Private Outsourcing of Polynomial Evaluation and Matrix Multiplication Using Multilinear Maps

Liang Feng Zhang and Reihaneh Safavi-Naini

Institute for Security, Privacy and Information Assurance
Department of Computer Science
University of Calgary

Abstract. *Verifiable computation* (VC) allows a computationally weak client to outsource evaluation of a function on many inputs to a powerful but untrusted server. The client invests a large amount of off-line computation to obtain an encoding of its function which is then given to the server. The server returns both the evaluation of the function on the client's input and a proof with which the client can verify the correctness of the evaluation using substantially less effort than doing the evaluation on its own. We consider *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 to achieve function privacy. Our main tool is the recently proposed *mutilinear maps*. We show that the proposed VC schemes can be used to implement verifiable outsourcing of private information retrieval (PIR).

1 Introduction

The rise of cloud computing in recent years has made outsourcing of storage and computation a reality. There are many scenarios where outsourcing computation will provide an attractive solution to the problem at hand. For example, large computations have a severe impact on resources (e.g. battery) of weak clients and outsourcing computation will provide an ideal way of freeing up the resources of the client. A natural question however is how to trust the computation result without trusting the server. The required assurance is not only against malicious behavior of the server but also random faults in the server infrastructure that can result in undetectable error in computation results. Verifiable computation (VC) systems [16] provide such assurance for many scenarios where computation must be delegated. The client in this model invests a large amount of off-line computation and generates an encoding of its function f. Given this encoding and any input α, the server computes and responds with y and a proof that $y = f(\alpha)$. With the server's response, the client can verify if the computation has been carried out correctly using substantially less effort than computing $f(\alpha)$ on its own. The client's off-line computation cost is *amortized* over the evaluations of f on multiple inputs α and will become negligible when computations of the same function is required.

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 costly in practice. However, the outsourced function is given to the server in clear and so function privacy is not provided. Benabbas, Gennaro and Vahlis [3] and several other works [13,12,23] design VC schemes for specific functions without using FHE. One scheme of [3] even achieves 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. We 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 oracle or knowledge of exponent (KoE) type assumptions, both of which are considered as strong [23] and have been carefully avoided in 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]. Recently, Garg, Gentry, and Halvei [14] proposed a candidate mechanism that would approximate multilinear maps for many applications. The proposed instantiation has generated much interest and promise of studying new constructions using a multilinear map abstraction [15]. We use a framework of leveled multilinear maps where one can call a group generator $\mathcal{G}(1^\lambda, k)$ 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 λ-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

$$\Gamma_k = (N, G_1, \ldots, G_k, e, g_1, \ldots, g_k) \leftarrow \mathcal{G}(1^\lambda, k) \tag{1}$$

a random k-multilinear map instance, where $N = pq$ for two λ-bit primes p and q. We start with the BGN encryption scheme (denoted by BGN_2) of Boneh, Goh and Nissim [6] which is based on Γ_2 and semantically secure when the subgroup decision assumption (abbreviated as SDA, see Definition 1) for Γ_2 holds. It is well-known that BGN_2 is both additively homomorphic and multiplicatively homomorphic, i.e., given BGN_2 ciphertexts $\mathsf{Enc}(m_1)$ and $\mathsf{Enc}(m_2)$ one can easily compute $\mathsf{Enc}(m_1+m_2)$ and $\mathsf{Enc}(m_1 m_2)$. Furthermore, BGN_2 supports an unlimited number of additive homomorphic operations: for any integer $k \geq 2$, given BGN_2 ciphertexts $\mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_k)$ one can easily compute $\mathsf{Enc}(m_1 + \cdots + m_k)$. This means one can easily compute $\mathsf{Enc}(f(\alpha))$ from $\mathsf{Enc}(\alpha)$ for any quadratic polynomial $f(x)$. On the other hand, BGN_2 supports only one multiplicative homomorphic operation: one cannot compute $\mathsf{Enc}(m_1 m_2 m_3)$ from $\mathsf{Enc}(m_1), \mathsf{Enc}(m_2)$ and $\mathsf{Enc}(m_3)$. In particular, one cannot compute $\mathsf{Enc}(f(\alpha))$ from $\mathsf{Enc}(\alpha)$ for any polynomial $f(x)$ of degree ≥ 3. In Section 2.2, we introduce BGN_k, which is a generalization of BGN_2 over Γ_k and semantically secure under the SDA for Γ_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 $\mathsf{Enc}(f(\alpha))$ from $\mathsf{Enc}(\alpha)$ 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 Π_{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 α from a polynomial size domain $\mathbb{D}$. We use a polynomial commitment scheme proposed in [20] to construct a basic VC scheme and then show how to convert it into a privacy preserving scheme. The polynomial commitment scheme uses the algebraic property that there is a polynomial $c(x)$ of degree $n-1$ such that $f(x)-f(\alpha) = (x-\alpha)c(x)$. The basic VC scheme works as follows. Let $e : G_1 \times G_1 \to G_2$ 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 $pk = (g_1, g_1^s, \ldots, g_1^{s^n}, f(x))$ to the server, where s is uniformly chosen from $\mathbb{Z}_p$. To verifiably compute $f(\alpha)$, the client gives α to the server and the server returns $\rho = f(\alpha)$ along with a proof $\pi = g_1^{c(s)}$. Finally the client verifies if $e(t/g_1^\rho, g_1) = e(g_1^s/g_1^\alpha, \pi)$. The basic VC scheme is secure under the SBDH assumption [20]. It is the univariate case of the VC schemes for multivariate polynomial evaluation of [23].

In Π_{pe}, the α should be hidden from the server (e.g., the client gives $\mathsf{Enc}(\alpha)$ to the server) which makes the server's computation of ρ and π (as in the basic VC scheme) impossible. Instead, the best one can expect is to compute a ciphertext $\rho = \mathsf{Enc}(f(\alpha))$ from $\mathsf{Enc}(\alpha)$ and $f(x)$. This can be achieved if the underlying encryption scheme Enc is an FHE which we want to avoid. On the other hand, a proof π that the computation of ρ has been carried out correctly should be given to the client. To the best of our knowledge, for generating such a proof π, 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 $\mathsf{Enc}(\alpha)$ and $f(x)$ and then generate $\rho = \mathsf{Enc}(f(\alpha))$. In $\Pi_{\rm pe}$, the client picks a $(2k+1)$-multilinear map instance Γ as in (1). It stores $t = g_1^{f(s)}$ and gives $\boldsymbol{\xi} = (g_1, g_1^s, g_1^{s^2} \ldots, g_1^{s^{2^{k-1}}})$ and $f(x)$ to the server, where $k = \log\lceil n+1 \rceil$. It also sets up 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 $\rho = \mathsf{Enc}(f(\alpha))$ along with a proof $\pi = \mathsf{Enc}(c(s))$, where $\sigma_\ell = \mathsf{Enc}(\alpha^{2^{\ell-1}})$ for every $\ell \in [k]$. Note that $f(\alpha)$ and $c(s) = (f(s)-f(\alpha))/(s-\alpha)$ are both polynomials in α and s. In Section 2.2, we show how the server can compute ρ and π from $f(x), \sigma$ and $\boldsymbol{\xi}$. Upon receiving (ρ, π), the client decrypts ρ to y and verifies if $e(t/g_1^y, g_{2k}^p) = e(g_1^s/g_1^\alpha, \pi^p)$. We can show the security and privacy of $\Pi_{\rm pe}$ under the assumptions $(2k+1, n)$-MSDHS (see Definition 2) and SDA (see Definition 1).

