend them such that the function privacy is also achieved. Our main tool is the multilinear maps [14, 15]. Very recently, Garg, Gentry, and Halevi [14] proposed a candidate mechanism that would approximate or be the moral equivalent of multilinear maps for many applications. In [15], it was believed that the mechanism opens an exciting opportunity to study new constructions using a multilinear map abstraction. Following [15], we use a framework of leveled multilinear maps where one can call a group generator \( G(1^k) \) to obtain a sequence of groups \( G_1, \ldots, G_k \) of order \( N \) along with their generators \( g_1, \ldots, g_k \), where \( N = pq \) for two \( \lambda \)-bit primes \( p \) and \( q \). Slightly abusing notation, if \( i + j \leq k \), we can compute a bilinear map operation on \( g_i^a \in G_i, g_j^b \in G_j \) as \( e(g_i^a, g_j^b) = g_{i+j}^{ab} \). These maps can be seen as implementing a \( k \)-multilinear map. We denote by

\[
\Gamma_k = (N, G_1, \ldots, G_k, e, g_1, \ldots, g_k) \leftarrow G(1^k)
\]  

(1)
a random \( k \)-multilinear map instance, where \( N = pq \) for two \( \lambda \)-bit primes \( p \) and \( q \). We start with the BGN encryption scheme (denoted by BGN2) of Boneh, Goh, and Nissim [6] which is based on \( \Gamma_2 \) and semantically secure when the subgroup decision assumption (abbreviated as SDA, see Definition 2.1) for \( \Gamma_2 \) holds. It is well-known that BGN2 is both additively homomorphic and multiplicatively homomorphic, i.e., given BGN2 ciphertexts \( \text{Enc}(m_1) \) and \( \text{Enc}(m_2) \) one can easily compute \( \text{Enc}(m_1 + m_2) \) and \( \text{Enc}(m_1 m_2) \). Furthermore, BGN2 supports an unlimited number of additive homomorphic operations: for any integer \( k \geq 2 \), given BGN2 ciphertexts \( \text{Enc}(m_1), \ldots, \text{Enc}(m_k) \) one can easily compute \( \text{Enc}(m_1 + \cdots + m_k) \). As a result, one can easily compute \( \text{Enc}(f(\alpha)) \) from \( \text{Enc}(\alpha) \) for any quadratic polynomial \( f(x) \). On the other hand, BGN2 supports only one multiplicative homomorphic operation: one cannot compute \( \text{Enc}(m_1 m_2 m_3) \) from \( \text{Enc}(m_1), \text{Enc}(m_2) \) and \( \text{Enc}(m_3) \). In particular, one cannot compute \( \text{Enc}(f(\alpha)) \) from \( \text{Enc}(\alpha) \) for any polynomial \( f(x) \) of degree \( \geq 3 \). In Section 2.2, we introduce BGN\(_k\), which is a generalization of BGN2 over \( \Gamma_k \) and semantically secure under the SDA for \( \Gamma_k \). BGN\(_k\) supports both an unlimited number of additive homomorphic operations and up to \( k - 1 \) multiplicative homomorphic operations. As a result, it allows us to compute \( \text{Enc}(f(\alpha)) \) from \( \text{Enc}(\alpha) \) for any degree-\( k \) polynomial \( f(x) \). In our VC schemes, the client’s input and function are encrypted using BGN\(_k\) for a suitable \( k \) and the server computes on the ciphertexts.

**Polynomial evaluation.** In Section 3.1 we propose a VC scheme \( \Pi_{pe} \) with input privacy (see Fig. 2) that allows the client to outsource the evaluation of a degree \( n \) polynomial \( f(x) \) on any input \( \alpha \) from a polynomial size domain \( D \). In [20], the algebraic property that there is a polynomial \( c(x) \) of degree \( n - 1 \) such that \( f(x) - f(\alpha) = (x - \alpha)c(x) \) was applied to construct polynomial commitment schemes. Those schemes actually give us a basic VC scheme for univariate polynomial evaluation but without input privacy. Let \( e : G_1 \times G_1 \rightarrow G_2 \) be a bilinear map, where \( G_1 \) and \( G_2 \) are cyclic groups of prime order \( p \) and \( G_1 \) is generated by \( g_1 \). In the basic VC scheme, the client makes public \( t = g_1^{f(s)} \) and gives \( pk = (g_1, g_1^s, \ldots, g_1^n, f(x)) \) to the server, where \( s \) is uniformly chosen from \( \mathbb{Z}_p \). To verifiably compute \( f(\alpha) \), the client gives \( \alpha \) to the server and the server returns \( \rho = f(\alpha) \) along with a proof \( \pi = g_1^\rho \). Finally the client verifies if \( e(t/g_1^\rho, g_1) = e(g_1^\rho/g_1^\alpha, \pi) \). The basic VC scheme is secure under the SBDH assumption [20]. It has been generalized in [23] to construct VC schemes for multivariate polynomial
evaluation.

In \( \Pi_{pe} \), the \( \alpha \) should be hidden from the server (e.g., the client gives \( \text{Enc}(\alpha) \) to the server) which makes the server’s computation of \( \rho \) and \( \pi \) (as in the basic VC scheme) impossible. Instead, the best one can expect is to compute a ciphertext \( \rho = \text{Enc}(f(\alpha)) \) from \( \text{Enc}(\alpha) \) and \( f(x) \). This can be achieved if the underlying encryption scheme \( \text{Enc} \) is an FHE which we want to avoid. On the other hand, a proof \( \pi \) that the computation of \( \rho \) has been carried out correctly should be given to the client. To the best of our knowledge, for generating such a proof \( \pi \), one may adopt the non-interactive proofs or arguments of [22, 4] but those constructions require the use of either random oracles or KoE type assumptions which we want to avoid as well. Our idea is to adopt the multilinear maps [14, 15] which allow the server to homomorphically compute on \( \text{Enc}(\alpha) \) and \( f(x) \) and then generate \( \rho = \text{Enc}(f(\alpha)) \).

In \( \Pi_{pe} \), the client picks a \((2k + 1)\)-multilinear map instance \( \Gamma \) as in (1). It stores \( t = g_1^{f(s)} \) and gives \( \xi = (g_1, g_1^s, g_1^{s^2}, \ldots, g_1^{s^{2k-1}}) \) and \( f(x) \) to the server, where \( k = \log[n + 1] \). It also sets up \( \text{BGN}_{2k+1} \).

In order to verifiably compute \( f(\alpha) \), the client gives \( k \) ciphertexts \( \sigma = (\sigma_1, \ldots, \sigma_k) \) to the server and the server returns \( \rho = \text{Enc}(f(\alpha)) \) along with a proof \( \pi = \text{Enc}(c(s)) \), where \( \sigma_\ell = \text{Enc}(\alpha^{2^{\ell-1}}) \) for every \( \ell \in [k] \). Note that \( f(\alpha) \) and \( c(s) = (f(s) - f(\alpha))/(s - \alpha) \) are both polynomials in \( \alpha \) and \( s \).

By \( \Pi_{pe} \), we show how the server can compute \( \rho \) and \( \pi \) from \( f(x), \sigma \) and \( \xi \). Upon receiving \((\rho, \pi)\), the client decrypts \( \rho \) to \( y \) and verifies if \( e(t/g_1^{s}, g_1^{s+1}) = g_1^{s}/g_1^{s+1}g_1^{\pi} \). Finally, we can show the security and privacy of \( \Pi_{pe} \) under the assumptions \((2k+1, n)\)-MSDHS (see Definition 2.2) and SDA (see Definition 2.1).

Matrix multiplication. In Section 3.2 we propose a VC scheme \( \Pi_{mm} \) with input privacy (see Fig. 3) that allows the client to outsource the computation of \( Mx \) for any \( n \times n \) matrix \( M = (M_{ij}) \) and vector \( x = (x_1, \ldots, x_n) \). It is based on the algebraic PRFs with closed form efficiency (firstly defined by [3]). In Section 2.3, we present an algebraic PRF with closed form efficiency \( \text{PRF}_{\text{dlin}} = (K, F) \) over a trilinear map instance \( \Gamma \), where for any secret key \( K \) generated by \( \text{KG} \), \( F_K \) is a function with domain \( [n]^2 \) and range \( G_1 \). In \( \Pi_{mm} \), the client gives both \( M \) and its blinded version \( T = (T_{ij}) \) to the server, where \( T_{ij} = g_3^{x_{ij}} \cdot F_K(i, j) \) for every \( (i, j) \in [n]^2 \) and \( a \) is randomly chosen from \( \mathbb{Z}_N \) and fixed for any \((i, j) \in [n]^2 \). It also sets up \( \text{BGN}_3 \). In order to verifiably compute \( Mx \), the client stores \( \tau_i = \prod_{j=1}^{n} e(F_K(i, j), g_2^{\beta_{ij}}) \) for every \( i \in [n] \), where \( \tau_i \) can be efficiently computed using the closed form efficiency property of \( \text{PRF}_{\text{dlin}} \). It gives the ciphertexts \( \sigma = (\text{Enc}(x_1), \ldots, \text{Enc}(x_n)) \) to the server and the server returns \( \rho = \text{Enc}(\sum_{j=1}^{n} M_{ij}x_j) \) along with a proof \( \pi = \prod_{j=1}^{n} e(T_{ij}, \text{Enc}(x_j)) \) for every \( i \in [n] \). Upon receiving \( \rho = (\rho_1, \ldots, \rho_n) \) and \( \pi = (\pi_1, \ldots, \pi_n) \), the client can decrypt \( \rho_i \) to \( y_i \) and verify if \( e(\pi_i, g_1^\rho_i) = \eta^{\rho_i} \cdot \tau_i \) for every \( i \in [n] \), where \( \eta = g_3^{x_{ii}} \). Finally, we can show the security and privacy of \( \Pi_{mm} \) under the assumptions 3-co-CDHS (see Definition 2.5), DLIN (see Definition 2.5) and SDA.

Applications. Our VC schemes have applications in outsourcing private information retrieval where the client stores a large database \( w = w_1 \cdots w_n \in \{0, 1\}^n \) with the cloud and later retrieves a bit without revealing which bit he is interested in. Outsourcing PIR has practical significance: for example a health professional that stores a database of medical records with the cloud may want to privately retrieve the record of a certain patient. Our VC schemes provide easy solutions for outsourcing PIR. A client with database \( w \) can outsource a polynomial \( f(x) \) to the cloud using \( \Pi_{pe} \), where \( f(i) = w_i \) for every \( i \in [n] \).

The client can also represent its database as a \( \sqrt{n} \times \sqrt{n} \) matrix \( M = (M_{ij}) \) and outsource it to the cloud using \( \Pi_{mm} \). Retrieving any bit \( M_{ij} \) can be reduced to computing \( Mx \) for a 0-1 vector \( x \in \{0, 1\}^{\sqrt{n}} \) whose \( j \)-th bit is 1 and all other bits are 0. Our indistinguishability based definition of input privacy (see Fig. 1) guarantees that the server cannot learn which bit the client is interested in.

Discussions. Note that decrypting \( \rho = \text{Enc}(f(\alpha)) \) in \( \Pi_{pe} \) requires computing discrete logarithms (see Section 2.2). Hence, the \( f(\alpha) \) should be from a polynomial-size domain \( M \) since otherwise the client will not be able to decrypt \( \rho \) and then verify its correctness. In fact, this is an inherent limitation of [6] and inherited by the generalized BGN encryption schemes. However, in Section 3.3 we shall see that the limitation is only theoretical and does not affect the application of \( \Pi_{pe} \) in outsourcing PIR, where \( f(\alpha) \in \{0, 1\} \). One may also argue that with \( f(x) \) and the knowledge of \( \{f(\alpha) \in M\} \),
the server may learn a polynomial size domain \( \mathbb{D} \) where \( \alpha \) is drawn from and therefore guess \( \alpha \) with non-negligible probability. However, due to Definition 2.9, we stress that the input privacy achieved by \( \Pi_{pe} \) is indistinguishability based which does not contradict to the above argument. In fact, such a privacy level suffices for our applications in outsourcing PIR. In Section 3.4, we show how to modify \( \Pi_{pe} \) such that \( f(x) \) is also hidden and therefore prevent the cloud from learning any information about \( \alpha \). Similar discussions as above are applicable to \( \Pi_{mm} \).

**Extensions.** In Section 3.4, we modify \( \Pi_{pe} \) and \( \Pi_{mm} \) such that the function privacy is also achieved. In the modified schemes \( \Pi'_{pe} \) (see Fig. 4) and \( \Pi'_{mm} \) (see Fig. 5), the outsourced functions are encrypted and then given to the server. The basic idea is increasing the multi-linearity by 1 such that both the server and the client can compute on encrypted inputs and functions with one more application of the multilinear map \( e \). The modified schemes \( \Pi'_{pe} \) and \( \Pi'_{mm} \) achieve both input and function privacy.

1.2 Related Work

Verifiable computation can be traced back to the work on interactive proofs or arguments [19, 22]. In the context of VC, the non-interactive proofs or arguments are much more desirable and have been considered in [22, 4] for various computations. However, they use either random oracles or KoE type assumptions.

Gennaro, Gentry and Parno [16] constructed the first non-interactive VC schemes without using random oracles or KoE type assumptions. Their construction is based on the FHE and garbled circuits. Also based on FHE, Chung et al. [11] proposed a VC scheme that requires no public key. Applebaum et al. [1] reduced VC to suitable variants of secure multiparty computation protocols. Barbosa et al. [2] also obtained VC schemes using delegatable homomorphic encryption. Although the input privacy has been explicitly considered in [16, 2], those schemes evaluate the outsourced functions as circuits and are not very efficient. Furthermore, in these schemes the outsourced functions are known to the server. Hence, they do not achieve function privacy.

