Verifiable Computation of Large Polynomials

Jiaqi Hong[1,2], Haixia Xu[1](✉), and Peili Li[1,2]

[1] Institute of Informassurance and Communication Security Research Center, CAS, Beijing, China
[2] Graduate University of the Chinese Academy of Sciences, Beijing, China
{hongjiaqi,xuhaixia,lipeili}@iie.ac.cn

Abstract. Due to the proliferation of powerful cloud service, verifiable computation, which makes a computationally weak client perform intensive computations possible through outsourcing tasks to a powerful server, is attracting increasing attention. The correctness of the returned result should be verified as the server may be not trusted.
In this paper, we present a verifiable computation protocol on large polynomials, which can be publicly verified by any parties in the network. Compared with verifiable computation protocol presented by Backes et al., which is on quadratic, multi-variable polynomials, our verifiable computation protocol is on high degree, multi-variable polynomials and publicly verifiable.

Keywords: Verifiable computation · Amortized · Pre-computation · Public verification

1 Introduction

Verifiable computation makes it possible for personal computers to perform intensive computations through outsourcing computation tasks to a powerful cloud. In the age of cloud, resources are becoming more centralized. Individuals lacking computational capacity need only to buy the corresponding service in the cloud instead of purchasing their own expensive equipments to perform computation tasks. In this way, not only individual performs its computations cheap, but also resources in the cloud shared by many individuals are made full use of. So this kind of service model is attracting increasing attention.

In this paper, we name those who want to outsource intensive tasks clients, and those who have powerful resources servers. As the server may be not honest, the returned result should be verified by client to avoid malicious behavior from dishonest server. The cost of verification must be cheaper than the cost of preforming the computation, otherwise this outsourcing task will make no sense. In many instances, other parties in the network except for the client want to use this computation result. It will be better if the returned result can be publicly verified by all parties. For example, a doctor asks a server to perform a computation on the data of his patient, nurses need the computation result for better nursing. If the nurses have the ability to verify the result, they can get access to the correct result even if the doctor is not online. Many cases like this.

1.1 Related Work

Verifiable computation has a large body of prior works. There are two branches, one is on general functions, the other is on specific functions. Researches on general functions often used the method of knowledge proofs to verify the correctness of the returned result [10,11,17,18,25]. Until Gentry et al. constructed a fully homomorphic encryption over idea lattices [13], several verifiable computation protocols on general functions using the fully homomorphic encryption appeared [1,8,16,21]. A representative is Gennaro et al.' research work [16]. The authors combined fully homomorphic encryption with Yao's garbled circuit and used the range of the circuit to verify the result. Researches on specific functions utilized the special structure of the outsourcing function. So those researches often focused on polynomial and matrix computations [2,4,12,26]. Of course, there were some other researches on linear algebra [23] and exponential operations [19]. Our verifiable computation protocol is on large polynomials which have a large scale of variables and are in high degree. This kind of polynomials has an extensive use in important statistics. We will introduce two notions most relevant with our work in the following, one is amortized verifiable computation, the other is public verification.

Amortized Verifiable Computation. This notion was proposed by Gennaro et al. [16], it was widely used in many of the later works about verifiable computation [2,4,7,12,26]. The client performs a pre-computation for a specific function, then the returned result from server can be verified by client in a cheap cost. Although the pre-computation cost may be as expensive as the cost of performing the outsourcing function, this function can be performed several times on different inputs by server. After several computations, this expensive pre-computation cost can be amortized.

Benabbas et al. [4] followed this amortized notion and they proposed a novel method on verifiable polynomial computations. They utilized pseudorandom functions which have closed-form efficiency to generate a series of numbers as new polynomial coefficients according to specific polynomial structure. Clients use the reconstructed polynomial to verify the correctness of the returned result. As the reconstructed polynomial can be efficiently computed, this protocol is efficient on the amortized notion. The randomness of their pseudorandom functions are based on decisional Diffie-Hellman assumption.

Backes et al. [2] proposed another brand new method on verifiable quadratic polynomial computations. They combined homomorphic MAC with verifiable computation and this is also a representative of amortized verifiable computation. Clients preform a pre-computation on the multi-variable, quadratic polynomial first, then the returned result from the server can be verified by computing a quadratic polynomials on two variables. The verification is pretty efficient. Though their protocol is on quadratic polynomials, their work is irradiative. The security of their homomorphic MAC is based on decision linear assumption.

Public Verification. Recently, two works on verifiable computation can be publicly verified. Parno et al. [24] used the primitive of attribute-based encryption.