Matrix Multiplication. In Section 3.2 we propose a VC scheme $\Pi_{\rm mm}$ 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 $\mathsf{PRF}_{\rm dlin} = (\mathsf{KG}, \mathsf{F})$ over a trilinear map instance Γ, where for any secret key K generated by KG, F_K is a function with domain $[n]^2$ and range G_1. In $\Pi_{\rm mm}$, the client gives both M and its blinded version $T = (T_{ij})$ to the server, where $T_{ij} = g_1^{p^2 a M_{ij}} \cdot \mathsf{F}_K(i,j)$ for every $(i,j) \in [n]^2$ and a is randomly chosen from $\mathbb{Z}_N$ and is fixed for any $(i,j) \in [n]^2$. It also sets up BGN_3. In order to verifiably compute Mx, the client stores $\tau_i = \prod_{j=1}^n e(\mathsf{F}_K(i,j), g_2^{px_j})$ for every $i \in [n]$, where τ_i can be efficiently computed using the closed form efficiency property of $\mathsf{PRF}_{\rm dlin}$. It gives the ciphertexts $\sigma = (\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_n))$ to the server and the server returns $\rho_i = \mathsf{Enc}(\sum_{j=1}^n M_{ij} x_j)$ along with a proof $\pi_i = \prod_{j=1}^n e(T_{ij}, \mathsf{Enc}(x_j))$ for every $i \in [n]$. Upon receiving $\rho = (\rho_1, \ldots, \rho_n)$ and $\pi = (\pi_1, \ldots, \pi_n)$, the client can decrypt ρ_i to y_i and verify if $e(\pi_i, g_1^p) = \eta^{py_i} \cdot \tau_i$ for every $i \in [n]$, where $\eta = g_3^{p^2 a}$. Finally, we can show the security and privacy of $\Pi_{\rm mm}$ under the assumptions 3-co-CDHS (see Definition 5), DLIN (see Definition 5) and SDA.

Applications. Our VC schemes can be used to implement verifiable outsourcing of private information retrieval (PIR) where a client stores a large database w (which is modeled as a bit string $w = w_1 \cdots w_n \in \{0,1\}^n$) with the cloud and later retrieves a bit without revealing which bit he is interested in. This is a scenario that is well motivated by real life applications. 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 $\Pi_{\rm pe}$, 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 $\Pi_{\rm mm}$. 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. We note that decrypting $\rho = \mathsf{Enc}(f(\alpha))$ in $\Pi_{\rm pe}$ requires computing discrete logarithms (see Section 2.2). Hence, the $f(\alpha)$ should be from a polynomial-size domain $\mathbb{M}$ since otherwise the client will not be able to decrypt ρ 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 in many applications such as outsourcing PIR where $f(\alpha) \in \{0, 1\}$, the limitation does not affect the applicability of our VC schemes in practice. One may also argue that with $f(x)$ and the knowledge of "$f(\alpha) \in \mathbb{M}$", the server may learn a polynomial size domain $\mathbb{D}$ where α is drawn from and therefore guess α with non-negligible probability. We note that the input privacy (see Definition 9) achieved by $\Pi_{\rm pe}$ is indistinguishability based and does not contradict to the above argument. In Section 3.4, we show how to modify $\Pi_{\rm pe}$ such that $f(x)$ is also hidden and therefore prevent the cloud from learning any information about α. Discussions similar to above are also applicable to $\Pi_{\rm mm}$.

Extensions. In Section 3.4, we modify $\Pi_{\rm pe}$ and $\Pi_{\rm mm}$ such that the function privacy is also achieved. In the modified schemes $\Pi'_{\rm pe}$ (see Fig. 4) and $\Pi'_{\rm mm}$ (see Fig. 5), the outsourced functions are encrypted and then given to the server. The basic approach is to increase 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 $\Pi'_{\rm pe}$ and $\Pi'_{\rm mm}$ 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. Using 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 efficient. None of them provides 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 but not input 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. Using 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.* The schemes of [3,25,13,12,23] do not provide input privacy. Extensions of VC schemes to other different models 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 ω is uniformly chosen from A. Let λ be a security parameter. We denote by $\mathsf{neg}(\lambda)$ the class of functions $\epsilon(\cdot)$ that are negligible in λ, i.e., for every constant $c > 0$, $\epsilon(\lambda) < \lambda^{-c}$ as long as λ is large enough. We denote by $\mathsf{poly}(\lambda)$ the class of polynomial functions in λ.

2.1 Multilinear Maps and Assumptions

In this section, we review several cryptographic assumptions concerning multilinear maps. Given the Γ_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 1. (SDA) *We say that* SDA$_i$ *holds if for any probabilistic polynomial time* (PPT) *algorithm* $\mathcal{A}$, $|\Pr[\mathcal{A}(\Gamma_k, u) = 1] - \Pr[\mathcal{A}(\Gamma_k, u^q) = 1]| < \mathsf{neg}(\lambda)$, *where the probabilities are taken over* $\Gamma_k \leftarrow \mathcal{G}(1^\lambda, k)$, $u \leftarrow G_i$ *and* $\mathcal{A}$*'s random coins. We say that* SDA *holds if* SDA$_i$ *holds for every* $i \in [k]$.

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

In the full version [26], we are able to construct a privacy preserving VC scheme for univariate polynomial evaluation which is secure based on (k,n)-MSDH. Under the (k,n)-MSDH assumption, the following lemma (see [26] 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 1. *If (k, n)-MSDH holds, then except for a negligible fraction of the k-multilinear map instances $\Gamma_k \leftarrow \mathcal{G}(1^\lambda, k)$, either* $\Pr[\mathcal{A}(p, q, \Gamma_k, g_1, g_1^s, \dots, g_1^{s^n}) = g_k^{p/s}] < \mathsf{neg}(\lambda)$ *for any PPT algorithm $\mathcal{A}$ or* $\Pr[\mathcal{A}(p, q, \Gamma_k, g_1, g_1^s, \dots, g_1^{s^n}) = g_k^{q/s}] < \mathsf{neg}(\lambda)$ *for any PPT algorithm $\mathcal{A}$, where the probabilities are taken over $s \leftarrow \mathbb{Z}_N$ and $\mathcal{A}$'s random coins.*

Due to Lemma 1, 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 Π_{pe} (see Fig. 2) that is more efficient than the one based on (k, n)-MSDH. In this version, we only present the scheme Π_{pe} based on (k, n)-MSDHS.