Benabbas et al. [3] initiated a line of research on efficient VC schemes for specific function (polynomial) evaluations based on algebraic PRFs with closed form efficiency. In particular, one of their VC schemes achieves function privacy. Parno et al. [25] initiated a line of research on public VC schemes for evaluating Boolean formulas, where the correctness of the server’s computation can be verified by any client. Also based on algebraic PRFs with closed form efficiency, Fiore et al. [13, 12] constructed public VC schemes for both polynomial evaluation and matrix multiplication. Using the idea of polynomial commitments [20], Papamanthou et al. [23] constructed public VC schemes that enable efficient updates. A common drawback of [3, 25, 13, 12, 23] is that the input privacy is not achieved. VC schemes with other specific properties have also been constructed in [18, 22, 10, 4, 8, 9]. However, none of them is privacy preserving.

**Organization.** In Section 2, we firstly review several cryptographic assumptions related to multilinear maps; then introduce a generalization of the BGN encryption scheme [6]; we also recall algebraic PRFs with closed form efficiency and the formal definition of VC. In Section 3, we present our VC schemes for univariate polynomial evaluation and matrix multiplication. In Section 4, we show applications of our VC schemes in outsourcing PIR. Section 5 contains some concluding remarks.

2 Preliminaries

For any finite set \( A \), the notation \( \omega \leftarrow A \) means that \( \omega \) is uniformly chosen from \( A \). Let \( \lambda \) be a security parameter. We denote by \( \text{neg}(\lambda) \) the class of negligible functions in \( \lambda \), i.e., for every constant \( c > 0 \), it is less than \( \lambda^{-c} \) as long as \( \lambda \) is large enough. We denote by \( \text{poly}(\lambda) \) the class of polynomial functions in \( \lambda \).
2.1 Multilinear Maps and Assumptions

In this section, we review several cryptographic assumptions concerning multilinear maps. Given the $\Gamma_k$ in (1) and $x \in G_i$, the subgroup decision problem in $G_i$ is deciding whether $x$ is of order $p$ or not, where $i \in [k]$. When $k = 2$, Boneh et al. [6] suggested the Subgroup Decision Assumption (SDA) which says that the subgroup decision problems in $G_1$ and $G_2$ are intractable. In this paper, we make the same assumption but for a general integer $k \geq 2$.

Definition 2.1 (SDA) We say that SDA$_i$ holds if for any probabilistic polynomial time (PPT) algorithm $A$, $|\Pr[A(\Gamma_k, u) = 1] - \Pr[A(\Gamma_k, u') = 1]| < \text{neg}(\lambda)$, where the probabilities are taken over $\Gamma_k \leftarrow G(1^\lambda, k), u \leftarrow G_i$ and $A$’s random coins. We say that SDA holds if SDA$_i$ holds for every $i \in [k]$.

The following lemma shows that SDA is equivalent to SDA$_1$ (see Appendix A for the proof).

Lemma 2.1 If SDA$_i$ holds, then SDA$_j$ holds for every $j = i + 1, \ldots, k$.

The $k$-Multilinear $n$-Strong Diffie-Hellman assumption ($(k, n)$-MSDH) was suggested in [24]: Given $g_1^s, g_1^{s^2}, \ldots, g_1^{s^n}$ for some $s \leftarrow \mathbb{Z}_N$, it is difficult for any PPT algorithm to find $\alpha \in \mathbb{Z}_N \setminus \{-s\}$ and output $g_k^{1/(s + \alpha)}$.

Definition 2.2 ($(k, n)$-MSDH) For any PPT algorithm $A$, $\Pr[A(p, q, \Gamma_k, g_1, g_1^s, \ldots, g_1^{s^n}) = (\alpha, g_k^{1/(s + \alpha)})] < \text{neg}(\lambda)$, where $\alpha \in \mathbb{Z}_N \setminus \{-s\}$ and the probability is taken over $\Gamma_k \leftarrow G(1^\lambda, k), s \leftarrow \mathbb{Z}_N$ and $A$’s random coins.

We are able to construct a privacy preserving VC scheme (see Section 3.1) for univariate polynomial evaluation which is secure based on $(k, n)$-MSDH. Under the $(k, n)$-MSDH assumption, the following lemma (see Appendix B for the proof) shows that either one of the following two problems is difficult for any PPT algorithm: (i) given $g_1, g_1^s, \ldots, g_1^{s^n}$ for some $s \leftarrow \mathbb{Z}_N$, compute $g_k^{p/s}$; (ii) given $g_1, g_1^s, \ldots, g_1^{s^n}$ for some $s \leftarrow \mathbb{Z}_N$, compute $g_k^{q/s}$.

Lemma 2.2 If $(k, n)$-MSDH holds, then except for a negligible fraction of the $k$-multilinear map instances $\Gamma_k \leftarrow G(1^\lambda, k)$, either $\Pr[A(p, q, \Gamma_k, g_1, g_1^s, \ldots, g_1^{s^n}) = g_k^{p/s}] < \text{neg}(\lambda)$ for any PPT algorithm $A$ or $\Pr[A(p, q, \Gamma_k, g_1, g_1^s, \ldots, g_1^{s^n}) = g_k^{q/s}] < \text{neg}(\lambda)$ for any PPT algorithm $A$, where the probabilities are taken over $s \leftarrow \mathbb{Z}_N$ and $A$’s random coins.

Due to Lemma 2.2, it looks reasonable to assume that (i) (resp. (ii)) is difficult. Furthermore, under this slightly stronger assumption (i.e., (i) is difficult, called $(k, n$)-MSDHS from now on), we can construct a VC scheme $\pi_{pe}$ (see Fig. 2) that is more efficient than the one based on $(k, n)$-MSDH. In Section 3.1, we present the scheme $\pi_{pe}$ based on $(k, n$)-MSDHS.

Definition 2.3 ($(k, n)$-MSDHS) For any PPT algorithm $A$, $\Pr[A(p, q, \Gamma_k, g_1, g_1^s, \ldots, g_1^{s^n}) = g_k^{p/s}] < \text{neg}(\lambda)$, where the probability is taken over $\Gamma_k \leftarrow G(1^\lambda, k), s \leftarrow \mathbb{Z}_N$ and $A$’s random coins.

The $k$-Multilinear Decision Diffie-Hellman assumption (k-MDDH) was suggested in [14, 15]: Given $g_1^s, g_1^{a_1}, \ldots, g_1^{a_k} \leftarrow G_1$, it is difficult for any PPT algorithm to distinguish between $g_k^{sa_1\cdots a_k}$ and $h \leftarrow G_k$.

Definition 2.4 (k-MDDH) For any PPT algorithm $A$, $|\Pr[A(p, q, \Gamma_k, g_1^s, g_1^{a_1}, \ldots, g_1^{a_k}, g_k^{sa_1\cdots a_k}) = 1] - \Pr[A(p, q, \Gamma_k, g_1^s, g_1^{a_1}, \ldots, g_1^{a_k}, h) = 1]| < \text{neg}(\lambda)$, where the probabilities are taken over $\Gamma_k \leftarrow G(1^\lambda, k), s, a_1, \ldots, a_k \leftarrow \mathbb{Z}_N, h \leftarrow G_k$ and $A$’s random coins.

Let $\Gamma_3 = (N, G_1, G_2, G_3, \epsilon, g_1, g_2, g_3) \leftarrow G(1^\lambda, 3)$ be a random trilinear map instance. Let $h_1 = g_1^s$ and $h_2 = g_2^t$. The trilinear co-Computational Diffie-Hellman assumption for the order $q$ Subgroups (3-co-CDHS) says that given $h_i'^t \leftarrow G_1$ and $h_2'^t \leftarrow G_2$, it is difficult for any PPT algorithm to compute $h_2'^t$.
Definition 2.5 (3-co-CDHS) For any PPT algorithm $A$, $\Pr[A(p,q,\Gamma_3, h_a^b, h_b^a) = h_2^b] < \text{neg}(\lambda)$, where the probability is taken over $\Gamma_3 \leftarrow \mathcal{G}(1^\lambda, 3)$, $a, b \leftarrow \mathbb{Z}_N$ and $A$’s random coins.

Based on the following technical Lemma 2.3, Lemma 2.4 shows that 3-co-CDHS is not a new assumption but weaker than 3-MDDH (see Appendix C and D for their proofs).

Lemma 2.3 Let $X$ and $Y$ be two uniform random variables over $\mathbb{Z}_N$. Then the random variable $Z = pXY \mod q$ is statistically close to the uniform random variable $U$ over $\mathbb{Z}_q$, i.e., we have that $\sum_{\omega \in \mathbb{Z}_q} |\Pr[Z = \omega] - \Pr[U = \omega]| < \text{neg}(\lambda)$.

Lemma 2.4 If 3-MDDH holds, then 3-co-CDHS holds.

The Decision LINear assumption (DLIN) has been suggested in [5] for cyclic groups that admit bilinear maps. In this paper, we use the DLIN assumption on the groups of $\Gamma_3$.

Definition 2.6 (DLIN) Let $G$ be a cyclic group of order $N = pq$, where $p, q$ are $\lambda$-bit primes. For any PPT algorithm $A$, $|\Pr[A(p,q,u,v,w,u^a,v^b,w^{a+b}) = 1] - \Pr[A(p,q,u,v,w,u^a,v^b,w^{a+b}) = 1]| < \text{neg}(\lambda)$, where the probabilities are taken over $u, v, w \leftarrow G$, $a, b, c \leftarrow \mathbb{Z}_N$ and $A$’s random coins.

2.2 Generalized BGN Encryption

BGN$_2$ [6] allows one to evaluate quadratic polynomials on encrypted inputs (see Section 1.1). Boneh et al. [6] noted that this property arises from the bilinear map and a $k$-multilinear map would enable the evaluation of degree-$k$ polynomials on encrypted inputs. Let $\mathbb{M}$ be a polynomial size domain, i.e. $|\mathbb{M}| = \text{poly}(\lambda)$. Below we generalize BGN$_2$ and define BGN$_k = (\text{Gen}, \text{Enc}, \text{Dec})$ for any $k \geq 2$, where

- $(pk, sk) \leftarrow \text{Gen}(1^\lambda, k)$ is a key generation algorithm. It picks $\Gamma_k$ as in (1) and then outputs both a public key $pk = (\Gamma_k, g_1, h)$ and a secret key $sk = p$, where $h = u^d$ for $u \leftarrow G_1$.
- $c \leftarrow \text{Enc}(pk, m)$ is an encryption algorithm which encrypts any message $m \in \mathbb{M}$ as a ciphertext $c = g_1^m h^r \in G_1$, where $r \leftarrow \mathbb{Z}_N$.
- $m \leftarrow \text{Dec}(sk, c)$ is a decryption algorithm which takes as input $sk$ and a ciphertext $c$, and outputs a message $m \in \mathbb{M}$ such that $c^p = (g_1^p)^m$.

Note that all algorithms above are defined over $G_1$ but in general they can be defined over $G_i$ for any $i \in [k]$. This can be done by setting $pk = (\Gamma_k, g_1, h)$ and replacing any occurrence of $g_1$ with $g_i$, where $h = u^d$ for $u \leftarrow G_i$. Similar to [6], one can show that BGN$_k$ is semantically secure under the SDA.

Below we discuss useful properties of BGN$_k$. For every integer $2 \leq i \leq k$, we define a map $e_i : G_1 \times \cdots \times G_1 \rightarrow G_i$ such that $e_i(g_1^{a_1}, \ldots, g_1^{a_n}) = g_i^{a_i}$ for any $a_1, \ldots, a_n \in \mathbb{Z}_N$. Firstly, we shall see that BGN$_k$ allows us to compute $\text{Enc}(m_1 \cdots m_k)$ from $\text{Enc}(m_1), \ldots, \text{Enc}(m_k)$. Suppose $\text{Enc}(m_\ell) = g_1^{m_\ell} h_\ell^r$ for every $\ell \in [k]$, where $h = g_1^d$ for some $d \in \mathbb{Z}_N$ and $r_\ell \leftarrow \mathbb{Z}_N$. Let $h_k = e_k(h, g_1, \ldots, g_1) = g_1^d$. Then $e_k(\text{Enc}(m_1), \ldots, \text{Enc}(m_k)) = g_1^{m_k} h_k^r$ is a ciphertext of $m_1 \cdots m_k$ in $G_k$, where $r = \frac{1}{q_0}(\prod_{\ell=1}^k (m_\ell + q_0 r_\ell) - m)$.