As we know, the ciphertext of an attribute-based encryption can be decrypted only if the attribute makes a function true. They used the result of an attribute under a one way function as public verification key. Any verifier performs this one-way function on the returned result to check the equality with the public verification key to verify the result. Their protocol was suitable for functions that can be expressed as poly-size Boolean Formulas. Another work is by Fiore et al. [12]. They followed the work of Benabbas et al. [4] and made the result publicly verifiable by combining it with a bilinear map. They used a pseudorandom function proposed by Lewko and Waters [22], and reduced the security of their verifiable computation protocol on co-computational Diffie-Hellman assumption.

In publicly verifiable computation protocols, the client performs an off-line pre-computation according to the function only, and then performs on-line pre-computations under inputs. The result of the on-line pre-computation should be public to allow a public verification. Expensive off-line pre-computation cost will be amortized to each on-line pre-computation if this function will be outsourced many time to server under different inputs.

1.2 Our Contribution

In this paper, we present a verifiable computation protocol on large polynomials. We call polynomials of a large scale of variables and in high degree large polynomials. This kind of polynomials has a significant use in statistics. We follow the idea of Backes et al. [2] and extend their protocol to a more generally applicable case. Their protocol is about verifiable computation on quadratic polynomials, while our protocol is on high degree polynomials and the result can be publicly verified. One challenge is that their protocol restricted in quadratic polynomial because their basic tool, homomorphic MAC, was constructed over a bilinear map. In a bilinear map setting, multiplication of exponents can be performed at most once. If we want to construct a verifiable computation protocol on high degree polynomials, the multilinear map is intuitive. Fortunately, Garg et al. [14] made a plausible lattices-based construction of multilinear map. Though this multilinear map is not efficient enough now, this cannot stop people from using it for new constructions [6,15,20,26]. This multilinear map makes sense in our verifiable computation protocol as the pre-computation is performed on integers first, and then encodes the result to a group element in multilinear map. Another challenge is that the randomness of their pseudorandom function used for constructing homomorphic MAC is based on decision linear assumption which will no longer hold in a multilinear map setting. So we construct a new pseudorandom function based on subgroup decisional assumption to build our verifiable computation protocol. What's more, this pseudorandom function has a better performance in reducing the pre-computation cost than the pseudorandom function used by Backes et al.. The last challenge is to realize public verification. We follow the idea of Fiore et al. [12] to make a publicly verifiable computation protocol. The security is based on co-computational Diffie-Hellman assumption.

Assume the outsourcing polynomial is of m variables and degree at most d in each monomial. The main features of our protocol are as follows:

– Our protocol is a publicly verifiable computation protocol on large polynomials.
– We follow the idea of amortized verifiable computation. The off-line pre-computation cost is $O((m+1)^d)$, same as the cost of performing the outsourcing polynomial computation. The on-line pre-computation cost is $O(d)$ in addition with a multilinear map operation. After several computations on different inputs, off-line pre-computation cost can be amortized.

2 Preliminaries

Notation. If S is a set, $x \overset{U}{\longleftarrow} S$ denotes uniformly choosing an element x from S. If $\mathcal{A}$ is an algorithm, $x \leftarrow \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x. Let $n \in \mathbb{N}$ be the security parameter, lastly we abbreviate *param* for public parameter, PPT for probabilistic polynomial time and PRF for pseudorandom function.

2.1 Multilinear Maps

One of our basic tool is the multilinear map. Garg et al. [14] made a plausible lattices based construction, then Coron et al. [9] made another construction over integers. Their multilinear map is a graded encoding system in fact, here we review an intuitive definition of it. The groups in this paper are all cyclic groups with order $N = pq$, where p, q are both n-bit primes.

Definition 1 (Multilinear Map). *Let $\overrightarrow{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_k)$ be a sequence of cyclic groups each of order N, and g_i be a canonical generator of $\mathbb{G}_i$. There exist a set of bilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j} | i, j \geq 1 \wedge i + j \leq k\}$, which satisfy the following operations:*

$$e_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab} : \forall a, b \in \mathbb{Z}_N.$$

when the context is obvious, we drop the subscripts i and j, such as, $e(g_i^a, g_j^b) = g_{i+j}^{ab}$.

Let $\mathcal{G}(1^n, k)$ denote a multilinear map generator with a security parameter n and a positive integer k which indicates the required encoding level as its inputs. The output of $\mathcal{G}(1^n, k)$ is a multilinear map $\Gamma_k = (N, \mathbb{G}_1, \ldots, \mathbb{G}_k, g_1, \ldots, g_k, e)$ as described before. In a multilinear map setting, multiplication of the exponents in high degree is possible without restriction of degree 2 as in a bilinear map setting.