Definition 3. ((k, n)-MSDHS) *For any PPT algorithm $\mathcal{A}$,* $\Pr[\mathcal{A}(p, q, \Gamma_k, g_1, g_1^s, \dots, g_1^{s^n}) = g_k^{p/s}] < \mathsf{neg}(\lambda)$, *where the probability is taken over $\Gamma_k \leftarrow \mathcal{G}(1^\lambda, k)$, $s \leftarrow \mathbb{Z}_N$ and $\mathcal{A}$'s random coins.*

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

Definition 4. (k-MDDH) *For any PPT algorithm $\mathcal{A}$,* $|\Pr[\mathcal{A}(p, q, \Gamma_k, g_1^s, g_1^{a_1}, \dots, g_1^{a_k}, g_k^{sa_1 \cdots a_k}) = 1] - \Pr[\mathcal{A}(p, q, \Gamma_k, g_1^s, g_1^{a_1}, \dots, g_1^{a_k}, h) = 1]| < \mathsf{neg}(\lambda)$, *where the probabilities are taken over $\Gamma_k \leftarrow \mathcal{G}(1^\lambda, k)$, $s, a_1, \dots, a_k \leftarrow \mathbb{Z}_N, h \leftarrow G_k$ and $\mathcal{A}$'s random coins.*

Let $\Gamma_3 = (N, G_1, G_2, G_3, e, g_1, g_2, g_3) \leftarrow \mathcal{G}(1^\lambda, 3)$ be a random trilinear map instance. Let $h_1 = g_1^p$ and $h_2 = g_2^p$. The trilinear co-Computational Diffie-Hellman assumption for the order q Subgroups (3-co-CDHS) says that given $h_1^a \leftarrow G_1$ and $h_2^b \leftarrow G_2$, it is difficult for any PPT algorithm to compute h_2^{ab}.

Definition 5. (3-co-CDHS) *For any PPT algorithm $\mathcal{A}$,* $\Pr[\mathcal{A}(p, q, \Gamma_3, h_1^a, h_2^b) = h_2^{ab}] < \mathsf{neg}(\lambda)$, *where the probability is taken over $\Gamma_3 \leftarrow \mathcal{G}(1^\lambda, 3)$, $a, b \leftarrow \mathbb{Z}_N$ and $\mathcal{A}$'s random coins.*

The following lemma shows that 3-co-CDHS is not a new assumption but weaker than 3-MDDH (see [26] for the proof).

Lemma 2. *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 Γ_3.

Definition 6. (DLIN) *Let G be a cyclic group of order $N = pq$, where p, q are λ-bit primes. For any PPT algorithm $\mathcal{A}$,* $|\Pr[\mathcal{A}(p, q, u, v, w, u^a, v^b, w^{a+b}) = 1] - \Pr[\mathcal{A}(p, q, u, v, w, u^a, v^b, w^c) = 1]| < \mathsf{neg}(\lambda)$, *where the probabilities are taken over $u, v, w \leftarrow G$, $a, b, c \leftarrow \mathbb{Z}_N$ and $\mathcal{A}$'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. $|\mathbb{M}| = \mathsf{poly}(\lambda)$. Below we generalize BGN$_2$ and define BGN$_k$ = $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for any $k \geq 2$, where

– $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda, k)$ is a key generation algorithm. It picks $\mathit{\Gamma}_k$ as in (1) and then outputs both a public key $pk = (\mathit{\Gamma}_k, g_1, h)$ and a secret key $sk = p$, where $h = u^q$ for $u \leftarrow G_1$.
– $c \leftarrow \mathsf{Enc}(pk, m)$ 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 $r \leftarrow \mathbb{Z}_N$.
– $m \leftarrow \mathsf{Dec}(sk, c)$ 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 $pk = (\mathit{\Gamma}_k, g_i, h)$ and replacing any occurrence of g_1 with g_i, where $h = u^q$ 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 $e_i : G_1 \times \cdots \times G_1 \to G_i$ such that $e_i(g_1^{a_1}, \ldots, g_1^{a_i}) = g_i^{a_1 \cdots a_i}$ for any $a_1, \ldots, a_i \in \mathbb{Z}_N$. Firstly, we shall see that BGN$_k$ allows us to compute $\mathsf{Enc}(m_1 \cdots m_k)$ from $\mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_k)$. Suppose $\mathsf{Enc}(m_\ell) = g_1^{m_\ell} h^{r_\ell}$ for every $\ell \in [k]$, where $h = g_1^{q\delta}$ for some $\delta \in \mathbb{Z}_N$ and $r_\ell \leftarrow \mathbb{Z}_N$. Let $h_k = e_k(h, g_1, \ldots, g_1) = g_k^{q\delta}$. Then $e_k(\mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_k)) = g_k^m h_k^r$ is a ciphertext of $m = m_1 \cdots m_k$ in G_k, where $r = \frac{1}{q\delta}(\prod_{\ell=1}^k (m_\ell + q\delta r_\ell) - m)$.

Computing ρ with Reduced Multi-linearity Level. In $\mathit{\Pi}_{\text{pe}}$, 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 $\rho = \mathsf{Enc}(f(\alpha))$, where $k = \lceil \log(n+1) \rceil$. Below we show how to compute the ρ using σ and $f(x)$. Suppose $\sigma_\ell = g_1^{\alpha^{2^{\ell-1}}} h^{r_\ell}$ for every $\ell \in [k]$, where $h = g_1^{q\delta}$ for some $\delta \in \mathbb{Z}_N$ and $r_\ell \leftarrow \mathbb{Z}_N$. Clearly, any $i \in \{0, 1, \ldots, n\}$ has a binary representation $(i_1, \ldots, i_k)$ such that $i = \sum_{\ell=1}^k i_\ell 2^{\ell-1}$. Then $\alpha^i = \alpha^{i_1} \cdot (\alpha^2)^{i_2} \cdots (\alpha^{2^{k-1}})^{i_k}$ is the product of $i_1 + \cdots + i_k$ 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 $\rho_i \triangleq e_k(\phi_1, \ldots, \phi_k) = g_k^{\mu_i} = g_k^m h_k^r$ is a ciphertext of $m = \alpha^i$ under BGN$_{2k+1}$, where $\mu_i = \prod_{\ell=1}^k (\alpha^{2^{\ell-1}} + q\delta r_\ell)^{i_\ell}$ and $r = \frac{1}{q\delta}(\mu_i - m)$. Thus, $\rho = \prod_{i=0}^n \rho_i^{f_i}$ is a ciphertext of $f(\alpha)$ under BGN$_{2k+1}$.

Computing π with Reduced Multi-linearity Level. In $\mathit{\Pi}_{\text{pe}}$, $k+1$ group elements $\boldsymbol{\xi} = (g_1, g_1^s, \ldots, g_1^{s^{2^{k-1}}})$ are also known to the server as part of the public key, where $s \leftarrow \mathbb{Z}_N$. The server must return $\pi = \mathsf{Enc}(c(s))$ as the proof that $\rho = \mathsf{Enc}(f(\alpha))$ has been correctly computed. Below we show how to compute π using $\boldsymbol{\xi}$ and σ. Note that $c(s) = (f(s) - f(\alpha))/(s - \alpha) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} f_{i+1} \alpha^j s^{i-j}$.

It suffices to show how to compute $\pi_{ij} \triangleq \mathsf{Enc}(f_{i+1}\alpha^j s^{i-j})$ for every $i \in \{0,1,\ldots,n-1\}$ and $j \in \{0,1,\ldots,i\}$. Let $(j_1,\ldots,j_k)$, $(i_1,\ldots,i_k) \in \{0,1\}^k$ be the binary representations of j and $i-j$, respectively. Let $\phi_\ell = \sigma_\ell$ if $j_\ell = 1$ and $\phi_\ell = g_1$ otherwise. Let $\psi_\ell = g_1^{s^{2^{\ell-1}}}$ if $i_\ell = 1$ and $\psi_\ell = g_1$ otherwise. Then it is easy to see that $\pi_{ij} = e(e_k(\phi_1,\ldots,\phi_k), e_k(\psi_1,\ldots,\psi_k)) = g_{2k}^{\nu_{ij}} = g_{2k}^m h_{2k}^r$ is a ciphertext of $m = \alpha^j s^{i-j}$, where $\nu_{ij} = s^{i-j}\prod_{\ell=1}^k(\alpha^{2^{\ell-1}} + q\delta r_\ell)^{j_\ell}, h_{2k} = g_{2k}^{q\delta}$ and $r = \frac{1}{q\delta}(\nu_{ij} - m)$. Let $\nu = \sum_{i=0}^{n-1}\sum_{j=0}^{i} f_{i+1}\nu_{ij}$. Thus, $\pi \triangleq g_{2k}^{\nu} = \prod_{i=0}^{n-1}\prod_{j=0}^{i}\pi_{ij}^{f_{i+1}} = \mathsf{Enc}(c(s))$.