Computing $\rho$ with reduced multi-linearity level. In $\Pi_{pe}$, the client gives a polynomial $f(x) = f_0 + f_1 x + \cdots + f_n x^n$ and $k$ ciphertexts $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k)$ of $\alpha, \alpha^2, \ldots, \alpha^{2^{k-1}}$ under BGN$_{2k+1}$ to the server and the server returns $\rho = \text{Enc}(f(\alpha))$, where $k = \lceil \log(n+1) \rceil$. Below we show how to compute the $\rho$ using $\sigma$ and $f(x)$. Suppose $\sigma_i = g_1^{2^{i-1}} h_\ell^r$ for every $\ell \in [k]$, where $h = g_1^d$ for some $d \in \mathbb{Z}_N$ and $r_\ell \leftarrow \mathbb{Z}_N$. Clearly, any $i \in \{0, 1, \ldots, n\}$ has a binary representation $(i_0, \ldots, i_k)$ such that $i = \sum_{\ell=1}^k i_\ell 2^{\ell-1}$. Then $\alpha^i = \alpha^{i_0} \cdot (\alpha^2)^{i_1} \cdots (\alpha^{2^{k-1}})^{i_k}$ is the product of $i_0 + \cdots + i_k$ elements of $\{\alpha, \alpha^2, \ldots, \alpha^{2^{k-1}}\}$. For every $\ell \in [k]$, let $\phi_\ell = \sigma_\ell$ if $i_\ell = 1$ and $\phi_\ell = g_1$ otherwise. Then $\rho_i = e_k(\phi_1, \ldots, \phi_k) = g_k^m h_k^r$ is a ciphertext of $m = \alpha^i$ under BGN$_{2k+1}$, where $m_i = \prod_{\ell=1}^k (\alpha^{2^{\ell-1}} + q_0 r_\ell)^{i_\ell}$ and $r = \frac{1}{q_0}(m_i - m)$. Thus, $\rho = \prod_{i=0}^n \rho_i$ is a ciphertext of $f(\alpha)$ under BGN$_{2k+1}$.
Computing $\pi$ with reduced multi-linearity level. In $\Pi_{pe}$, $k+1$ group elements $\xi = (g_1, g_1^s, \ldots, g_1^{s^{k-1}})$ are also known to the server as part of the public key, where $s \leftarrow \mathbb{Z}_N$. The server must return $\pi = \text{Enc}(c(s))$ as the proof that $\rho = \text{Enc}(f(\alpha))$ has been correctly computed. Below we show how to compute $\pi$ using $\xi$ and $\sigma$. Note that $c(s) = (f(s) - f(\alpha))/(s - \alpha) = \sum_{j=0}^{n-1} f_i + 1 \alpha^j s^{-j}$. It suffices to show how to compute $\pi_{ij} = \text{Enc}(f_i + 1 \alpha^j s^{-j})$ for every $i \in \{0, 1, \ldots, n - 1\}$ and $j \in \{0, 1, \ldots, i\}$. Let $(j_1, \ldots, j_k)$ be the binary representations of $j$ and $i - j$, respectively. Let $\phi_i$ be the $\sigma$ if $j = 1$ and $\phi_i = g_1$ otherwise. Let $\psi_i = g_1^{\phi_i}$ and $\psi_{i+1} = g_1^{\phi_{i+1}}$. Then it is easy to see that $\pi_{ij} = \psi_i \psi_{i+1} \cdots \psi_k = g_{2k}^{\mu_k^i}$ is a ciphertext of $m = \alpha^j s^{-j}$, where $\mu_k^i = s^{-j} \prod_{l=1}^{k} (\alpha^{2i-l} + q^2 r_i)^{\nu_i^j}$, $h_{2k} = g_{1k}^\nu$, and $r = \frac{1}{q^\nu} (\nu_{ij} - m)$. Thus, $\pi \triangleq g_{2k}^{\mu_k^i} = \prod_{i=0}^{n-1} \prod_{j=0}^{i} \pi_{ij} = \text{Enc}(c(s))$.

2.3 Algebraic PRFs with Closed Form Efficiency

In $\Pi_{mm}$, the client gives both a square matrix $M = (M_{ij})$ of order $n$ and its blinded version $T = (T_{ij})$ to the server. The computation of $T$ requires an algebraic PRF with closed form efficiency, which has very efficient algorithms for certain computations on large data. Formally, an algebraic PRF with closed form efficiency is a pair $\text{PRF} = (\text{KG}, \text{F})$, where $\text{KG}(\lambda, pp)$ generates a secret key $K$ from any public parameter $pp$ and $F_K : I \rightarrow G$ is a function with domain $I$ and range $G$ (both specified by $pp$). We say that $\text{PRF}$ has pseudorandom property if for any $pp$ and any PPT algorithm $A$, it holds that $|\text{Pr}[A^{F_K(\cdot)}(1^\lambda, pp) = 1] - \text{Pr}[A^{\text{PRF}(1^\lambda, pp)} = 1]| < \text{neg}(\lambda)$, where the probabilities are taken over the randomness of $\text{KG}$, $A$ and the random function $R : I \rightarrow G$. Consider an arbitrary computation $\text{Comp}$ that takes as input $R = (R_1, \ldots, R_n) \in G^n$ and $x = (x_1, \ldots, x_n)$, and assume that the best algorithm to compute $\text{Comp}(R_1, \ldots, R_n, x_1, \ldots, x_n)$ takes time $t$. Let $z = (z_1, \ldots, z_n) \in I^n$. We say that $\text{PRF}$ has closed form efficiency for $(\text{Comp}, z)$ if there is an efficient algorithm $\text{CFE}_{\text{Comp}, z}$ such that $\text{CFE}_{\text{Comp}, z}(K, z) = \text{Comp}(F_K(z_1), \ldots, F_K(z_n), x_1, \ldots, x_n)$ and its running time is $o(t)$.

A PRF with closed form efficiency. Fiore et al. [13] constructed an algebraic PRF with closed form efficiency $\text{PRF}_{\text{dlin}}$ based on the DLIN assumption for the bilinear groups. We generalize it over trilinear groups. In the generalized setting, $\text{KG}$ generates $\Gamma_3 \leftarrow G(1^{\lambda}, 3)$, picks $\alpha_i, \beta_i \leftarrow \mathbb{Z}_N$, $A_i, B_i \leftarrow G_1$ for every $i \in [n]$, and outputs $K = \{\alpha_i, \beta_i, A_i, B_i : i \in [n]\}$. The function $F_K$ maps any pair $(i, j) \in [n]^2$ to $F_K(i, j) = A_i^{\alpha_i} B_j^{\beta_j}$. The closed form efficiency of $\text{PRF}_{\text{dlin}}$ is described as below. Let $x = (x_1, \ldots, x_n) \in \mathbb{Z}_N^n$. The computation $\text{Comp}$ we consider is computing $\prod_{i=1}^{n} F_K(i, j)^{x^j}$ for all $i \in [n]$. Clearly, it requires $O(n^2)$ exponentiations if no CFE is available. However, one can precompute $A = A_1^{x_1} \cdots A_n^{x_n}$ and $B = B_1^{x_1} \cdots B_n^{x_n}$ and have that $\prod_{i=1}^{n} F_K(i, j)^{x^j} = A^{\alpha_i} B^{\beta_j}$ for every $i \in [n]$. Computing $A^{\alpha_i} B^{\beta_j}$ requires 2 exponentiations and hence the $\text{PRF}_{\text{dlin}}$ has closed form efficiency for $(\text{Comp}, z)$, where $z = \{i, j : i, j \in [n]\}$. The $\text{PRF}_{\text{dlin}}$ in [13] is pseudorandom merely based on the DLIN assumption for bilinear groups. Similarly, the generalized $\text{PRF}_{\text{dlin}}$ is also pseudorandom based on the DLIN assumption for trilinear groups. Consequently, we have the following lemma.

Lemma 2.5 If DLIN holds in the trilinear setting, then $\text{PRF}_{\text{dlin}}$ is an algebraic PRF with closed form efficiency.

2.4 Verifiable Computation

Verifiable computation [16, 3, 13] is a two-party protocol between a client and a server, where the client gives encodings of its function $f$ and input $x$ to the server, the server returns an encoding of $f(x)$ along with a proof, and finally the client efficiently verifies the server’s computation. Formally, a VC scheme $\Pi = (\text{KeyGen}, \text{ProbGen}, \text{Compute}, \text{Verify})$ is defined by four algorithms, where

- $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda, f)$ takes as input a security parameter $\lambda$ and a function $f$, and generates both a public key $pk$ and a secret key $sk$.

\[ (pk, sk) \leftarrow \text{KeyGen}(1^\lambda, f) \]
(σ, τ) ← ProbGen(sk, x) takes as input the secret key sk and an input x, and generates both an encoded input σ and a verification key τ;

(ρ, π) ← Compute(pk, σ) takes as input the public key pk and an encoded input σ, and produces both an encoded output ρ and a proof π;

\{f(x), ⊥\} ← Verify(sk, τ, ρ, π) takes as input the secret key sk, the verification key τ, the encoded output ρ and a proof π, and outputs either f(x) or ⊥ (which indicates that ρ is not valid).

**Correctness.** The scheme Π should be correct. Intuitively, the scheme Π is correct if an honest server always outputs a pair (ρ, π) that gives the correct computation result. Let F be a family of functions.

**Definition 2.7** The scheme Π is said to be \( F \)-correct if for any \( f \in F \), any \((pk, sk) \leftarrow KeyGen(1^λ, f)\), any input \( x \) to \( f \), any \((σ, τ) \leftarrow ProbGen(sk, x)\), any \((ρ, π) \leftarrow Compute(pk, σ)\), it holds that \( f(x) = Verify(sk, τ, ρ, π) \).

**Experiment** \( \text{Exp}_{\text{Ver}}^V(Π, f, λ) \)

1. \((pk, sk) \leftarrow KeyGen(1^λ, f)\);
2. for \( i = 1 \) to \( l = poly(λ) \) do
3. \( x_i \leftarrow A(pk, x_1, σ_1, \ldots, x_{i-1}, σ_{i-1})\);
4. \((σ_i, τ_i) \leftarrow ProbGen(sk, x_i)\);
5. \( ˆx \leftarrow A(pk, x_1, σ_1, \ldots, x_i, σ_i)\);
6. \((ˆσ, ˆτ) \leftarrow ProbGen(sk, ˆx)\);
7. \((ˆρ, ˆπ) \leftarrow A(pk, x_1, σ_1, \ldots, x_i, σ_i, ˆσ)\);
8. \( ˆy \leftarrow Verify(sk, ˆτ, ˆρ, ˆπ)\);
9. output 1 if \( ˆy \notin \{f(ˆx), ⊥\} \) and 0 otherwise.

**Experiment** \( \text{Exp}_{\text{Pri}}^V(Π, f, λ) \)

1. \((pk, sk) \leftarrow KeyGen(1^λ, f)\);
2. \((x_0, x_1) \leftarrow A^{\text{PubProbGen}(sk, ·)}(pk)\);
3. \( b \leftarrow \{0, 1\} \);
4. \((σ, τ) \leftarrow ProbGen(sk, x_0)\);
5. \( b' \leftarrow A^{\text{PubProbGen}(sk, ·)}(pk, x_0, x_1, σ)\);
6. output 1 if \( b' = b \) and 0 otherwise.

Remark: \( \text{PubProbGen}(sk, ·) \) takes as input x, runs \((σ, τ) \leftarrow ProbGen(sk, x)\) and returns σ.

**Security.** The scheme Π should be secure. As in [16], we say that the scheme Π is secure if no untrusted server can cause the client to accept an incorrect computation result with a forged proof. This intuition can be formalized by an experiment \( \text{Exp}_{\text{Ver}}^V(Π, f, λ) \) (see Fig. 1) where the challenger plays the role of the client and the adversary A plays the role of the untrusted server.

**Definition 2.8** The scheme Π is said to be \( F \)-secure if for any \( f \in F \) and any PPT adversary A, it holds that \( \text{Pr}[\text{Exp}_{\text{Ver}}^V(Π, f, λ) = 1] < \text{neg}(λ) \).

**Privacy.** The client’s input should be hidden from the server in Π. As in [16], we define input privacy based on the intuition that no untrusted server can distinguish between different inputs of the client. This is formalized by an experiment \( \text{Exp}_{\text{Pri}}^V(Π, f, λ) \) (see Fig. 1) where the challenger plays the role of the client and the adversary A plays the role of the untrusted server.

**Definition 2.9** The scheme Π is said to achieve input privacy if for any function \( f \in F \), any PPT algorithm A, it holds that \( \text{Pr}[\text{Exp}_{\text{Pri}}^V(Π, f, λ) = 1] < \text{neg}(λ) \).

**Efficiency.** The algorithms ProbGen and Verify will be run by the client for each evaluation of the outsourced function \( f \). Their running time should be substantially less than evaluating \( f \).

**Definition 2.10** The scheme Π is said to be outsourced if for any \( f \in F \) and any input \( x \) to \( f \), the running time of ProbGen and Verify is \( o(t) \), where \( t \) is the time required to compute \( f(x) \).
3 Our Schemes

3.1 Univariate Polynomial Evaluation

In this section, we present our VC scheme $\Pi_{pe}$ with input privacy (see Fig. 2) for univariate polynomial evaluation. In $\Pi_{pe}$, the client outsources a degree $n$ polynomial $f(x) = f_0 + f_1 x + \cdots + f_n x^n \in \mathbb{Z}_q[x]$ to the server and may evaluate $f(\alpha)$ for any input $\alpha \in \mathbb{D} \subseteq \mathbb{Z}_q$, where $q$ is a $\lambda$-bit prime not known to the server and $|\mathbb{D}| = \text{poly}(\lambda)$. Our scheme uses a $(2k + 1)$-multilinear map instance $\Gamma$ with groups of order $N = pq$, where $k = \lceil \log(n + 1) \rceil$ and $p$ is also a $\lambda$-bit prime not known to the server. The client stores $t = g_{f(s)}^1$ and gives $(g_1^1, g_1^2, \ldots, g_1^{2^{k-1}}, f)$ to the server, where $s \leftarrow \mathbb{Z}_N$. It also sets up BGN$_{2k+1}$ based on $\Gamma$. In order to verifiably compute $f(\alpha)$, the client gives $\sigma = (\sigma_1, \ldots, \sigma_k)$ to the server and the server returns $\rho = \text{Enc}(f(\alpha))$ along with $\pi = \text{Enc}(c(s))$, where $\sigma_\ell = \text{Enc}(\alpha^{2^{\ell-1}})$ for every $\ell \in [k]$ and $(\rho, \pi)$ is computed using the techniques in Section 2.2. At last, the client decrypts $\rho$ to $y$ and verifies if the equation (2) holds.