2.2 Pseudorandom Function

Here, we review a definition of PRF. A PRF consists of two algorithm, KeyGen and $F_K(\cdot)$. Assume that the domain of the PRF is $\mathcal{X}$ and the range is $\mathcal{Y}$, KeyGen produces a secret key K while $F_K(\cdot)$ produces $y \in \mathcal{Y}$ according to K and an input $x \in \mathcal{X}$. A definition of PRF is as follows:

Definition 2 (PRF). *F is a pseudorandom function if for every* PPT *adversary $\mathcal{A}$, there exists a negligible function $neg(\cdot)$ such that for all n:*

$$|Pr[\mathcal{A}^{F_K(\cdot)}(1^n, param) = 1] - Pr[\mathcal{A}^{R(\cdot)}(1^n, param) = 1]| \leq neg(n)$$

where $R : \mathcal{X} \rightarrow \mathcal{Y}$ is a random function.

2.3 Computational Assumptions

Let $\Gamma_k = (N, \mathbb{G}_1, \ldots, \mathbb{G}_k, g_1, \ldots, g_k, e) \leftarrow \mathcal{G}(1^n, k)$ be a k-linear map. We review the (k, l)-Multilinear Diffie-Hellman Inversion assumption suggested by Sahai et al. [20]:

Definition 3 ((k, l)-MDHI). *Given Γ_k and $g_1, g_1^a, \ldots, g_1^{a^l} \in \mathbb{G}_1$, where $a \overset{U}{\longleftarrow} \mathbb{Z}_N$, the advantage of an adversary $\mathcal{A}$ in finding out $g_k^{a^{kl+1}}$ is*

$$\mathbf{Adv}_{\mathcal{A}}^{mdhi} = |Pr[\mathcal{A}(\Gamma_k, g_1^a, \ldots, g_1^{a^l}) = g_k^{a^{kl+1}}]|.$$

For any PPT *adversary $\mathcal{A}$, there exists a negligible function* neg$(\cdot)$ *such that for all n, $\mathbf{Adv}_{\mathcal{A}}^{mdhi}(n) \leq neg(n)$.*

The subgroup decisional assumption was first suggested by Boneh et al. [3]. Given $\mathbb{G}_i$ with order $N = pq$ and $u \overset{U}{\longleftarrow} \mathbb{G}_i$, it is hard to determine whether u belongs to subgroup $\mathbb{G}_i^q$ or not.

Definition 4 (SDA$_i$). *Given $\mathbb{G}_i$ and $u \overset{U}{\longleftarrow} \mathbb{G}_i$, the advantage of an adversary $\mathcal{A}$ in determining whether u belongs to subgroup $\mathbb{G}_i^q$ or not is*

$$\mathbf{Adv}_{\mathcal{A}}^{sda_i} = |Pr[\mathcal{A}(\mathbb{G}_i, u) = 1] - Pr[\mathcal{A}(\mathbb{G}_i, u^p) = 1]|.$$

For any PPT *adversary $\mathcal{A}$, there exists a negligible function $neg(\cdot)$ such that for all n, $\mathbf{Adv}_{\mathcal{A}}^{sda_i}(n) \leq neg(n)$.*

Zhang et al. [26] proved that subgroup decisional assumption holds for Γ_k if SDA_i holds for every $\mathbb{G}_i$, $i = 1, \ldots, k$.

The last one is the co-computational Diffie-Hellman assumption suggested by Boneh et al. [5].

Definition 5 (co-CDH Assumption). *Given Γ_k and g_1^a, g_2^b, where $a, b \overset{U}{\longleftarrow} \mathbb{Z}_N$, the advantage of an adversary $\mathcal{A}$ in finding out g_1^{ab} is*

$$\mathbf{Adv}_{\mathcal{A}}^{cdh} = Pr[\mathcal{A}(\Gamma_k, g_1^a, g_2^b) = g_1^{ab}].$$

For any PPT *adversary $\mathcal{A}$, there exists a negligible function* neg$(\cdot)$ *such that for all n, $\mathbf{Adv}_{\mathcal{A}}^{cdh}(n) \leq neg(n)$.*

2.4 Basic Model

Now we review a basic publicly verifiable computation model. The client performs an off-line pre-computation according to the outsourcing function only through the following **KeyGen** algorithm, and then performs an on-line pre-computation on specific inputs through the following **ProbGen** algorithm. The result of the on-line pre-computation should be public to allow a public verification. The server runs the **Compute** algorithm and returns a σ_y. Any third party can verify the returned computation result and output a value y or an error $\perp$.