2.3 Algebraic PRFs with Closed Form Efficiency

In Π_{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 $\mathsf{PRF} = (\mathsf{KG}, \mathsf{F})$, where $\mathsf{KG}(1^\lambda, \mathsf{pp})$ generates a secret key K from any public parameter pp and $\mathsf{F}_K : I \to G$ is a function with domain I and range G (both specified by pp). We say that PRF has pseudorandom property if for any pp and any PPT algorithm $\mathcal{A}$, it holds that $|\Pr[\mathcal{A}^{\mathsf{F}_K(\cdot)}(1^\lambda, \mathsf{pp}) = 1] - \Pr[\mathcal{A}^{\mathsf{R}(\cdot)}(1^\lambda, \mathsf{pp}) = 1]| < \mathsf{neg}(\lambda)$, where the probabilities are taken over the randomness of $\mathsf{KG}, \mathcal{A}$ and the random function $\mathsf{R} : I \to G$. Consider an arbitrary computation 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 $\mathsf{Comp}(R_1,\ldots,R_n,x_1,\ldots,x_n)$ takes time t. Let $z = (z_1,\ldots,z_n) \in I^n$. We say that PRF has closed form efficiency for (Comp, z) if there is an efficient algorithm CFE such that $\mathsf{CFE}_{\mathsf{Comp},z}(K,x) = \mathsf{Comp}(\mathsf{F}_K(z_1),\ldots,\mathsf{F}_K(z_n),x_1,\ldots,x_n)$ 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 $\mathsf{PRF}_{\mathrm{dlin}}$ based on the DLIN assumption for the bilinear groups. We generalize it over trilinear groups. In the generalized setting, KG generates $\Gamma_3 \leftarrow \mathcal{G}(1^\lambda, 3)$, 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 $\mathsf{F}_K(i,j) = A_j^{\alpha_i} B_j^{\beta_i}$. The closed form efficiency of $\mathsf{PRF}_{\mathrm{dlin}}$ is described as below. Let $x = (x_1,\ldots,x_n) \in \mathbb{Z}_N^n$. The computation Comp we consider is computing $\prod_{j=1}^n \mathsf{F}_K(i,j)^{x_j}$ for all $i \in [n]$. Clearly, it requires $\Omega(n^2)$ 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 $\prod_{j=1}^n \mathsf{F}_K(i,j)^{x_j} = A^{\alpha_i}B^{\beta_i}$ for every $i \in [n]$. Computing $A^{\alpha_i}B^{\beta_i}$ requires 2 exponentiations and hence the $\mathsf{PRF}_{\mathrm{dlin}}$ has closed form efficiency for (Comp, z), where $z = \{(i,j) : i,j \in [n]\}$. The $\mathsf{PRF}_{\mathrm{dlin}}$ in [13] is pseudorandom merely based on the DLIN for bilinear groups. Similarly, the generalized $\mathsf{PRF}_{\mathrm{dlin}}$ is also pseudorandom based on the DLIN assumption for trilinear groups. Consequently, we have the following lemma.

Lemma 3. *If DLIN holds in the trilinear setting, then* $\mathsf{PRF}_{\mathrm{dlin}}$ *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 $\Pi = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ is defined by four algorithms, where

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$ takes as input a security parameter λ and a function f, and generates both a public key pk and a secret key sk;
- $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(sk, x)$ takes as input the secret key sk and an input x, and generates both an encoded input σ and a verification key τ;
- $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \sigma)$ takes as input the public key pk and an encoded input σ, and produces both an encoded output ρ and a proof π;
- $\{f(x), \perp\} \leftarrow \mathsf{Verify}(sk, \tau, \rho, \pi)$ 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 $\perp$ (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 $\mathcal{F}$ be a family of functions.

Definition 7. *The scheme Π is said to be $\mathcal{F}$-*correct *if for any $f \in \mathcal{F}$, any $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$, any input x to f, any $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(sk, x)$, any $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \sigma)$, it holds that $f(x) = \mathsf{Verify}(sk, \tau, \rho, \pi)$.*

```
Experiment $\mathsf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\Pi, f, \lambda)$
1. $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$;
2. for $i = 1$ to $l = \mathsf{poly}(\lambda)$ do
3.     $x_i \leftarrow \mathcal{A}(pk, x_1, \sigma_1, \ldots, x_{i-1}, \sigma_{i-1})$;
4.     $(\sigma_i, \tau_i) \leftarrow \mathsf{ProbGen}(sk, x_i)$;
5. $\hat{x} \leftarrow \mathcal{A}(pk, x_1, \sigma_1, \ldots, x_l, \sigma_l)$
6. $(\hat{\sigma}, \hat{\tau}) \leftarrow \mathsf{ProbGen}(sk, \hat{x})$;
7. $(\bar{\rho}, \bar{\pi}) \leftarrow \mathcal{A}(pk, x_1, \sigma_1, \ldots, x_l, \sigma_l, \hat{\sigma})$
8. $\bar{y} \leftarrow \mathsf{Verify}(sk, \hat{\tau}, \bar{\rho}, \bar{\pi})$;
9. output 1 if $\bar{y} \notin \{f(\hat{x}), \perp\}$ and 0 otherwise.
```

```
Experiment $\mathsf{Exp}^{\mathsf{Pri}}_{\mathcal{A}}(\Pi, f, \lambda)$
1. $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$;
2. $(x_0, x_1) \leftarrow \mathcal{A}^{\mathsf{PubProbGen}(sk,\cdot)}(pk)$;
3. $b \leftarrow \{0, 1\}$;
4. $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(sk, x_b)$;
5. $b' \leftarrow \mathcal{A}^{\mathsf{PubProbGen}(sk,\cdot)}(pk, x_0, x_1, \sigma)$
6. output 1 if $b' = b$ and 0 otherwise.
Remark: $\mathsf{PubProbGen}(sk, \cdot)$ takes as input $x$, runs $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(sk, x)$ and returns $\sigma$.
```

Fig. 1. Experiments for security and privacy [16]

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 $\mathsf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\Pi, f, \lambda)$ (see Fig. 1) where the challenger plays the role of the client and the adversary $\mathcal{A}$ plays the role of the untrusted server.

Definition 8. *The scheme Π is said to be $\mathcal{F}$-*secure *if for any $f \in \mathcal{F}$ and any PPT adversary $\mathcal{A}$, it holds that* $\Pr[\mathsf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\Pi, f, \lambda) = 1] < \mathsf{neg}(\lambda)$.

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 $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Pri}}(\Pi, f, \lambda)$ (see Fig. 1) where the challenger plays the role of the client and the adversary $\mathcal{A}$ plays the role of the untrusted server.