- **KeyGen**($1^\lambda, f(x)$): Pick $\Gamma = (N, G_1, \ldots, G_{2k+1}, e, g_1, \ldots, g_{2k+1}) \leftarrow G(1^\lambda, 2k + 1)$. Pick $s \leftarrow \mathbb{Z}_N$ and compute $t = g_{f(s)}^1$. Pick $u \leftarrow G_1$ and compute $h = u^q$, where $u = g_1^\delta$ for an integer $\delta \in \mathbb{Z}_N$. Set up BGN$_{2k+1}$ with public key $(\Gamma, g_1, h)$ and secret key $p$. Output $sk = (p, q, s, t)$ and $pk = (\Gamma, g_1, h; g_1^1, g_1^2, \ldots, g_1^{2^{k-1}}; f)$.

- **ProbGen**($sk, \alpha$): For every $\ell \in [k]$, pick $r_\ell \leftarrow \mathbb{Z}_N$ and compute $\sigma_\ell = g_1^\alpha h^{r_\ell}$. Output $\sigma = (\sigma_1, \ldots, \sigma_k)$ and $\tau = \bot$ ($\tau$ is not used).

- **Compute**($pk, \sigma$): Compute $\rho_i = g_{f(i)}^\mu$ for every $i \in \{0, 1, \ldots, n\}$ using the technique in Section 2.2. Compute $\rho = \prod_{i=0}^{n-1} \rho_i^1$. Compute $\pi_{ij} = g_{g_{2k}^j}^\nu_{ij}$ for every $i \in \{0, 1, \ldots, n-1\}$ and $j \in \{0, 1, \ldots, i\}$ using the technique in Section 2.2. Compute $\pi = \prod_{i=0}^{n-1} \prod_{j=0}^{i} \pi_{ij}^{f(i+1)}$. Output $\rho$ and $\pi$.

- **Verify**($sk, \tau, \rho, \pi$): Compute the $y \in \mathbb{Z}_q$ such that $\rho^q = (g_1^p)^y$. If  
  \[ e(t/g_1^y, g_{2k}^p) = e(g_1^y/g_1^\alpha, \pi^p), \tag{2} \]
  then output $y$; otherwise, output $\bot$.

![Figure 2: Univariate polynomial evaluation (\Pi_{pe})](image)

**Correctness.** The correctness of $\Pi_{pe}$ requires that the client always outputs $f(\alpha)$ as long as the server is honest, i.e., $y = f(\alpha)$ and (2) holds. It is shown by the following lemma.

**Lemma 3.1** If the server is honest, then $y = f(\alpha)$ and (2) holds.

**Proof:** Firstly, $p\mu_i \equiv p\alpha^j \mod N$ for every $i \in \{0, 1, \ldots, n\}$. Since $g_k$ is of order $N$, we have that  
\[ \rho^p = \prod_{i=0}^{n} \rho_i^p = \prod_{i=0}^{n} g_{k_i}^{p\mu_i} = \prod_{i=0}^{n} g_{k_i}^{p\alpha^j} = (g_{k_i}^{p})^{f(\alpha)}, \]
which implies that $y = f(\alpha)$. Secondly, $p\nu_{ij} \equiv p\alpha^j s^{i-j} \mod N$ for every $i \in \{0, 1, \ldots, n-1\}$ and $j \in \{0, 1, \ldots, i\}$. Thus,  
\[ \pi^p = \prod_{i=0}^{n-1} \prod_{j=0}^{i} \pi_{ij}^{p\nu_{ij}^p} = \prod_{i=0}^{n-1} \prod_{j=0}^{i} g_{g_{2k}^j}^{p\nu_{ij}^p} = \prod_{i=0}^{n-1} \prod_{j=0}^{i} g_{g_{2k}^j}^{p\alpha^j s^{i-j}} = g_{g_{2k}^j}^{c(s)}. \]
It follows that $e(g_1^y/g_1^\alpha, \pi^p) = g_{g_{2k}^j}^{(s-\alpha)c(s)} = g_{g_{2k}^{f(\alpha)}} = e(t/g_1^y, g_{2k}^p)$, i.e., the equality (2) holds.
Security. The security of $\Pi_{pe}$ requires that no untrusted server can cause the client to accept a value $\bar{y} \neq f(\alpha)$ with a forged proof. It is based on the $(2k + 1, n)$-MSDHS assumption (see Definition 2.2).

Lemma 3.2 If $(2k + 1, n)$-MSDHS holds for $\Gamma$, then the scheme $\Pi_{pe}$ is secure.

Proof: Suppose that $\Pi_{pe}$ is not secure. Then there is a PPT adversary $A$ that breaks its security with non-negligible probability $\epsilon_1$. We shall construct a PPT simulator $B$ that simulates $A$ and breaks the $(2k + 1, n)$-MSDHS for $\Gamma$. The simulator $B$ takes as input $(p, q, \Gamma, g_1, g_1^s, \ldots, g_1^n)$, where $s \leftarrow Z_N$.

The simulator $B$ is required to output $g_2^{p/s}$. In order to do so, $B$ simulates $A$ as below:

(A) Pick a polynomial $f(x) = f_0 + f_1 x + \cdots + f_n x^n \in Z_q[x]$. Pick $u \leftarrow G_1$, compute $h = u^d$ and set up $\text{BGN}_{2k+1}$ with public key $(\Gamma, g_1, h)$ and secret key $p$. Pick $b \leftarrow \mathbb{D} \cap \mathbb{D}$ and implicitly set $s = b + \beta$ ($\beta$ is not known to $B$). Mimic $\text{KeyGen}$ by sending $pk = (\Gamma, g_1, h, g_1^s, g_1^{s_{2k-1}}, f)$ to $A$ (note that $B$ can compute $g_1^{s_{2k-1}}$ for every $\ell \in [k]$ based on the knowledge of $\beta$ and $g_1, g_1^s, g_1^{s_{2k-1}}$).

Set $sk = (p, q, t)$, where $t = g_1(f(\hat{s}))$ (note that $sk$ does not include $\hat{s}$ as a component because $\hat{s}$ is neither known to $B$ nor used by $B$);

(B) Upon receiving $\alpha \in \mathbb{D}$ from $A$, mimic $\text{ProbGen}$ as below: pick $r_\ell \leftarrow Z_N$ and compute $\sigma_\ell = g_1^{s_{2k-1}} h^{r_\ell}$ for every $\ell \in [k]$; send $\sigma = (\sigma_1, \ldots, \sigma_k)$ to $A$.

It is trivial to verify that the $pk$ and $\sigma$ generated by $B$ are identically distributed to those generated by the client in an execution of $\Pi_{pe}$. We remark that (A) is the step 1 in $\text{Exp}^{\text{Ver}}_\mathbb{A}(\Pi, f, \lambda)$ (see Fig. 1) and (B) consists of steps 3 and 4 in $\text{Exp}^{\text{Ver}}_\mathbb{A}(\Pi, f, \lambda)$. Furthermore, (B) may be run $l = \text{poly}(\lambda)$ times as described by step 2 of $\text{Exp}^{\text{Ver}}_\mathbb{A}(\Pi, f, \lambda)$. After $l$ executions of (B), the adversary $A$ will provide an input $\hat{\alpha}$ on which he is willing to be challenged. If $\hat{\alpha} \neq \beta$, then the simulator $B$ aborts; otherwise, it continues. Note that both $\beta$ and $\hat{\alpha}$ are from the same polynomial size domain $\mathbb{D}$, the event that $\hat{\alpha} = \beta$ will occur with probability $\epsilon_2 \geq 1/|\mathbb{D}|$, which is non-negligible. If the simulator $B$ does not abort, it next runs $(\hat{\beta}, \hat{\pi}) \leftarrow \text{ProbGen}(sk, \hat{\alpha})$ and gives $A$ an encoded input $\hat{\sigma}$. Then the adversary $A$ may maliciously reply with $(\hat{\beta}, \hat{\pi})$ such that $\text{Verify}(sk, \hat{\alpha}, \hat{\pi}) \triangleq \bar{y} \notin \{f(\hat{\alpha}), \perp\}$. On the other hand, an honest server in $\Pi_{pe}$ will reply with $(\hat{\beta}, \hat{\pi})$. Due to Theorem 3.1, it must be the case that $\text{Verify}(sk, \hat{\alpha}, \hat{\pi}) \triangleq \bar{y} = f(\hat{\alpha})$.

Note that the event that $\bar{y} \notin \{f(\hat{\alpha}), \perp\}$ occurs with probability $\epsilon_1$. Suppose the event $\bar{y} \notin \{f(\hat{\alpha}), \perp\}$ occurs, then the equation (2) is satisfied by both $(\bar{y}, \hat{\pi})$ and $(\hat{\beta}, \hat{\pi})$, i.e.,

$$e(t/g_1^{\bar{y}}, g_2^p) = e(g_1^{\hat{\beta}}/g_1, \hat{\pi}^p)$$

The equalities in (3) imply that $e(g_1^{\bar{y} - \hat{\beta}}, g_2^p) = e(g_1^{\bar{y} - \hat{\beta}}, (\hat{\pi}/\hat{\pi}^p)^p)$. Hence,

$$g_2^{p/s} \equiv e(g_1, (\hat{\pi}/\hat{\pi}^p)^p) \hat{\pi}^{1/p}.$$  

Note that the left-hand side of (4) is $g_2^{p/s}$ due to $\beta = \hat{\alpha}$. Therefore, (4) means that the simulator $B$ can break the $(2k + 1, n)$-MSDHS assumption (Definition 2.3) with probability $\epsilon = \epsilon_1\epsilon_2$, which is non-negligible and contradicts to the $(2k + 1, n)$-MSDHS assumption. Hence, under the $(2k + 1, n)$-MSDHS assumption, $\epsilon_1$ must be negligible in $\lambda$, i.e., the scheme $\Pi_{pe}$ is secure.

Privacy. The input privacy of $\Pi_{pe}$ requires that no untrusted server can distinguish between different inputs of the client. This is formally defined by the experiment $\text{Exp}^{\text{Pri}}_\mathbb{A}(\Pi, f, \lambda)$ in Fig. 1. The client in our VC scheme encodes its input $\alpha$ using $\text{BGN}_{2k+1}$ which is semantically secure under SDA for $\Gamma$. As a result, our VC scheme achieves input privacy under SDA for $\Gamma$ (see Appendix E for a proof of the following lemma).

Lemma 3.3 If SDA holds for $\Gamma$, then the scheme $\Pi_{pe}$ achieves the input privacy.
Efficiency. In order to verifiably compute \( f(\alpha) \) with the cloud, the client computes \( k = \lceil \log(n+1) \rceil \) ciphertexts \( \sigma_1, \ldots, \sigma_k \) under \( \text{BGN}_{2k+1} \) in the execution of \( \text{ProbGen} \); it also decrypts one ciphertext \( \rho = \text{Enc}(f(\alpha)) \) under \( \text{BGN}_{2k+1} \) and then verifies the equation (2). The overall computation of the client will be \( O(\log n) = o(n) \) and therefore \( \Pi_{pe} \) is outsourced. On the other hand, the server needs to perform \( O(n^2 \log n) \) multilinear map computations and \( O(n^2) \) exponentiations in each execution of \( \text{Compute} \), which is comparable with the VC schemes based on FHE. Based on Lemmas 3.1, 3.2, 3.3 and the efficiency analysis, we have the following theorem.

**Theorem 3.1** If the \((2k+1,n)\)-MSDHS and SDA assumptions for \( \Gamma \) both hold, then \( \Pi_{pe} \) is a VC scheme with input privacy.

A variant of \( \Pi_{pe} \) based on \((2k+1,n)\)-MSDH. The VC scheme \( \Pi_{pe} \) we constructed in this section has its security based on the \((2k+1,n)\)-MSDHS which is slightly stronger than the \((2k+1,n)\)-MSDH. The \((2k+1,n)\)-MSDH implies that except for a negligible fraction of the \((2k+1)\)-multilinear map instances, at least one of the following problems is difficult for any PPT algorithm: (i) given \( g_1, g_1^p, \ldots, g_1^n \), compute \( g_1^{p/s} \); (ii) given \( g_1, g_1^p, \ldots, g_1^n \), compute \( g_1^{q/s} \). While it is not known how to determine which one of the two problems is difficult for a given \( \Gamma_{2k+1} \) instance, the construction of \( \Pi_{pe} \) simply assumes one of them is always difficult. In fact, we can also construct a VC scheme with input privacy for univariate polynomial evaluation based on the weaker assumption \((2k+1,n)\)-MSDH. A natural idea is generating multiple \((2k+1)\)-multilinear map instances, say \( \Gamma_{2k+1,1}, \ldots, \Gamma_{2k+1,\lambda} \), where \( \Gamma_{2k+1,l} \) is defined over groups of order \( N_l = pqq \) for every \( l \in [\lambda] \). The client can simply run one instance of \( \Pi_{pe} \) based on each one of the \( \lambda \) multilinear map instances \( \Gamma_{2k+1,1}, \ldots, \Gamma_{2k+1,\lambda} \). In particular, in each execution, the client simply picks a random one from the two problems and believe that the chosen problem is difficult. It will base that execution on the hardness of the chosen problem. The client will not accept the \( \lambda \) computation results from the \( \lambda \) executions of \( \Pi_{pe} \) except all of them agree with each other and passes their respective verifications. The client will output a wrong computation result only when the \( \lambda \) chosen problems are all easy and the server turns out to be able to break all of them. However, this will happen with probability at most \( 2^{-\lambda} \), which is negligible. Therefore, this modified scheme will be secure merely based on \((2k+1,n)\)-MSDH. Surely, as the price of using a weaker assumption, the resulting scheme incurs a \( \lambda \) overhead to the computation and communication complexities of every party in \( \Pi_{pe} \), which however is acceptable in some scenarios because the \( \lambda \) is independent of \( n \).