Let $\mathcal{F}$ be a family of functions. A publicly verifiable computation protocol $\mathcal{VC}$ for $\mathcal{F}$ is as follows:

- **KeyGen**$(1^n, f) \rightarrow (SK, PK, EK)$. With a security parameter n and $f \in \mathcal{F}$, key generation algorithm produces secret key SK, public key PK, and evaluation key EK. Send EK to server. This is the off-line pre-computation on f.
- **ProbGen**$(PK, SK, x) \rightarrow (\sigma_x, VK_x)$. With an input x in the domain of f, the problem generation algorithm allows the client to produce an input encoding σ_x and a public verification key VK_x. This is the on-line pre-computation on specific input x.
- **Compute**$(PK, EK, f, \sigma_x) \rightarrow \sigma_y$. With PK, EK, f and σ_x, this algorithm allows server to perform a computation on f and return a σ_y to the verifier.
- **Verify**$(PK, VK_x, \sigma_y) \rightarrow y/\perp$. With PK, VK_x, and σ_y, this algorithm allows any party to verify the result and return a value y or an error $\perp$.

A verifiable computation protocol is secure if it holds the following properties: correctness and soundness. Simply, correctness is the value output by an honest server can be verified correctly.

Definition 6 (Correctness). *For any $f \in \mathcal{F}$, any $(SK, PK, EK) \leftarrow$ **KeyGen**$(1^n, f)$, any $x \in Dom(f)$, if $(\sigma_x, VK_x) \leftarrow$ **ProbGen**(PK, SK, x) and $\sigma_y \leftarrow$ **Compute**(PK, EK, f, σ_x), then the output of **Verify**(PK, VK_x, σ_y) is $f(x)$ with all but negligible probability.*

Soundness is any PPT adversary $\mathcal{A}$ cannot persuade a verifier to accept an incorrect computation result. Define the following experiment:

```
$\mathbf{Exp}_{\mathcal{A}}^{\mathrm{PubVer}}[\mathcal{VC}, f, l, n]$ :
$(SK, PK, EK) \leftarrow \mathbf{KeyGen}(1^n, f)$,
For $i = 1$ to $l$:
  $x_i \leftarrow \mathcal{A}(PK, EK, \sigma_{x,1}, VK_{x,1}, \ldots, \sigma_{x,i-1}, VK_{x,i-1})$,
  $(\sigma_{x,i}, VK_{x,i}) \leftarrow \mathbf{ProbGen}(PK, SK, x_i)$;
$x^* \leftarrow \mathcal{A}(PK, EK, \sigma_{x,1}, VK_{x,1}, \ldots, \sigma_{x,l}, VK_{x,l})$,
$(\sigma_{x^*}, VK_{x^*}) \leftarrow \mathbf{ProbGen}(PK, SK, x^*)$,
$\widehat{\sigma}_y \leftarrow \mathcal{A}(PK, EK, \sigma_{x,1}, VK_{x,1}, \ldots, \sigma_{x,l}, VK_{x,l}, \sigma_{x^*}, VK_{x^*})$,
$\widehat{y} \leftarrow \mathbf{Verify}(PK, VK_{x^*}, \widehat{\sigma}_y)$,
If $\widehat{y} \neq \perp$ and $\widehat{y} \neq f(x^*)$, output 1, else output 0.
```

For any $n \in \mathbb{N}$, any function $f \in \mathcal{F}$, the advantage of an adversary $\mathcal{A}$ making at most $l = poly(n)$ queries in the above experiment against $\mathcal{VC}$ is

$$\mathbf{Adv}_{\mathcal{A}}^{\text{PubVer}}(\mathcal{VC}, f, l, n) = Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{PubVer}}[\mathcal{VC}, f, l, n] = 1]$$

Definition 7 (Soundness). *A verifiable computation protocol $\mathcal{VC}$ is sound for $\mathcal{F}$, if for any $f \in \mathcal{F}$ and any PPT adversary $\mathcal{A}$ there exists a negligible function* neg$(\cdot)$ *such that for all n, $\mathbf{Adv}_{\mathcal{A}}^{\text{PubVer}}(\mathcal{VC}, f, l, n) \leq neg(n)$.*

3 Multi-labeled Program

The idea of our work is inspired by Backes et al.' multi-labeled verifiable computation protocol [2]. Briefly describing the conception of multi-labeled program and its corresponding verifiable computation protocol will help readers appreciate our work more easily.