Definition 9. *The scheme Π is said to achieve* input privacy *if for any function $f \in \mathcal{F}$, any PPT algorithm $\mathcal{A}$, it holds that* $\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Pri}}(\Pi, f, \lambda) = 1] < \mathsf{neg}(\lambda)$.

Efficiency. The algorithms $\mathsf{ProbGen}$ 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 10. *The scheme Π is said to be* outsourced *if for any $f \in \mathcal{F}$ and any input x to f, the running time of* $\mathsf{ProbGen}$ *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 $\Pi_{\rm pe}$ with input privacy (see Fig. 2) for univariate polynomial evaluation. In $\Pi_{\rm pe}$, 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 λ-bit prime not known to the server and $|\mathbb{D}| = \mathsf{poly}(\lambda)$. Our scheme uses a $(2k+1)$-multilinear map instance Γ with groups of order $N = pq$, where $k = \lceil \log(n+1) \rceil$ and p is also a λ-bit prime not known to the server. The client stores $t = g_1^{f(s)}$ and gives $(g_1^s, g_1^{s^2} \ldots, g_1^{s^{2^{k-1}}}, f)$ to the server, where $s \leftarrow \mathbb{Z}_N$. It also sets up BGN_{2k+1} based on Γ. In order to verifiably compute $f(\alpha)$, the client gives $\sigma = (\sigma_1, \ldots, \sigma_k)$ to the server and the server returns $\rho = \mathsf{Enc}(f(\alpha))$ along with $\pi = \mathsf{Enc}(c(s))$, where $\sigma_\ell = \mathsf{Enc}(\alpha^{2^{\ell-1}})$ for every $\ell \in [k]$ and (ρ, π) is computed using the techniques in Section 2.2. At last, the client decrypts ρ to y and verifies if the equation (2) holds.

Correctness. The correctness of $\Pi_{\rm pe}$ 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 (see [26] for the proof).

Lemma 4. *If the server is honest, then $y = f(\alpha)$ and (2) holds.*

Security. The security of $\Pi_{\rm pe}$ 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).

Lemma 5. *If $(2k+1, n)$-MSDHS holds for Γ, then the scheme $\Pi_{\rm pe}$ is secure.*

Proof. Suppose that $\Pi_{\rm pe}$ is not secure. Then there is a PPT adversary $\mathcal{A}$ that breaks its security with non-negligible probability ϵ_1. We shall construct a PPT simulator $\mathcal{B}$ that simulates $\mathcal{A}$ and breaks the $(2k+1, n)$-MSDHS for Γ.

– $\mathsf{KeyGen}(1^\lambda, f(x))$: Pick $\Gamma = (N, G_1, \ldots, G_{2k+1}, e, g_1, \ldots, g_{2k+1}) \leftarrow \mathcal{G}(1^\lambda, 2k+1)$. Pick $s \leftarrow \mathbb{Z}_N$ and compute $t = g_1^{f(s)}$. Pick $u \leftarrow G_1$ and compute $h = u^q$, where $u = g_1^\delta$ for an integer $\delta \in \mathbb{Z}_N$. Set up BGN_{2k+1} with public key (Γ, g_1, h) and secret key p. Output $sk = (p, q, s, t)$ and $pk = (\Gamma, g_1, h; g_1^s, g_1^{s^2}, \ldots, g_1^{s^{2^{k-1}}}; f)$.
– $\mathsf{ProbGen}(sk, \alpha)$: For every $\ell \in [k]$, pick $r_\ell \leftarrow \mathbb{Z}_N$ and compute $\sigma_\ell = g_1^{\alpha^{2^{\ell-1}}} h^{r_\ell}$. Output $\sigma = (\sigma_1, \ldots, \sigma_k)$ and $\tau = \perp$ (τ is not used).
– $\mathsf{Compute}(pk, \sigma)$: Compute $\rho_i = g_k^{\mu_i}$ for every $i \in \{0, 1, \ldots, n\}$ using the technique in Section 2.2. Compute $\rho = \prod_{i=0}^n \rho_i^{f_i}$. Compute $\pi_{ij} = g_{2k}^{\nu_{ij}}$ 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 ρ and π.
– $\mathsf{Verify}(sk, \tau, \rho, \pi)$: Compute the $y \in \mathbb{Z}_q$ such that $\rho^p = (g_k^p)^y$. If

$$e\Big(t/g_1^y, g_{2k}^p\Big) = e\Big(g_1^s/g_1^\alpha, \pi^p\Big), \tag{2}$$

then output y; otherwise, output $\perp$.

Fig. 2. Univariate polynomial evaluation (Π_{pe})

The simulator $\mathcal{B}$ takes as input $(p, q, \Gamma, g_1, g_1^s, \ldots, g_1^{s^n})$, where $s \leftarrow \mathbb{Z}_N$. The simulator $\mathcal{B}$ is required to output $g_{2k+1}^{p/s}$. In order to do so, $\mathcal{B}$ simulates $\mathcal{A}$ as below:

(A) Pick a polynomial $f(x) = f_0 + f_1 x + \cdots + f_n x^n \in \mathbb{Z}_q[x]$. Pick $u \leftarrow G_1$, compute $h = u^q$ and set up BGN_{2k+1} with public key (Γ, g_1, h) and secret key p. Pick $\beta \leftarrow \mathbb{D}$ and implicitly set $\hat{s} = s + \beta$ ($\hat{s}$ is not known to $\mathcal{B}$). Mimic KeyGen by sending $pk = (\Gamma, g_1, h, g_1^{\hat{s}}, g_1^{\hat{s}^2} \ldots, g_1^{\hat{s}^{2^{k-1}}}, f)$ to $\mathcal{A}$ (note that $\mathcal{B}$ can compute $g_1^{\hat{s}^{2^{\ell-1}}}$ for every $\ell \in [k]$ based on the knowledge of β and $g_1, g_1^s, \ldots, g_1^{s^n}$). 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 $\mathcal{B}$ nor used by $\mathcal{B}$);

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

It is trivial to verify that the pk and σ generated by $\mathcal{B}$ are identically distributed to those generated by the client in an execution of Π_{pe}. We remark that (A) is the step 1 in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\Pi, f, \lambda)$ (see Fig. 1) and (B) consists of steps 3 and 4 in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\Pi, f, \lambda)$. Furthermore, (B) may be run $l = \mathsf{poly}(\lambda)$ times as described by step 2 of $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\Pi, f, \lambda)$. After l executions of (B), the adversary $\mathcal{A}$ will provide an input $\hat{\alpha}$ on which he is willing to be challenged. If $\hat{\alpha} \neq \beta$, then the simulator $\mathcal{B}$ aborts; otherwise, it continues. Note that both β 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 $\mathcal{B}$ does not abort, it next runs $(\hat{\sigma}, \hat{\tau}) \leftarrow \mathsf{ProbGen}(sk, \hat{\alpha})$ and gives $\mathcal{A}$ an encoded input $\hat{\sigma}$. Then the adversary $\mathcal{A}$ may maliciously reply with $(\bar{\rho}, \bar{\pi})$ such that $\mathsf{Verify}(sk, \hat{\tau}, \bar{\rho}, \bar{\pi}) \triangleq \bar{y} \notin \{f(\hat{\alpha}), \perp\}$. On the other hand, an honest server in Π_{pe} will reply with $(\hat{\rho}, \hat{\pi})$. Due to Theorem 4, it must be the case that $\mathsf{Verify}(sk, \hat{\tau}, \hat{\rho}, \hat{\pi}) \triangleq \hat{y} = f(\hat{\alpha})$. Note that the event that $\bar{y} \notin \{f(\hat{\alpha}), \perp\}$ occurs with probability ϵ_1. Suppose the event $\bar{y} \notin \{f(\hat{\alpha}), \perp\}$ occurs, then the equation (2) is satisfied by both $(\bar{y}, \bar{\pi})$ and $(\hat{y}, \hat{\pi})$, i.e.,

$$e\Big(t/g_1^{\bar{y}}, g_{2k}^p\Big) = e\Big(g_1^{\hat{s}}/g_1^{\hat{\alpha}}, \bar{\pi}^p\Big) \text{ and } e\Big(t/g_1^{\hat{y}}, g_{2k}^p\Big) = e\Big(g_1^{\hat{s}}/g_1^{\hat{\alpha}}, \hat{\pi}^p\Big). \tag{3}$$