### 3.2 Matrix Multiplication

In this section, we present our VC scheme \( \Pi_{mm} \) with input privacy (see Fig. 3) for matrix multiplication. In \( \Pi_{mm} \), the client outsources an \( n \times n \) matrix \( M = (M_{ij}) \) over \( \mathbb{Z}_q \) to the server and may compute \( Mx \) for an input vector \( x = (x_1, \ldots, x_n) \in \mathbb{D} \subseteq \mathbb{Z}_q^n \), where \( q \) is a \( \lambda \)-bit prime not known to the server and \( |\mathbb{D}| = \text{poly}(\lambda) \). Our scheme uses a trilinear map instance \( \Gamma \) with groups of order \( N = pq \), where \( p \) is also a \( \lambda \)-bit prime not known to the server. In \( \Pi_{mm} \), the client gives both \( M \) and its blinded version \( T = (T_{ij}) \) to the server, where \( T \) is computed using the \( \text{PRF}_{\text{dim}} \). It also sets up \( \text{BGN}_3 \). In order to verifiably compute \( Mx \), the client stores \( \tau = (\tau_1, \ldots, \tau_n) \), where each \( \tau_i \) is efficiently computed using the closed form efficiency property of \( \text{PRF}_{\text{dim}} \). It gives \( \sigma = (\text{Enc}(x_1), \ldots, \text{Enc}(x_n)) \) to the server and the server returns \( \rho = (\rho_1, \ldots, \rho_n) = \text{Enc}(Mx) \) along with \( \pi = (\pi_1, \ldots, \pi_n) \). At last, the client decrypts \( \rho_i \) to \( y_i \) and verify if (5) holds for every \( i \in [n] \).

**Correctness.** The correctness of \( \Pi_{mm} \) requires that the client always outputs \( Mx \) as long as the server is honest, i.e., \( y = Mx \) and (5) holds for every \( i \in [n] \). It is shown by the following lemma.

**Lemma 3.4** If the server is honest, then \( y = Mx \) and (5) holds for every \( i \in [n] \).
Proof: Firstly, \( \sigma_j^p = g_1^{px_j} \) for every \( j \in [n] \). Thus we have that
\[
\rho_i^p = \prod_{j=1}^{n} \sigma_j^{M_{ij}} = \prod_{j=1}^{n} g_1^{M_{ij}x_j} = (g_1^p)^{\sum_{j=1}^{n} M_{ij}x_j}
\]
for every \( i \in [n] \). It follows that \( y_i = \sum_{j=1}^{n} M_{ij}x_j \in \mathbb{Z}_q \) for every \( i \in [n] \). Hence, \( y = Mx \). Secondly, we have that
\[
e(\pi_i, g_1^p) = \prod_{j=1}^{n} e(T_{ij}, \sigma_j, g_1^p) = \prod_{j=1}^{n} e(g_2^{g_1^{px_j}}, g_1^p) \cdot e(\prod_{j=1}^{n} F_K(i, j)^{x_j}, g_2^p) = \eta^{\sum_{j=1}^{n} \cdot \pi_i},
\]
i.e., the equality (5) holds. \(\square\)

Security. The security of \( \Pi_{mm} \) requires that no untrusted server can cause the client to accept \( \tilde{y} \notin \{Mx, \perp\} \) with a forged proof. It is based on the 3-co-CDHS assumption for \( \Gamma \) (Lemma 2.4) and the DLIN assumption (Definition 2.6).

Lemma 3.5 If the 3-co-CDHS assumption for \( \Gamma \) and the DLIN assumption both hold, then the scheme \( \Pi_{mm} \) is secure.

Proof: We define three games \( G_0, G_1 \) and \( G_2 \) as below:

\( G_0 \): this is the standard security game \( \text{Exp}^\text{Ver}_A(\Pi, M, \lambda) \) defined in Fig. 1.

\( G_1 \): the only difference between this game and \( G_0 \) is a change to \( \text{ProbGen} \). For any \((x_1, \ldots, x_n)\) queried by the adversary, instead of computing \( \tau \) using the efficient CFE algorithm, the inefficient evaluation of \( \tau_i \) is used, i.e., \( \tau_i = \prod_{j=1}^{n} e(F_K(i, j)^{x_j}, g_2^p) \) for every \( i \in [n] \).

\( G_2 \): the only difference between this game and \( G_1 \) is that the matrix \( T \) is computed as \( T_{ij} = g_1^{g_2^{M_{ij}}} \cdot R_{ij}, \) where \( R_{ij} \leftarrow G_1 \) for every \( i, j \in [n] \).
For every \( i \in \{0, 1, 2\} \), we denote by \( G_i(A) \) the output of game \( i \) when it is run with an adversary \( A \).

The proof of the theorem proceeds by a standard hybrid argument, and is obtained by combining the proofs of the following three claims.

**Claim 1.** We have that \( \Pr[G_2(A) = 1] = \Pr[G_1(A) = 1] \).

The only difference between \( G_1 \) and \( G_0 \) is in the computation of \( \tau \). Due to the correctness of the CFE algorithm, such difference does not change the distribution of the values \( \tau \) returned to the adversary. Therefore, the probabilities that \( A \) wins in both games are identical.

**Claim 2.** We have that \( |\Pr[G_1(A) = 1] - \Pr[G_2(A) = 1]| < \epsilon \).

The only difference between \( G_2 \) and \( G_1 \) is that we replace the pseudorandom group elements \( F_K(i, j) \) with truly random group elements \( R_{ij} \leftarrow G_1 \) for every \( i, j \in [n] \). Clearly, if \( |\Pr[G_1(A) = 1] - \Pr[G_2(A) = 1]| \) is non-negligible, we can construct an simulator \( B \) that simulates \( A \) and breaks the pseudorandom property of PRF with a non-negligible advantage.

**Claim 3.** We have that \( \Pr[G_2(A) = 1] < \epsilon \).

Suppose that there is a PPT adversary \( A \) that wins with non-negligible probability \( \epsilon \) in \( G_2 \). We want to construct a PPT simulator \( B \) that simulates \( A \) and breaks the 3-co-CDHS assumption (see Definition 2.5) with non-negligible probability. The adversary \( B \) takes as input a tuple \( (p, q, \Gamma, h_1^2, h_2^2) \), where \( h_1 = g_1^p, h_2 = g_2^p \) and \( \alpha, \beta \leftarrow \mathbb{Z}_N \). The adversary \( B \) is required to output \( h_2^{\alpha \beta} \). In order to do so, \( B \) simulates \( A \) as below:

1. **(A)** Pick an \( n \times n \) matrix \( M \) and mimic the KeyGen of game \( G_2 \) as below:
   - implicitly set \( a = \alpha \beta \) by computing \( \eta = e(h_1^a, h_2^b) = g_3^{p^2 \alpha \beta} \);
   - pick \( u \leftarrow G_1 \), compute \( h = u^q \) and set up BGN with public key \( (\Gamma, g_1, h) \) and secret key \( p \);
   - pick \( T_{ij} \leftarrow G_1 \) for every \( i, j \in [n] \) and send \( pk = (\Gamma, g_1, h, M, T) \) to \( A \), where \( T = (T_{ij}) \);

2. **(B)** Upon receiving a query \( x = (x_1, \ldots, x_n) \) from \( A \), mimic ProbGen as below:
   - for every \( j \in [n] \), pick \( r_j \leftarrow \mathbb{Z}_N \) and compute \( \sigma_j = g_1^{x_j} h^r_j \);
   - for every \( i, j \in [n] \), compute \( Z_{ij} = e(T_{ij}, g_2^{\alpha x_j}/p^\lambda M_i x_j) \);
   - for every \( i \in [n] \), compute \( \tau_i = \prod_{j=1}^n Z_{ij} \);
   - send \( \sigma = (\sigma_1, \ldots, \sigma_n) \) to \( A \).

It is straightforward to verify that the \( pk, \sigma \) and \( \tau \) generated by \( B \) are identically distributed to those generated by the client in game \( G_2 \). We remark that (A) is the step 1 in \( \text{Exp}^\text{Ver}_A(\Pi, M, \lambda) \) (see Fig. 1) and (B) consists of steps 3 and 4 in \( \text{Exp}^\text{Ver}_A(\Pi, M, \lambda) \). Furthermore, (B) may be run \( l = \text{poly}(\lambda) \) times as described by step 2 of \( \text{Exp}^\text{Ver}_A(\Pi, M, \lambda) \). After \( l \) executions of (B), the adversary \( A \) will provide an input \( \hat{x} = (\hat{x}_1, \ldots, \hat{x}_n) \) on which he is willing to be challenged. Upon receiving \( \hat{x} \), the simulator \( \hat{B} \) mimics \( \text{ProbGen} \) as (B) and gives \( A \) an encoded input \( \hat{\sigma} \). Then the adversary \( A \) may maliciously reply with \( \hat{\rho} = (\hat{\rho}_1, \ldots, \hat{\rho}_n) \) and \( \hat{\pi} = (\hat{\pi}_1, \ldots, \hat{\pi}_n) \) such that \( \text{Verify}(sk, \hat{\tau}, \hat{\rho}, \hat{\pi}) \triangleq \frac{\hat{\eta} \notin \{M \hat{x}, \bot\}}{} \).

On the other hand, an honest server in our VC scheme will reply with \( \hat{\rho} = (\hat{\rho}_1, \ldots, \hat{\rho}_n) \) and \( \hat{\pi} = (\hat{\pi}_1, \ldots, \hat{\pi}_n) \). Due to Lemma 3.4, it must be the case that \( \text{Verify}(sk, \hat{\tau}, \hat{\rho}, \hat{\pi}) \triangleq \frac{\hat{\eta} \notin \{M \hat{x}, \bot\}}{} \). Note that the event \( \hat{\eta} \notin \{M \hat{x}, \bot\} \) occurs with probability \( \epsilon \). Suppose it occurs. Then there is an integer \( i \in [n] \) such that \( \hat{\eta}_i \neq \hat{\eta} \). Note that neither \( \hat{\eta} \) nor \( \hat{\eta} \) is \( \bot \), the equation (5) must be satisfied by both \( (\hat{\eta}, \hat{\pi}) \) and \( (\hat{\eta}, \hat{\pi}) \), which translates into \( e(\hat{\pi}_i, g_1^p) \neq e(\hat{\eta}_i, g_1^p) \). Therefore, this \( \epsilon \) must be negligible in \( \lambda \), i.e., \( \Pr[G_2(A) = 1] < \epsilon \).
Privacy. The input privacy of $\Pi_{mm}$ requires that no untrusted server can distinguish between different inputs of the client. This is formally defined by the experiment $\text{Exp}_A^{\text{Pri}}(\Pi, f, \lambda)$ in Fig. 1. The client in our VC scheme encrypts its input $x$ using BGN3 which is semantically secure under SDA for $\Gamma$. As a result, $\Pi_{mm}$ achieves input privacy under SDA for $\Gamma$ (see Appendix ?? for proof).

Lemma 3.6 If the SDA for $\Gamma$ holds, then $\Pi_{mm}$ achieves the input privacy.

Efficiency. In order to verifiably compute $Mx$ with the cloud, the client computes $n$ ciphertexts $\sigma_1, \ldots, \sigma_k$ under BGN3 and $n$ verification keys $\tau_1, \ldots, \tau_n$ in the execution of $\text{ProbGen}$; it also decrypts $n$ ciphertexts $\rho = \text{Enc}(Mx)$ under BGN3 and then verifies the equation (5). The overall computation of the client will be $O(n) = o(n^2)$ and therefore $\Pi_{pe}$ is outsourced. On the other hand, the server needs to perform $O(n^2)$ multilinear map computations and $O(n)$ exponentiations in each execution of $\text{Compute}$, which is comparable with the VC schemes based on FHE. Based on Lemmas 3.4, 3.5, 3.6 and the efficiency analysis, we have the following theorem.

Theorem 3.2 If the 3-co-CDHS, DLIN and SDA assumptions for $\Gamma$ all hold, then $\Pi_{mm}$ is a VC scheme with input privacy.

3.3 Discussions

A theoretical limitation of our VC schemes $\Pi_{pe}$ and $\Pi_{mm}$ is that the computation results (i.e., $f(\alpha)$ and $Mx$) must belong to a polynomial size domain $M$ since otherwise the client will not be able to decrypt $\rho$ and then verify its correctness. However, we stress that this is not a real limitation when we apply both schemes in outsourcing PIR (see Section 4) where the computation results are either 0 or 1. On the other hand, with $f(x)$ and the knowledge “$f(\alpha) \in M$” (resp. $M$ and the knowledge “$Mx \subseteq M$”), one may argue that the cloud can also learn a polynomial size domain $D$ where $\alpha$ (resp. $Mx$) is drawn from and therefore guess the actual value of $\alpha$ (resp. $x$) with non-negligible probability. However, recall that our privacy experiment $\text{Exp}_A^{\text{Pri}}(\Pi, f, \lambda)$ in Fig. 1 only requires the indistinguishability of different inputs. This is achieved by $\Pi_{pe}$ and $\Pi_{mm}$ (though for polynomial size domains) and suffices for our applications. Furthermore, in Section 3.4, we shall show how to modify $\Pi_{pe}$ and $\Pi_{mm}$ such that the functions (i.e., $f(x)$ and $M$) are encrypted and then given to the cloud. As a consequence, the cloud learns no information on either the outsourced function or input unless it can break the underlying encryption scheme.

3.4 Function Privacy

Note that $\Pi_{pe}$ and $\Pi_{mm}$ only achieve input privacy. We say that a VC scheme achieves function privacy if the server cannot learn any information about the outsourced function. A formal definition of function privacy can be given using an experiment similar to $\text{Exp}_A^{\text{Pri}}(\Pi, f, \lambda)$. Both $\Pi_{pe}$ and $\Pi_{mm}$ can be modified such that function privacy is also achieved. In the modified VC scheme $\Pi'_{pe}$ (see Fig. 4 in Appendix G), the client gives BGN$_{2k+2}$ ciphertexts $\text{Enc}(f) = (\text{Enc}(f_0), \ldots, \text{Enc}(f_n))$ and $\sigma = (\text{Enc}(\alpha), \ldots, \text{Enc}(\alpha^{2k-1}))$ to the server. Then the server can compute $\rho = \text{Enc}(f(\alpha))$ along with a proof $\pi = \text{Enc}(c(s))$ using $\text{Enc}(f)$ and $\sigma$. In the modified VC scheme $\Pi'_{mm}$ (see Fig. 5 in Appendix G), the client gives BGN3 ciphertexts $\text{Enc}(M) = (\text{Enc}(M_i))$ and $\sigma = (\text{Enc}(x_1), \ldots, \text{Enc}(x_n))$ to the server. Then the server can compute $\text{Enc}(\sum_{j=1}^{n} M_{ij} x_j)$ along with a proof $\pi_i$ using $\text{Enc}(M)$ and $\sigma$ for every $i \in [n]$. It is not hard to prove that the schemes $\Pi'_{pe}$ and $\Pi'_{mm}$ are secure and achieve both input and function privacy.

4 Applications

Our VC schemes have application in outsourcing private information retrieval (PIR). PIR [21] allows a client to retrieve any bit $w_i$ of a database $w = w_1 \cdots w_n \in \{0,1\}^n$ from a remote server without
revealing $i$ to the server. In a trivial solution of PIR, the client simply downloads $w$ and extracts $w_i$. The main drawback of this solution is its prohibitive communication cost (i.e. $n$). In [21, 7, 17], PIR schemes with non-trivial communication complexity $o(n)$ have been constructed based on various cryptographic assumptions. However, all of them assume that the server is honest-but-curious. In real-life scenarios, the server may have strong incentive to give the client an incorrect response. Such malicious behaviors may cause the client to make completely wrong decisions in its economic activities (say the client is retrieving price information from a stock database and deciding in which stock it is going to invest). Therefore, PIR schemes that are secure against malicious servers are very interesting. In particular, outsourcing PIR to untrusted clouds in the modern age of cloud computing is very interesting. Both of our VC schemes can provide easy solutions in outsourcing PIR. Using $\Pi_{pe}$, the client can outsource a degree $n$ polynomial $f(x)$ to the cloud, where $f(i) = w_i$ for every $i \in [n]$. To privately retrieve $w_i$, the client can execute $\Pi_{pe}$ with input $i$. In this solution, the communication cost consists of $O(\log n)$ group elements. Using $\Pi_{mm}$, the client can represent the $w$ as a square matrix $M = (M_{ij})$ of order $\sqrt{n}$ and delegate $M$ to the cloud. To privately retrieve a bit $M_{ij}$, the client can execute $\Pi_{mm}$ with input $x \in \{0,1\}^{\sqrt{n}}$, where $x_j = 1$ and all the other bits are 0. In this solution, the communication cost consists of $O(\sqrt{n})$ group elements. Note that in our outsourced PIR schemes, the computation results always belong to $\{0,1\} \subseteq M$. Therefore, the theoretical limitation we discussed in Section 3.3 does not really affect the application of our VC schemes in outsourcing PIR.

5 Conclusions

In this paper, we constructed privacy preserving VC schemes for both univariate polynomial evaluation and matrix multiplication, which have useful applications in outsourcing PIR. Our main tools are the recently developed multilinear maps. A theoretical limitation of our constructions is that the results of the computations should belong to a polynomial-size domain. Although this limitation does not really affect their applications in outsourcing PIR, it is still interesting to remove it in the future works. We also note that our VC schemes are only privately verifiable. It is also interesting to construct privacy preserving VC schemes that are publicly verifiable.

References

[1] Applebaum, B., Ishai, Y., Kushilevitz, E.: From Secrecy to Soundness: Efficient Verification via Secure Computation. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 152–163. Springer, Heidelberg (2010)

[2] Barbosa, M., Farshim, P.: Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 296–312. Springer, Heidelberg (2012)

[3] Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable Delegation of Computation over Large Datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011)

[4] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. In: ITCS 2012, pp. 326–349 (2012)

[5] Boneh, D., Boyen, X., Shacham, H.: Short Group Signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)

[6] Boneh, D., Goh, E., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–342. Springer, Heidelberg (2005)
[7] Cachin, C., Micali, S., Stadler, M.: Computationally Private Information Retrieval with Polylogarithmic Communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999)

[8] Canetti, R., Riva, B., Rothblum, G.: Practical Delegation of Computation Using Multiple Servers. In: CCS 2011, pp. 445–454 (2011)

[9] Choi, S. G., Katz, J., Kumaresan, R., Cid C.: Multi-Client Non-Interactive Verifiable Computation. In: Sahai A. (ed.): TCC 2013, LNCS 7785, pp. 499–518. Springer, Heidelberg (2013)

[10] Chung, K.-M., Kalai, Y.T., Liu, F.-H., Raz, R.: Memory Delegation. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 151–168. Springer, Heidelberg (2011)

[11] Chung, K.-M., Kalai, Y., Vadhan, S.P.: Improved Delegation of Computation Using Fully Homomorphic Encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)

[12] Catalano, D., Fiore, D., Gennaro, R., Vamvourellis, K.: Algebraic (Trapdoor) One Way Functions and Their Applications. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 680–699. Springer, Heidelberg (2013)

[13] Fiore, D., Gennaro, R.: Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications. In: CCS 2012, pp. 501–512 (2012)

[14] Garg, S., Gentry, C., Halevi, S.: Candidate Multilinear Maps from Ideal Lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013)

[15] Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-Based Encryption for Circuits from Multilinear Maps. In: Canetti R., Garay, J. A. (Eds.): CRYPTO 2013. LNCS 8043, pp. 479–499. Springer, Heidelberg (2013)

[16] Gennaro, R., Gentry, C., Parno, B.: Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)

[17] Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005)

[18] Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating Computation: Interactive Proofs for Muggles. In: STOC 2008, pp. 113-122 (2008)

[19] Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. In: STOC 1985, pp. 186–208 (1985)

[20] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010)

[21] Kushilevitz, E., Ostrovsky, R.: Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In: FOCS 1997, pp. 364–373 (1997)

[22] Micali, S.: Computationally Sound Proofs. SIAM Journal of Computing 30(4), 1253–1298 (2000)

[23] Papamanthou, C., Shi, E., Tamassia, R.: Signatures of Correct Computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (2013)
A Proof for Lemma 2.1

Lemma 2.1 If SDA$_i$ holds, then SDA$_j$ holds for every $j = i + 1, \ldots, k$.

Suppose there is a PPT algorithm $A$ such that

$$\left| \Pr[v \leftarrow G_j : A(\Gamma, v) = 1] - \Pr[v \leftarrow G_j : A(\Gamma, v^	heta) = 1] \right| \geq \epsilon$$

(6)

for some integer $j \in \{i + 1, \ldots, k\}$. We shall construct a PPT algorithm $B$ that breaks SDA$_i$ with the same advantage, where the $B$ is given a pair $(\Gamma, u)$ and must decide whether $u \leftarrow G_i$ or $u = \alpha^q$ for $\alpha \leftarrow G_i$. On input $(\Gamma, u)$, the algorithm $B$ computes $v = e(u, g_{j-i})$ and gives $(\Gamma, v)$ to $A$. Upon receiving the output bit $b$ of $A$, the algorithm $B$ will also output $b$. Clearly, $v$ will be uniformly distributed over $G_j$ if $u \leftarrow G_i$. On the other hand, $v = e(\alpha, g_{j-i})^q \in G_j$ will be a random element of order $p$ if $u = \alpha^q$ for $\alpha \leftarrow G_i$. Due to (6), the algorithm $B$ can distinguish the two cases for $v$ with advantage $\epsilon$ and therefore distinguish the two cases for $u$ with the same advantage $\epsilon$ (i.e., break SDA$_i$ with advantage $\epsilon$). Since SDA$_i$ holds, the $\epsilon$ must be negligible in the security parameter $\lambda$. Therefore, SDA$_j$ must also hold for every $j = i + 1, \ldots, k$.

B Proof for Lemma 2.2

Lemma 2.2 If $(k, n)$-MSDH holds, then except for a negligible fraction of the $k$-multilinear map instances $\Gamma_k \leftarrow G(1^\lambda, k)$, either $\Pr[A(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_k^{p/s}] < \text{neg}(\lambda)$ for any PPT algorithm $A$ or $\Pr[A(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_k^{q/s}] < \text{neg}(\lambda)$ for any PPT algorithm $A$, where the probabilities are taken over $s \leftarrow \mathbb{Z}_N$ and $A$’s random coins.

Firstly, under the $(k, n)$-MSDH assumption, for any PPT algorithm $A$, we must have that

$$\Pr[A(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_1^{1/s}] < \text{neg}(\lambda),$$

(7)

where the probability is taken over $\Gamma_k \leftarrow G(1^\lambda, k)$, $s \leftarrow \mathbb{Z}_N$ and $A$’s random coins. This is true because otherwise there would be a PPT algorithm which can find $\alpha = 0$, compute $g_1^{1/(s+\alpha)}$ with non-negligible probability and therefore break the $(k, n)$-MSDH assumption.

Suppose there are $\epsilon$ fraction (of non-negligible) of the $k$-multilinear map instances $\Gamma_k \leftarrow G(1^\lambda, k)$ for each of which there are two PPT algorithms $A_1$ and $A_2$ such that

$$\Pr[A_1(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_k^{p/s}] \quad \text{and} \quad \Pr[A_2(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_k^{q/s}]$$

(8)

are both non-negligible, where the probabilities are taken over $s \leftarrow \mathbb{Z}_N$ and the random coins of $A_1$ and $A_2$. We call such an instance $\Gamma_k$ excellent. Fix an excellent instance $\Gamma_k$. We say that $s \in \mathbb{Z}_N$ is good if $\Pr[A_1(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_k^{p/s}]$ is non-negligible and nice if $\Pr[A_2(p, q, \Gamma_k, g_1, g_1^\ast, \ldots, g_1^n) = g_k^{q/s}]$ is non-negligible, where the probabilities are taken over the random coins of $A_1$ and $A_2$, respectively. For every $s \in \mathbb{Z}_N$, define $X_s$ and $Y_s$ to be two 0-1 random variables, where $X_s = 1$ iff $s$ is good and $Y_s = 1$ iff $s$ is nice. Due to (8), we must have that $\Pr[X_s = 1] \doteq \delta_1$ and $\Pr[Y_s = 1] \doteq \delta_2$ are both non-negligible, where the probabilities are taken over $s \leftarrow \mathbb{Z}_N$. This so because otherwise both probabilities in (8) would be negligible as they must be taken over $s \leftarrow \mathbb{Z}_N$. Below we construct a simulator $S$ that
simulates $A_1$ and $A_2$ and contradicts to (7). The simulator $S$ is given a tuple $(p, q, \Gamma, g, g_1, \ldots, g_n)$ and required to compute $g_k^{1/s}$, where $s \leftarrow \mathbb{Z}_N$. In order to do so the simulator $S$ picks $s_1, s_2 \leftarrow \mathbb{Z}_N$ independently. Let $\hat{s}_1 = ss_1$ and $\hat{s}_2 = ss_2$. It feeds $A_1$ and $A_2$ with $(p, q, \Gamma_k, g, g_1^{\hat{s}_1}, \ldots, g_n^{\hat{s}_1})$ and $(p, q, \Gamma_k, g, g_1^{\hat{s}_2}, \ldots, g_n^{\hat{s}_2})$, respectively. Note that the distributions of both $\hat{s}_1$ and $\hat{s}_2$ are statistically close to the uniform distribution over $\mathbb{Z}_N$. Therefore, with probability $\geq \delta_1 - \text{neg}(\lambda)$, $\hat{s}_1$ is good; with probability $\geq \delta_2 - \text{neg}(\lambda)$, $\hat{s}_2$ is nice. As a consequence, with non-negligible probabilities $A_1$ and $A_2$ will return $g_1^{\hat{s}_1}$ and $g_1^{\hat{s}_2}$, respectively. As a consequence, the simulator $S$ can compute $g_1^{1/s}$ and $g_1^{1/s}$ with non-negligible probability since it knows $s_1$ and $s_2$. Let $u, v$ be integers such that $up + vq = 1$. Then the simulator $S$ can furthermore compute $g_1^{1/s} = (g_1^{1/s})^u (g_1^{1/s})^v$ with the same (non-negligible) probability. In other words, we have that $\Pr[S(p, q, \Gamma, g, g_1, \ldots, g_n) = g_1^{1/s}]$ is non-negligible for the fixed excellent instance $\Gamma$, where the probability is taken over $s \leftarrow \mathbb{Z}_N$ and the random coins of $S$. Since the fraction $\epsilon$ of excellent instances is also non-negligible, we have that $\Pr[S(p, q, \Gamma, g, g_1, \ldots, g_n) = g_1^{1/s}] \geq \Pr[A(p, q, \Gamma, g, g_1, \ldots, g_n) = g_1^{1/s}] \geq \text{neg}(\lambda)$, which is also non-negligible and contradicts to (7). The lemma follows.

C Proof for Lemma 2.3

Lemma 2.2 Let $X$ and $Y$ be two uniform random variables over $\mathbb{Z}_N$. Then the random variable $Z = pXY \mod q$ is statistically close to the uniform random variable $U$ over $\mathbb{Z}_q$, i.e., we have that $\sum_{\omega \in \mathbb{Z}_q} |\Pr[Z = \omega] - \Pr[U = \omega]| < \text{neg}(\lambda)$.