The equalities in (3) imply that $e\Big(g_1^{\bar{y}-\hat{y}}, g_{2k}^p\Big) = e\Big(g_1^{\hat{s}-\hat{\alpha}}, \big(\hat{\pi}/\bar{\pi}\big)^p\Big)$. Hence,

$$g_{2k+1}^{\frac{p}{\hat{s}-\hat{\alpha}}} = e\Big(g_1, \big(\hat{\pi}/\bar{\pi}\big)^p\Big)^{\frac{1}{\bar{y}-\hat{y}}}. \tag{4}$$

Note that the left hand side of (4) is $g_{2k+1}^{p/s}$ due to $\beta = \hat{\alpha}$. Therefore, (4) means that the simulator $\mathcal{B}$ can break the $(2k+1, n)$-MSDHS assumption (Definition 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, ϵ_1 must be negligible in λ, i.e., the scheme $\Pi_{\rm pe}$ is secure.

Privacy. The input privacy of $\Pi_{\rm pe}$ requires that no untrusted server can distinguish between different inputs of the client. This is formally defined by the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Pri}}(\Pi, f, \lambda)$ in Fig. 1. The client in our VC scheme encrypts its input α using BGN_{2k+1} which is semantically secure under SDA for Γ. As a result, our VC scheme achieves input privacy under SDA for Γ (see [26] for the proof of the following lemma).

Lemma 6. *If SDA holds for Γ, then the scheme $\Pi_{\rm pe}$ 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 BGN_{2k+1} in the execution of $\mathsf{ProbGen}$; it also decrypts one ciphertext $\rho = \mathsf{Enc}(f(\alpha))$ under BGN_{2k+1} and then verifies the equation (2). The overall computation of the client will be $O(\log n) = o(n)$ and therefore $\Pi_{\rm pe}$ 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 $\mathsf{Compute}$, which is comparable with the VC schemes based on FHE. Based on Lemmas 4, 5, 6 and the efficiency analysis, we have the following theorem.

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

3.2 Matrix Multiplication

In this section, we present our VC scheme $\Pi_{\rm mm}$ with input privacy (see Fig. 3) for matrix multiplication. In $\Pi_{\rm mm}$, 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 λ-bit prime not known to the server and $|\mathbb{D}| = \mathsf{poly}(\lambda)$. Our scheme uses a trilinear map instance Γ with groups of order $N = pq$, where p is also a λ-bit prime not known to the server. In $\Pi_{\rm mm}$, the client gives both M and its blinded version $T = (T_{ij})$ to the server, where T is computed using the $\mathsf{PRF}_{\rm dlin}$. It also sets up BGN_3. In order to verifiably compute

- $\mathsf{KeyGen}(1^\lambda, M)$: Pick a trilinear map instance $\varGamma = (N, G_1, G_2, G_3, e, g_1, g_2, g_3) \leftarrow \mathcal{G}(1^\lambda, 3)$. Consider the $\mathsf{PRF}_{\mathrm{dlin}}$ in Section 2.3. Run $\mathsf{KG}(1^\lambda, n)$ and pick a secret key K. Pick $a \leftarrow \mathbb{Z}_N$ and compute $T_{ij} = g_1^{p^2 a M_{ij}} \cdot \mathsf{F}_K(i,j)$ for every $(i,j) \in [n]^2$. Pick $u \leftarrow G_1$ and compute $h = u^q$. Set up BGN$_3$ with public key $(\varGamma, g_1, h)$ and secret key p. Output $sk = (p, q, K, a, \eta)$ and $pk = (\varGamma, g_1, h, M, T)$, where $\eta = g_3^{p^2 a}$.
- $\mathsf{ProbGen}(sk, x)$: 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 $i \in [n]$, compute $\tau_i = e(\prod_{j=1}^n \mathsf{F}_K(i,j)^{x_j}, g_2^p)$ using the efficient CFE algorithm in Section 2.3. Output $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\tau = (\tau_1, \ldots, \tau_n)$.
- $\mathsf{Compute}(pk, \sigma)$: Compute $\rho_i = \prod_{j=1}^n \sigma_j^{M_{ij}}$ 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 $\pi = (\pi_1, \ldots, \pi_n)$.
- $\mathsf{Verify}(sk, \tau, \rho, \pi)$: For every $i \in [n]$, compute y_i such that $\rho_i^p = (g_1^p)^{y_i}$. If
$$e(\pi_i, g_1^p) = \eta^{p y_i} \cdot \tau_i \tag{5}$$
for every $i \in [n]$, then output $y = (y_1, \ldots, y_n)$; otherwise output $\perp$.

Fig. 3. Matrix multiplication ($\varPi_{\mathrm{mm}}$)

Mx, the client stores $\tau = (\tau_1, \ldots, \tau_n)$, where each τ_i is efficiently computed using the closed form efficiency property of $\mathsf{PRF}_{\mathrm{dlin}}$. It gives $\sigma = (\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_n))$ to the server and the server returns $\rho = (\rho_1, \ldots, \rho_n) = \mathsf{Enc}(Mx)$ along with $\pi = (\pi_1, \ldots, \pi_n)$. At last, the client decrypts ρ_i to y_i and verify if (5) holds for every $i \in [n]$.

Correctness. The correctness of $\varPi_{\mathrm{mm}}$ 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 (see [26] for the proof).

Lemma 7. *If the server is honest, then $y = Mx$ and (5) holds for every $i \in [n]$.*

Security. The security of $\varPi_{\mathrm{mm}}$ requires that no untrusted server can cause the client to accept $\bar{y} \notin \{Mx, \perp\}$ with a forged proof. It is based on the 3-co-CDHS assumption for $\varGamma$ (Lemma 2) and the DLIN assumption (Definition 6).

Lemma 8. *If the 3-co-CDHS assumption for $\varGamma$ and the DLIN assumption both hold, then the scheme $\varPi_{\mathrm{mm}}$ is secure.*

Proof. We define three games $\mathsf{G}_0, \mathsf{G}_1$ and G_2 as below:

G_0 : this is the standard security game $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\varPi, M, \lambda)$ defined in Fig. 1.

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

G_2 : the only difference between this game and G_1 is that the matrix T is computed as $T_{ij} = g_1^{p^2 a M_{ij}} \cdot R_{ij}$, where $R_{ij} \leftarrow G_1$ for every $i, j \in [n]$.