Let $W = XY \mod N$. Then $Z = pW \mod q$. We firstly determine the distribution of $W$. Since both $X$ and $Y$ are uniformly distributed over $\mathbb{Z}_N$, it is easy to see that

$$\Pr[W = w] = \frac{|\{(x, y) \in \mathbb{Z}_N^2 : xy = w \mod N\}|}{N^2}$$

for every $w \in \mathbb{Z}_N$. Therefore, it suffices to determine the number $N_w$ of pairs $(x, y) \in \mathbb{Z}_N^2$ such that $xy = w \mod N$ for every $w \in \mathbb{Z}_N$. Let $S_1 = \{w : w = 0 \mod p \text{ and } w = 0 \mod q\}$, $S_2 = \{w : w \neq 0 \mod p \text{ and } w \neq 0 \mod q\}$, $S_3 = \{w : w = 0 \mod p \text{ and } w \neq 0 \mod q\}$, and $S_4 = \{w : w \neq 0 \mod p \text{ and } w = 0 \mod q\}$. Clearly, we have that $|S_1| = 1, |S_2| = N - p - q + 1, |S_3| = q - 1$ and $|S_4| = p - 1$. Simple calculations show that

$$N_w = \begin{cases} 4N - 2p - 2q + 1, & \text{if } w \in S_1; \\ N - p - q + 1, & \text{if } w \in S_2; \\ 2N - 2p - q + 1, & \text{if } w \in S_3; \\ 2N - p - 2q + 1, & \text{if } w \in S_4. \\ \end{cases}$$

Next, we consider the map $\sigma : \mathbb{Z}_N \to \mathbb{Z}_q$ defined by $\sigma(w) = pw \mod q$ for every $w \in \mathbb{Z}_N$. It is not hard to verify that $\sigma|_{S_1}$ and $\sigma|_{S_3}$ are zero maps, $\sigma|_{S_2}$ is a $(p - 1)$-to-1 map with range $\mathbb{Z}_q^*$, and $\sigma|_{S_4}$ is a 1-to-1 map with range $\mathbb{Z}_q^*$. It follows that $\sigma^{-1}(0) = S_1 \cup S_4$, and

$$\sigma^{-1}(\omega) \subseteq S_2 \cup S_3,$$

$$|\sigma^{-1}(\omega) \cap S_2| = p - 1,$$

$$|\sigma^{-1}(\omega) \cap S_3| = 1$$

for every $\omega \in \mathbb{Z}_q^*$. Due to (9) and (10), the distribution of $Z$ can be described by

$$\Pr[Z = 0] = \frac{1}{N^2}((4N - 2p - 2q + 1)|S_1| + (2N - p - 2q + 1)|S_4|) = \frac{p(2N - p)}{N^2}$$

and

$$\Pr[Z = \omega] = \frac{1}{N^2}((N - p - q + 1)(p - 1) + (2N - 2p - q + 1)) = \frac{p^2(q - 1)}{N^2}.$$
where $\omega \in \mathbb{Z}_q^*$ is arbitrary. It follows that

$$\sum_{\omega \in \mathbb{Z}_q} |\Pr[Z = \omega] - \Pr[U = \omega]| = \left| \frac{p(2N - p)}{N^2} - \frac{1}{q} \right| + \left| \frac{p^2(q - 1)}{N^2} - \frac{1}{q} \right| (q - 1) = \frac{2(q - 1)}{q^2},$$

which is negligible in the security parameter $\lambda$ as $q$ is a $\lambda$-bit prime.

## D Proof for Lemma 2.4

**Lemma 2.3** If 3-MDDH holds, then 3-co-CDHS also holds.

Suppose that the 3-co-CDHS does not hold. Then there is a PPT adversary $A$ such that

$$\Pr[A(p, q, \Gamma, h_1^a, h_2^b) = h_2^{ab} | \epsilon] \geq \epsilon,$$

where $\epsilon$ is non-negligible, and the probability is taken over $a, b \leftarrow \mathbb{Z}_N$ and the random coins of $A$. We shall construct a PPT adversary $B$ that breaks the 3-MDDH. The adversary $B$ is given $(p, q, \Gamma)$ along with five group elements $t = g_1^t, \alpha = g_1^\alpha, \beta = g_1^\beta, \gamma = g_1^\gamma$ and $h \leftarrow G_3$, where $s, a, b, c \leftarrow \mathbb{Z}_N$. It must decide whether $h = g_3^{abc}$ or not. In order to do so, $B$ computes $u = \alpha^\sigma, v = e(\beta, \gamma)^p = g_2^{pbc}$, gives $(p, q, \Gamma, u, v)$ to $A$ and receives $w$ from $A$. Clearly, $u \in G_1$ is a random element of order $q$. On the other hand, due to Lemma 2.3, the distribution of $v = g_2^{pbc}$ is statistically close to the uniform distribution over the order $q$ subgroup of $G_2$. Therefore, equation (11) implies that

$$\Pr[w = g_2^{pabc}] \geq \epsilon - \text{neg}(\lambda),$$

where the probability is taken over $a, b, c \leftarrow \mathbb{Z}_N$ and the random coins of $A$. Given $t \in G_1$ and $w \in G_2$, $B$ will compute $\sigma = e(t, w)$ and compare with $\tau = h^p$. At last, $B$ will output 1 if $\sigma = \tau$ and a random bit $\psi \in \{0, 1\}$ otherwise. Let $E_1$ be the event that $B(p, q, \Gamma, t, \alpha, \beta, \gamma, h) = 1$ when $h = g_3^{abc}$. Then

$$\Pr[E_1] = \Pr[E_1 | w = g_2^{pabc}] \Pr[w = g_2^{pabc}] + \Pr[E_1 | w \neq g_2^{pabc}] \Pr[w \neq g_2^{pabc}] \geq \frac{1}{2} \left(1 - \Pr[w = g_2^{pabc}]\right) \geq \frac{1}{2} + \frac{1}{2}\epsilon - \text{neg}(\lambda).$$

Let $E_2$ be the event that $B(p, q, \Gamma, t, \alpha, \beta, \gamma, h) = 1$ when $h \leftarrow G_3$ is a random group element. Then

$$\Pr[E_2] = \Pr[E_2 | w = g_2^{pabc}] \Pr[w = g_2^{pabc}] + \Pr[E_2 | w \neq g_2^{pabc}] \Pr[w \neq g_2^{pabc}]\leq \frac{1}{2} + \frac{1}{2q}.$$  

Let $h = g_3^\delta$ for $\delta \leftarrow \mathbb{Z}_N$. Let $\eta \in \mathbb{Z}_N$ be fixed. It is not hard to see that $\Pr[h^p = g_3^{p\eta}] = \Pr[\delta \equiv \eta \bmod q] = 1/q$ and $\Pr[h^p = g_3^\eta] = \Pr[p\delta \equiv \eta \bmod N] \leq 1/q$ where the probability is taken over $\delta \in \mathbb{Z}_N$. It follows that

$$\Pr[E_2 | w = g_2^{pabc}] = \Pr[h^p = g_3^{pabc}] \leq \frac{1}{q};$$

$$\Pr[E_2 | w \neq g_2^{pabc}] = \Pr[E_2, h^p = e(t, w) | w \neq g_2^{pabc}] + \Pr[E_2, h^p \neq e(t, w) | w \neq g_2^{pabc}]$$

$$= \Pr[h^p = e(t, w) | w \neq g_2^{pabc}] + \frac{1}{2} \Pr[h^p \neq e(t, w) | w \neq g_2^{pabc}]$$

$$= \frac{1}{2} (1 + \Pr[h^p = e(t, w) | w \neq g_2^{pabc}]) \leq \frac{1}{2} + \frac{1}{2q}.\quad (14)$$

Due to (13) and (14), we have that

$$\Pr[E_2] \leq \frac{1}{q} \Pr[w = g_2^{pabc}] + \left( \frac{1}{2} + \frac{1}{2q} \right) (1 - \Pr[w = g_2^{pabc}])$$

$$\leq \frac{1}{q} + \left( \frac{1}{2} + \frac{1}{2q} \right) (1 - \epsilon + \text{neg}(\lambda)) \leq \frac{1}{2} - \frac{1}{2}\epsilon + \text{neg}(\lambda),$$

(15)
where the second inequality follows from $\Pr[w = g_0^{pabc}] = \epsilon - \text{neg}(\lambda)$ and the third inequality follows from the fact that $1/q$ is negligible in $\lambda$. Due to (12) and (15), we have that

$$|\Pr[E_1] - \Pr[E_2]| \geq \epsilon - \text{neg}(\lambda),$$

which says that 3-MDDH assumption can be broken with advantage at least $\epsilon - \text{neg}(\lambda)$, which is non-negligible as long as $\epsilon$ is non-negligible. Hence, the 3-co-CDHS assumption must hold as long as the 3-MDDH assumption holds.

E Proof for Lemma 3.3

Lemma 3.3 If SDA holds for $\Gamma$, then the scheme $\Pi_{pe}$ achieves the input privacy.

Let $\Gamma$ be the multilineal map instance in $\Pi_{pe}$. Given any input $\alpha$, the only message that contains information about $\alpha$ is $\sigma = (\sigma_1, \ldots, \sigma_k)$, which is sent to the server by the client. Suppose that there is an adversary $A$ that breaks the input privacy of our scheme. Then $A$ must succeed with a non-negligible advantage $\epsilon$ in $\text{Exp}^{\text{Pr}}_A(\Pi, f, \lambda)$, i.e., $\Pr[\text{Exp}^{\text{Pr}}_A(\Pi, f, \lambda) = 1] \geq \frac{1}{2} + \epsilon$. Below we construct a simulator $S$ that breaks the semantic security game of BGN$_{2k+1}$ with non-negligible advantage $\geq \epsilon/k$. In the semantic security game of BGN$_{2k+1}$, the challenger stores the secret key $p$ locally and gives a public key $(\Gamma, g_1, h)$ to $S$. To break BGN$_{2k+1}$, the simulator $S$ simulates $A$ as follows:

(A) Mimic $\text{ProbGen}$: Picks a polynomial $f(x) = f_0 + f_1 x + \cdots + f_n x^n$. Picks $s \leftarrow Z_N$ and computes $t = g_1^{f(s)}$. Stores $sk = (s, t)$ and gives $pk = (\Gamma, g_1, h, g_1, g_1^2, \ldots, g_1^{2^{k-1}}, f)$ to $A$. Note that $sk$ does not include $p, q$ as components of $sk$. This is because $p$ and $q$ are neither known to the simulator $S$ nor used by the simulator $S$.

(B) Mimic $\text{PubProbGen}(sk, \cdot)$: Upon receiving a query $\alpha \in \mathbb{D}$ from $A$, the simulator $S$ picks $r \leftarrow Z_N$ and computes $\sigma_r = g_1^{2^r} h^r$ for every $r \in [k]$. Then gives $\sigma = (\sigma_1, \ldots, \sigma_k)$ to $A$.

(C) Upon receiving $a_0, a_1 \in \mathbb{D}$ from $A$, the simulator $S$ picks $i \leftarrow \{0, 1, \ldots, k-1\}$ and sends $\beta_0 = \alpha_i^{a_0}$ and $\beta_1 = \alpha_i^{a_1}$ to the challenger. The challenger will pick $b \leftarrow \{0, 1\}$ and sends $\text{Enc}(\beta_b)$ to $S$. Upon receiving $\text{Enc}(\beta_b)$ from the challenger, the simulator $S$ gives

$$Z = (\text{Enc}(\alpha_1), \ldots, \text{Enc}(\alpha_i^{2^{i-1}}), \text{Enc}(\beta_b), \text{Enc}(\alpha_0^{2i+1}), \ldots, \text{Enc}(\alpha_0^{2k-1}))$$

to $A$ and learns a bit $b'$ in return.

(D) The simulator $S$ outputs $\hat{b} = 1$ if $b' = 1$ and $\hat{b} = 0$ otherwise. It wins if $\hat{b} = b$.

For every $j \in \{0, 1, \ldots, k\}$, we define the following probability ensemble

$$Z_j = (\text{Enc}(\alpha_1), \ldots, \text{Enc}(\alpha_i^{2^{i-1}}), \text{Enc}(\alpha_0^{a_j}), \ldots, \text{Enc}(\alpha_0^{2k-1})).$$

Then $Z_0 = (\text{Enc}(\alpha_0), \text{Enc}(\alpha_0^{a_2}), \ldots, \text{Enc}(\alpha_0^{2k-1}))$ and $Z_k = (\text{Enc}(\alpha_1), \text{Enc}(\alpha_1^{a_2}), \ldots, \text{Enc}(\alpha_1^{2k-1}))$. The inequality $\Pr[\text{Exp}^{\text{Pr}}_A(\Pi, f, \lambda) = 1] \geq \frac{1}{2} + \epsilon$ implies that $\Pr[A(pk, Z_k) = 1] - \Pr[A(pk, Z_0) = 1] \geq 2\epsilon$. Note that $Z = Z_i$ when $b = 0$ and $Z = Z_{i+1}$ when $b = 1$. Let $E$ be the event that $\hat{b} = b$. Then

$$\Pr[E] = \sum_{l=0}^{k-1} \Pr|E| = l \Pr[i = l] = \frac{1}{k} \sum_{l=0}^{k-1} (\Pr|E| = l, b = 0) \Pr|b = 0| + \Pr|E| = l, b = 1) \Pr|b = 1|)$$

$$= \frac{1}{2k} \sum_{l=0}^{k-1} (\Pr[b' = 0] = l, b = 0) + \Pr[b' = 1] = l, b = 1)$$

$$= \frac{1}{2k} \sum_{l=0}^{k-1} (1 - \Pr[A(pk, Z_l) = 1] + \Pr[A(pk, Z_{l+1}) = 1])$$

$$= \frac{1}{2} + \frac{1}{2k} (\Pr[A(pk, Z_k) = 1] - \Pr[A(pk, Z_0) = 1]) \geq \frac{1}{2} + \frac{\epsilon}{k},$$
i.e., \( B \) breaks the semantic security of \( BGN_{2k+1} \) with a non-negligible advantage \( \epsilon/k \), which is not true when \( SDA \) holds. Hence, \( \Pi_{pe} \) achieves input privacy.