For every $i \in \{0,1,2\}$, we denote by $\mathsf{G}_i(\mathcal{A})$ the output of game i when it is run with an adversary $\mathcal{A}$. 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 $\Pr[\mathsf{G}_0(\mathcal{A}) = 1] = \Pr[\mathsf{G}_1(\mathcal{A}) = 1]$.

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

Claim 2. We have that $|\Pr[\mathsf{G}_1(\mathcal{A}) = 1] - \Pr[\mathsf{G}_2(\mathcal{A}) = 1]| < \mathsf{neg}(\lambda)$.

The only difference between G_2 and G_1 is that we replace the pseudorandom group elements $\mathsf{F}_K(i,j)$ with truly random group elements $R_{ij} \leftarrow G_1$ for every $i,j \in [n]$. Clearly, if $|\Pr[\mathsf{G}_1(\mathcal{A}) = 1] - \Pr[\mathsf{G}_2(\mathcal{A}) = 1]|$ is non-negligible, we can construct an simulator $\mathcal{B}$ that simulates $\mathcal{A}$ and breaks the pseudorandom property of PRF with a non-negligible advantage.

Claim 3. We have that $\Pr[\mathsf{G}_2(\mathcal{A}) = 1] < \mathsf{neg}(\lambda)$.

Suppose that there is a PPT adversary $\mathcal{A}$ that wins with non-negligible probability ϵ in G_2. We want to construct a PPT simulator $\mathcal{B}$ that simulates $\mathcal{A}$ and breaks the 3-co-CDHS assumption (see Definition 5) with non-negligible probability. The adversary $\mathcal{B}$ takes as input a tuple $(p, q, \Gamma, h_1^\alpha, h_2^\beta)$, where $h_1 = g_1^p, h_2 = g_2^p$ and $\alpha, \beta \leftarrow \mathbb{Z}_N$. The adversary $\mathcal{B}$ is required to output $h_2^{\alpha\beta}$. In order to do so, $\mathcal{B}$ simulates $\mathcal{A}$ as below:

(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 $\eta = e(h_1^\alpha, h_2^\beta) = g_3^{p^2\alpha\beta}$;
- pick $u \leftarrow G_1$, compute $h = u^q$ and set up BGN_3 with public key (Γ, 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 $\mathcal{A}$, where $T = (T_{ij})$;

(B) Upon receiving a query $x = (x_1, \ldots, x_n)$ from $\mathcal{A}$, mimic $\mathsf{ProbGen}$ 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 $Z_{ij} = e(T_{ij}, g_2^{px_j})/\eta^{pM_{ij}x_j}$;
- for every $i \in [n]$, compute $\tau_i = \prod_{j=1}^n Z_{ij}$;
- send $\sigma = (\sigma_1, \ldots, \sigma_n)$ to $\mathcal{A}$.

It is straightforward to verify that the pk, σ and τ generated by $\mathcal{B}$ are identically distributed to those generated by the client in game G_2. We remark that (A) is the step 1 in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\Pi, M, \lambda)$ (see Fig. 1) and (B) consists of steps 3 and 4 in $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\Pi, M, \lambda)$. Furthermore, (B) may be run $l = \mathsf{poly}(\lambda)$ times as described by step 2 of $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Ver}}(\Pi, M, \lambda)$. After l executions of (B), the adversary $\mathcal{A}$ 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 $\mathcal{B}$ mimics $\mathsf{ProbGen}$ as (B) and gives $\mathcal{A}$ an encoded input $\hat{\sigma}$. Then the adversary $\mathcal{A}$ may maliciously reply with $\bar{\rho} = (\bar{\rho}_1, \ldots, \bar{\rho}_n)$ and $\bar{\pi} = (\bar{\pi}_1, \ldots, \bar{\pi}_n)$ such that $\mathsf{Verify}(sk, \hat{\tau}, \bar{\rho}, \bar{\pi}) \triangleq \bar{y} \notin \{M\hat{x}, \perp\}$. 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 7, it must be the case that $\mathsf{Verify}(sk, \hat{\tau}, \hat{\rho}, \hat{\pi}) \triangleq$

$\hat{y} = M\hat{x}$. Note that the event $\bar{y} \notin \{M\hat{x}, \perp\}$ occurs with probability ϵ. Suppose it occurs. Then there is an integer $i \in [n]$ such that $\bar{y}_i \neq \hat{y}_i$. Note that neither $\bar{y}$ nor $\hat{y}$ is $\perp$, the equation (5) must be satisfied by both $(\bar{y}, \bar{\pi})$ and $(\hat{y}, \hat{\pi})$, which translates into $e(\bar{\pi}_i, g_1^p) = \eta^{p\bar{y}_i} \cdot \hat{\tau}_i$ and $e(\hat{\pi}_i, g_1^p) = \eta^{p\hat{y}_i} \cdot \hat{\tau}_i$, we have that

$$e(\hat{\pi}_i/\bar{\pi}_i, g_1^p) = \eta^{p(\hat{y}_i - \bar{y}_i)} = e(g_2^{p^2\alpha\beta(\hat{y}_i - \bar{y}_i)}, g_1^p),$$

which in turn implies that $\hat{\pi}_i/\bar{\pi}_i = g_2^{p\alpha\beta \cdot p(\hat{y}_i - \bar{y}_i)}$. Let $\phi \in \mathbb{Z}_q^*$ be the multiplicative inverse of $p(\hat{y}_i - \bar{y}_i) \in \mathbb{Z}_q^*$. Then $g_2^{p\alpha\beta} = (\hat{\pi}_i/\bar{\pi}_i)^\phi$, i.e., $h_2^{\alpha\beta} = (\hat{\pi}_i/\bar{\pi}_i)^\phi$, which implies that $\mathcal{B}$ can break the 3-co-CDHS with probability at least ϵ. Therefore, this ϵ must be negligible in λ, i.e., $\Pr[\mathsf{G}_2(\mathcal{A}) = 1] < \mathsf{neg}(\lambda)$.

Privacy. The input privacy of $\Pi_{\rm mm}$ requires that no untrusted server can distinguish between different inputs of the client. This is formally defined by the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Pri}}(\Pi, f, \lambda)$ in Fig. 1. The client in our VC scheme encrypts its input x using BGN$_3$ which is semantically secure under SDA for Γ. As a result, $\Pi_{\rm mm}$ achieves input privacy under SDA for Γ (see [26] for the proof).

Lemma 9. *If the SDA for Γ holds, then $\Pi_{\rm mm}$ 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 BGN$_3$ and n verification keys $\tau_1, \ldots, \tau_n$ in the execution of $\mathsf{ProbGen}$; it also decrypts n ciphertext $\rho = \mathsf{Enc}(Mx)$ under BGN$_3$ and then verifies the equation (5). The overall computation of the client will be $O(n) = o(n^2)$ and therefore $\Pi_{\rm pe}$ 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 $\mathsf{Compute}$, which is comparable with the VC schemes based on FHE. Based on Lemmas 7, 8, 9 and the efficiency analysis, we have the following theorem.

Theorem 2. *If the 3-co-CDHS, DLIN and SDA assumptions for Γ all hold, then $\Pi_{\rm mm}$ is a VC scheme with input privacy.*

3.3 Discussions

A theoretical limitation of our VC schemes $\Pi_{\rm pe}$ and $\Pi_{\rm mm}$ is that the computation results (i.e., $f(\alpha)$ and Mx) must belong to a polynomial size domain $\mathbb{M}$ since otherwise the client will not be able to decrypt ρ 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 "$f(\alpha) \in \mathbb{M}$" (resp. M and the knowledge "$Mx \subseteq \mathbb{M}$"), one may argue that the cloud can also learn a polynomial size domain $\mathbb{D}$ where α (resp. Mx) is drawn from and therefore guess the actual value of α (resp. x) with non-negligible probability. However, recall that our privacy experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Pri}}(\Pi, f, \lambda)$ in Fig. 1 only requires the indistinguishability of different inputs. This is achieved by $\Pi_{\rm pe}$ and $\Pi_{\rm mm}$ (though for polynomial size domains) and suffices for our applications. Furthermore, in

Section 3.4, we shall show how to modify Π_{pe} and Π_{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 Π_{pe} and Π_{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 $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{Pri}}(\Pi, f, \lambda)$. Both Π_{pe} and Π_{mm} can be modified such that function privacy is also achieved. In the modified VC scheme Π'_{pe} (see Fig. 4 in Appendix A), the client gives BGN_{2k+2} ciphertexts $\mathsf{Enc}(f) = (\mathsf{Enc}(f_0), \ldots, \mathsf{Enc}(f_n))$ and $\sigma = (\mathsf{Enc}(\alpha), \ldots, \mathsf{Enc}(\alpha^{2^{k-1}}))$ to the server. Then the server can compute $\rho = \mathsf{Enc}(f(\alpha))$ along with a proof $\pi = \mathsf{Enc}(c(s))$ using $\mathsf{Enc}(f)$ and σ. In the modified VC scheme Π'_{mm} (see Fig. 5 in Appendix A), the client gives BGN_3 ciphertexts $\mathsf{Enc}(M) = (\mathsf{Enc}(M_{ij}))$ and $\sigma = (\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_n))$ to the server. Then the server can compute $\mathsf{Enc}(\sum_{j=1}^{n} M_{ij}x_j)$ along with a proof π_i using $\mathsf{Enc}(M)$ and σ for every $i \in [n]$. It is not hard to prove that the schemes Π'_{pe} and Π'_{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 severs 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 Π_{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 Π_{pe} with input i. In this solution, the communication cost consists of $O(\log n)$ group elements. Using Π_{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 Π_{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 $\{0, 1\} \subseteq \mathbb{M}$. 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.

Acknowledgement. This research was in part supported by Alberta Innovates Technology Future, Alberta, Canada.

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.-J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
7. Cachin, C., Micali, S., Stadler, M.A.: 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, vol. 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.: 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, Part II. LNCS, vol. 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)
24. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal Authenticated Data Structures with Multilinear Forms. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 246–264. Springer, Heidelberg (2010)
25. Parno, B., Raykova, M., Vaikuntanathan, V.: How to Delegate and Verify in Public: Verifiable Computation from Attribute-Based Encryption. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer, Heidelberg (2012)
26. Zhang, L.F., Safavi-Naini, R.: Private Outsourcing of Polynomial Evaluation and Matrix Multiplication using Multilinear Maps? (Full Version of this Paper), `http://arxiv.org/abs/1308.4218`

A Privacy Preserving VC Schemes

- $\mathsf{KeyGen}(1^\lambda, f(x))$: 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 $u = g_1^\delta$ for an integer $\delta \in \mathbb{Z}_N$. Set up BGN_{2k+2} with public key (Γ, 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 $pk = (\Gamma, g_1, h, g_1^s, g_1^{s^2}, \ldots, g_1^{s^{2^{k-1}}}, \gamma)$, where $\gamma = (\gamma_0, \ldots, \gamma_n)$.
- $\mathsf{ProbGen}(sk, \alpha)$: For every $\ell \in [k]$, pick $r_\ell \leftarrow \mathbb{Z}_N$ and compute $\sigma_\ell = g_1^{\alpha^{2^{\ell-1}}} h^{r_\ell}$. Output $\sigma = (\sigma_1, \ldots, \sigma_k)$ and $\tau = \perp$ (τ is not used).
- $\mathsf{Compute}(pk, \sigma)$: Compute $\rho_i = g_k^{\mu_i}$ for every $i \in \{0, 1, \ldots, n\}$ using the technique in Section 2.2. Compute $\rho'_i = e(\gamma_i, \rho_i) = g_{k+1}^{\mu'_i}$, where $\mu'_i = (f_i + q\delta v_i)\mu_i$. Compute $\rho = \prod_{i=0}^n \rho'_i$. Compute $\pi_{ij} = g_{2k}^{\nu_{ij}}$ using the technique in Section 2.2 for every $i \in \{0, 1, \ldots, n-1\}$ and $j \in \{0, 1, \ldots, i\}$. Compute $\pi'_{ij} = e(\gamma_{i+1}, \pi_{ij}) = g_{2k+1}^{\nu'_{ij}}$, 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 ρ and π.
- $\mathsf{Verify}(sk, \tau, \rho, \pi)$: Compute the $y \in \mathbb{Z}_q$ such that $\rho^p = (g_{k+1}^p)^y$. If the equality $e\big(t/g_1^y, g_{2k+1}^p\big) = e\big(g_1^s/g_1^\alpha, \pi^p\big)$ holds, output y; otherwise, output $\perp$.

Fig. 4. Univariate polynomial evaluation (Π'_{pe})

- $\mathsf{KeyGen}(1^\lambda, M)$: Pick $\Gamma = (N, G_1, G_2, G_3, e, g_1, g_2, g_3) \leftarrow \mathcal{G}(1^\lambda, 3)$. Consider the $\mathsf{PRF}_{\mathrm{dlin}}$ in Section 2.3. Run $\mathsf{KG}(1^\lambda, n)$ and pick a secret key K. Pick $a \leftarrow \mathbb{Z}_N$ and compute $T_{ij} = g_1^{p^2 a M_{ij}} \cdot \mathsf{F}_K(i,j)$ 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 (Γ, g_1, h) and secret key p. For every $(i,j) \in [n]^2$, pick $v_{ij} \leftarrow \mathbb{Z}_N$ and compute $\gamma_{ij} = g_1^{M_{ij}} h^{v_{ij}}$. Output $sk = (p, q, K, a, \eta)$ and $pk = (\Gamma, g_1, h, \gamma, T)$, where $\eta = g_3^{p^2 a}$ and $\gamma = (\gamma_{ij})$.
- $\mathsf{ProbGen}(sk, x)$: 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 $i \in [n]$, compute $\tau_i = e(\prod_{j=1}^n \mathsf{F}_K(i,j)^{x_j}, g_2^p)$ using the efficient CFE algorithm in Section 2.3. Output $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\tau = (\tau_1, \ldots, \tau_n)$.
- $\mathsf{Compute}(pk, \sigma)$: 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 $\pi = (\pi_1, \ldots, \pi_n)$.
- $\mathsf{Verify}(sk, \tau, \rho, \pi)$: For every $i \in [n]$, compute y_i such that $\rho_i^p = (g_2^p)^{y_i}$. If $e(\pi_i, g_1^p) = \eta^{p y_i} \cdot \tau_i$ for every $i \in [n]$, then output $y = (y_1, \ldots, y_n)$; otherwise, output $\perp$.

Fig. 5. Matrix multiplication (Π'_{mm})