\section{Proof for Lemma 3.6}

\textbf{Lemma 3.6} If the \( SDA \) for \( \Gamma \) holds, then \( \Pi_{mm} \) achieves the input privacy.

We define three games \( G_0, G_1 \) and \( G_2 \) as below:

- \( G_0 \): this is the standard security game \( \text{Exp}^\Pi_A(\Pi, M, \lambda) \) defined in Fig. 1.
- \( G_1 \): the only difference between this game and \( G_0 \) is a change to \( \text{ProbGen} \). For any \( (x_1, \ldots, x_n) \) queried by the adversary, instead of computing \( \tau \) using the efficient CFE algorithm, the inefficient evaluation of \( \tau_i \) is used, i.e., \( \tau_i = \prod_{j=1}^n e(F_K(i, j)^{x_j}, g_2^0) \) for every \( i \in [n] \).

- \( G_2 \): the only difference between this game and \( G_1 \) is that the matrix \( T \) is computed as \( T_{ij} = g_1^{p^2 a M_{ij}} \).

For every \( i \in \{0, 1, 2\} \), we denote by \( G_i(A) \) the output of game \( i \) when it is run with an adversary \( A \). The proof of the theorem proceeds by a standard hybrid argument, and is obtained by combining the proofs of the following three claims.

\textbf{Claim 1.} We have that \( \Pr[G_0(A) = 1] = \Pr[G_1(A) = 1] \).

The only difference between \( G_1 \) and \( G_0 \) is in the computation of \( \tau \). Due to the correctness of the CFE algorithm, such difference does not change the distribution of the values \( \tau \) returned to the adversary. Therefore, the probabilities that \( A \) wins in both games are identical.

\textbf{Claim 2.} We have that \( |\Pr[G_1(A) = 1] - \Pr[G_2(A) = 1]| < \text{neg}(\lambda) \).

The only difference between \( G_2 \) and \( G_1 \) is that we replace the pseudorandom group elements \( F_K(i, j) \) with truly random group elements \( R_{ij} \) for every \( i, j \in [n] \). Clearly, if \( |\Pr[G_1(A) = 1] - \Pr[G_2(A) = 1]| \) is non-negligible, we can construct an simulator \( B \) that simulates \( A \) and breaks the pseudorandom property of PRF with a non-negligible advantage.

\textbf{Claim 3.} We have that \( \Pr[G_2(A) = 1] < \text{neg}(\lambda) \).

Let \( \Gamma \) be the multilinear map instance in \( \Pi_{mm} \). Given any input \( x \), the only message that contains information about \( x \) is \( \sigma = (\sigma_1, \ldots, \sigma_n) \), which is sent to the server by the client. Suppose that there is an adversary \( A \) that breaks the input privacy of our scheme. Then \( A \) must succeed with a non-negligible advantage \( \epsilon \) in \( \text{Exp}^\Pi_A(\Pi, M, \lambda) \), i.e., \( \Pr[\text{Exp}^\Pi_A(\Pi, M, \lambda) = 1] \geq \frac{1}{2} + \epsilon \). Below we construct a simulator \( S \) that breaks the semantic security of \( BGN_3 \) with non-negligible advantage \( \epsilon/n \). In the semantic security game of \( BGN_3 \), the challenger stores the secret key \( p \) locally and gives a public key \( (\Gamma, g_1, h) \) to \( S \). To break \( BGN_3 \), the simulator \( S \) simulates \( A \) as below:

\begin{itemize}
  \item[(A)] \textbf{Mimic ProbGen:} Picks a square matrix \( M = (M_{ij}) \) of order \( n \). Picks a \( \in \mathbb{Z}_N \). Computes \( T_{ij} = g_1^{p^2 a M_{ij}} R_{ij} \) for every \( (i, j) \in [n]^2 \), where \( R_{ij} \leftarrow G_1 \). Stores \( sk = (a, \eta) \) and gives \( pk = (\Gamma, g_1, h, M, T) \) to \( A \), where \( \eta = g_2^{p^2 a} \). Note that \( sk \) does not include \( p, q \) as components of \( sk \). This is because \( p \) and \( q \) are neither known to the simulator \( S \) nor used by the simulator \( S \).
  \item[(B)] \textbf{Mimic PubProbGen} \((sk, \cdot)\): Upon receiving a query \( x = (x_1, \ldots, x_n) \in \mathbb{D} \), picks \( r_j \in \mathbb{Z}_N \) and computes \( \sigma_j = g_2^{r_j h^{x_j}} \) for every \( j \in [n] \). Then gives \( \sigma = (\sigma_1, \ldots, \sigma_n) \) to \( A \).
  \item[(C)] \textbf{Mimic} \( SDA \) \((\cdot, b)\): Upon receiving a query \( x_0, x_1 \in \mathbb{D} \) from \( A \), the simulator \( S \) picks \( i \in [n] \) and sends \( \beta_0 = x_0^i \) and \( \beta_1 = x_1^i \) to the challenger (Here without loss of generality, we can suppose that the Hamming distance between \( x_0 \) and \( x_1 \) is \( n \). Otherwise, we can ignore their equal components and only consider the different components.). The challenger will pick \( b \in \{0, 1\} \) and sends \( \text{Enc(} \beta_b \text{)} \) to \( S \). Upon receiving \( \text{Enc(} \beta_b \text{)} \) from the challenger. The simulator \( S \) gives
    \[
    Z = (\text{Enc}(x_1^1), \ldots, \text{Enc}(x_1^{i-1}), \text{Enc}(\beta_b), \text{Enc}(x_1^{i+1}), \ldots, \text{Enc}(x_1^n))
    \]
\end{itemize}
to \( \mathcal{A} \) and learns a bit \( b' \) in return.

(D) The simulator \( \mathcal{S} \) outputs \( \hat{b} = 1 \) if \( b' = 1 \) and \( \hat{b} = 0 \) otherwise. It breaks BGN3 if \( \hat{b} = b \).

For every \( j \in \{0, 1, \ldots, n\} \), we define the following probability ensemble

\[
Z_j = (\text{Enc}(x_1^j), \ldots, \text{Enc}(x_j^j), \text{Enc}(x_{j+1}^j), \ldots, \text{Enc}(x_n^j)).
\]

Then \( Z_0 = (\text{Enc}(x_0^0), \text{Enc}(x_0^1), \ldots, \text{Enc}(x_0^n)) \) and \( Z_n = (\text{Enc}(x_1^n), \text{Enc}(x_2^n), \ldots, \text{Enc}(x_n^n)) \). The inequality

\[
\Pr[\text{Exp}_A(\mathcal{P}, M, \lambda) = 1] \geq \frac{1}{2} + \epsilon
\]

implies that \( \Pr[\mathcal{A}(pk, Z_n) = 1] - \Pr[\mathcal{A}(pk, Z_0) = 1] \geq 2\epsilon \). Note that \( Z = Z_{i-1} \) when \( b = 0 \) and \( Z = Z_i \) when \( b = 1 \). Let \( E \) be the event that \( \hat{b} = b \). Then

\[
\Pr[E] = \frac{1}{2n} \sum_{l=1}^{n} (Pr[b' = 0 | i = l, b = 0] + Pr[b' = 1 | i = l, b = 1])
\]

\[
= \frac{1}{2n} \sum_{l=1}^{n} (1 - Pr[A(pk, Z_{l-1}) = 1] + Pr[A(pk, Z_l) = 1])
\]

\[
= \frac{1}{2} + \frac{1}{2n} (Pr[A(pk, Z_n) = 1] - Pr[A(pk, Z_0) = 1]) \geq \frac{1}{2} + \frac{\epsilon}{n},
\]

i.e., \( \mathcal{B} \) breaks the semantic security of BGN3 with a non-negligible advantage \( \epsilon/n \), which is not true when SDA holds. Hence, \( \Pi_{mm} \) achieves input privacy.

G Privacy Preserving VC Schemes

This section shows the modifications of our VC schemes \( \Pi_{pe} \) and \( \Pi_{mm} \) that achieve both input and function privacy.

- **KeyGen** \((1^\lambda, f(x))\): Pick \( \Gamma = (N, G_1, \ldots, G_{2k+2}, e, g_1, \ldots, g_{2k+2}) \leftarrow \mathcal{G}(1^\lambda, 2k + 2) \). Pick \( s \leftarrow \mathbb{Z}_N \) and compute \( t = g_1^{f(s)} \). Pick \( u \leftarrow G_1 \) and compute \( h = u^q \), where \( u = g_1^t \) for an integer \( \delta \in \mathbb{Z}_N \). Set up BGN\(_{2k+2}\) with public key \((\Gamma, g_1, h)\) and secret key \( p \). For every \( i \in \{0, 1, \ldots, n\} \), pick \( v_i \leftarrow \mathbb{Z}_N \) and compute \( \gamma_i = g_1^{f_i h^{v_i}} \). Output \( sk = (p, q, s, t) \) and \( pk = (\Gamma, g_1, h, g_1^s, g_1^{g_1^2}, \ldots, g_1^{g_1^{2k+1}}, \gamma) \), where \( \gamma = (\gamma_0, \ldots, \gamma_n) \).

- **ProbGen** \((sk, \alpha)\): For every \( \ell \in [k] \), pick \( r_\ell \leftarrow \mathbb{Z}_N \) and compute \( \sigma_\ell = g_1^{2^\ell - 1} h^{r_\ell} \). Output \( \sigma = (\sigma_1, \ldots, \sigma_k) \) and \( \tau = \perp \) (\( \tau \) is not used).

- **Compute** \((pk, \sigma)\): Compute \( \rho_i = g_2^{\mu_i} \) for every \( i \in \{0, 1, \ldots, n\} \) using the technique in Section 2.2. Compute \( \rho'_i = e(\gamma_i, \rho_i) = g_2^{\mu'_i} \), where \( \mu'_i = (f_i + q\delta v_i)\mu_i \). Compute \( \rho = \prod_{i=0}^n \rho'_i \). Compute \( \pi_{ij} = g_2^{\nu_{ij}} \) using the technique in Section 2.2 for every \( i \in \{0, 1, n - 1\} \) and \( j \in \{0, 1, \ldots, i\} \). Compute \( \pi'_{ij} = e(\gamma_{i+1}, \pi_{ij}) = g_2^{\nu'_{ij}} \), where \( \nu'_{ij} = (f_{i+1} + q\delta v_{i+1})\nu_{ij} \). Set \( \pi = \prod_{i=0}^{n-1} \prod_{j=0}^{i} \pi'_{ij} \). Output \( \rho \) and \( \pi \).

- **Verify** \((sk, \tau, \rho, \pi)\): Compute the \( y \in \mathbb{Z}_q \) such that \( \rho^y = (g_2^{k+1})^y \). If the equality \( e(t/g_1^y, g_2^{k+1}) = e(g_1/y, \pi') \) holds, output \( y \); otherwise, output \( \perp \).

Figure 4: Univariate polynomial evaluation \((\Pi'_{pe})\)
• KeyGen$(1^\lambda, M)$: Pick $\Gamma = (N, G_1, G_2, G_3, e, g_1, g_2, g_3) \leftarrow G(1^\lambda, 3)$. Consider the PRF $dlin$ in Section 2.3. Run KG$(1^\lambda, n)$ and pick a secret key $K$. Pick $a \leftarrow \mathbb{Z}_N$ and compute $T_{ij} = g_1^{g_2^{aM_{ij}}} \cdot F_K(i, j)$ for every $(i, j) \in [n]^2$. Pick $u \leftarrow G_1$ and compute $h = u^q$, where $u = g_1^\delta$ for an integer $\delta \in \mathbb{Z}_N$. Set up BGN$_3$ with public key $(\Gamma, g_1, h)$ and secret key $p$. For every $(i, j) \in [n]^2$, pick $v_{ij} \leftarrow \mathbb{Z}_N$ and compute $\gamma_{ij} = g_4^{g_2^{aM_{ij}}} h^{v_{ij}}$. Output $sk = (p, q, K, a, \eta)$ and $pk = (\Gamma, g_1, h, \gamma, T)$, where $\eta = g_3^{g_2^a}$ and $\gamma = (\gamma_{ij})$.

• ProbGen$(sk, x)$: For every $j \in [n]$, pick $r_j \leftarrow \mathbb{Z}_N$ and compute $\sigma_j = g_2^{g_1^{r_j} h^{r_j}}$. For every $i \in [n]$, compute $\tau_i = e(\prod_{j=1}^{n} F_K(i, j)^{x_j}, g_2^q)$ using the efficient CFE algorithm in Section 2.3. Output $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\tau = (\tau_1, \ldots, \tau_n)$.

• Compute$(pk, \sigma)$: Compute $\rho_i = \prod_{j=1}^{n} e(\gamma_{ij}, \sigma_j)$ and $\pi_i = \prod_{j=1}^{n} e(T_{ij}, \sigma_j)$ for every $i \in [n]$. Output $\rho = (\rho_1, \ldots, \rho_n)$ and $\pi = (\pi_1, \ldots, \pi_n)$.

• Verify$(sk, \tau, \rho, \pi)$: For every $i \in [n]$, compute $y_i$ such that $\rho_i^p = (g_2^p)^{y_i}$. If $e(\pi_i, \gamma_i^\rho) = \eta^{qy_i} \cdot \tau_i$ for every $i \in [n]$, then output $y = (y_1, \ldots, y_n)$; otherwise, output $\perp$.

Figure 5: Matrix multiplication ($\Pi_{mn}'$)