In a multi-labeled program, a pair of labels $L = (\Delta, \tau)$ is used to identify a set of input message, where Δ is data set identifier and τ is input identifier. For an instance, if we want to record the weather condition per hour in a day, then we should keep track of temperature, humidity, sunlight and so on hourly. $\tau = (\tau_1, \tau_2, \cdots)$ labels temperature, humidity, sunlight etc. respectively, while Δ labels time. Regard the recordings in each hour as one data set. Different Δ_i labels different data sets, then τ can be reused to label inputs in different data sets. A pair $L = (\Delta, \tau)$ can uniquely identify a set of inputs while any single Δ or τ can not. Please refer to [2] for details.

The authors proposed a verifiable computation protocol on quadratic polynomials of m variables using multi-label. The verification cost is the cost of performing a quadratic polynomial on two variables. This verifiable computation protocol is efficient if m is large enough. We briefly review their protocols in the following:

Assume that the outsourcing function f is a quadratic polynomial of m variables. For every input x_i, $i = 1, \ldots, m$, client generates two pairs of pseudorandom values according to their labels such as: $(u_i, v_i) \leftarrow F_{K_1}(\tau_i)$, $(a, b) \leftarrow F_{K_2}(\Delta)$, where F is a PRF and K_1, K_2 are secret keys of F. Client chooses $\alpha \overset{U}{\longleftarrow} \mathbb{Z}_N$ as its secret key and sets $y_0^{(i)} = x_i, Y_1^{(i)} = (g^{u_i a + v_i b - x_i})^{\frac{1}{\alpha}}, Y_2^{(i)} = 1 \in \mathbb{G}_1$ for $i = 1, \ldots, m$, sends m tuples (y_0, Y_1, Y_2) to server. Server computes σ_y according to the arithmetic circuit of f gate by gate:

– **Addition**. If the gate is an addition gate, assume values on two input wires are respectively $y_0^{(1)}$ and $y_0^{(2)}$. Compute (y_0, Y_1, Y_2) as follows:

$$y_0 = y_0^{(1)} + y_0^{(2)}, Y_1 = Y_1^{(1)} \cdot Y_1^{(2)},$$

$$Y_2 = Y_2^{(1)} \cdot Y_2^{(2)}.$$

– **Multiplication**. If the gate is a multiplication gate, assume values on two input wires are respectively $y_0^{(1)}$ and $y_0^{(2)}$. Compute (y_0, Y_1, Y_2) as follows:

$$y_0 = y_0^{(1)} \cdot y_0^{(2)}, Y_1 = (Y_1^{(1)})^{y_0^{(2)}} \cdot (Y_1^{(2)})^{y_0^{(1)}},$$
$$Y_2 = e(Y_1^{(1)}, Y_1^{(2)}).$$

– **Mulplication with constant**. If the gate is a multiplication gate, the value of one input wire is a constant c, the value of another input wire is $y_0^{(1)}$. Compute (y_0, Y_1, Y_2) as follows:

$$y_0 = c \cdot y_0^{(1)}, Y_1 = (Y_1^{(1)})^c,$$
$$Y_2 = (Y_2^{(1)})^c.$$

After finishing the computation, server sets $\sigma_y = (y_0, Y_1, Y_2)$ and returns it to client. The verification equation is:

$$W = e(g, g)^{y_0} \cdot e(Y_1, g)^{\alpha} \cdot Y_2^{\alpha^2}, \tag{1}$$

where W is computed by client in two steps. Firstly, the client performs a pre-computation on the outsourcing quadratic polynomial f to obtain a quadratic polynomial on two variables:

$$\rho(z_1, z_2) = f(\rho_1(z_1, z_2), \ldots, \rho_m(z_1, z_2))$$

where $\rho_i(z_1, z_2) = u_i z_1 + v_i z_2$, $(u_i, v_i) \leftarrow F_{K_1}(\tau_i)$. Then, when the client wants to outsource this polynomial computation on specific inputs, it generates $(a, b) \leftarrow F_{K_2}(\Delta)$ according to data set label Δ and computes $W = \rho(a, b)$. If Eq. (1) holds, the returned σ_y is honestly computed and y_0 is the correct computation result. Otherwise, client outputs $\perp$. This polynomial f can be outsourcing many times on different inputs and the verification cost is the cost of performing a quadratic polynomial computation on two variables. The correctness and soundness of this protocol have been proved by Backes et al. [2].

This verifiable computation protocol can deal with polynomials in degree at most 2 as it is in the setting of bilinear map. If we just extend it to high degree polynomial using multilinear map, the verification cost will be a two variables polynomial of the same high degree. Unfortunately, the decision linear assumption which the protocol reduces the randomness of its PRF on no longer holds in a multilinear map setting. We construct a variant of the PRF which has a better performance in reducing the on-line pre-computation cost while realizing public verification.

4 Our Protocol

In this section, we present a publicly verifiable computation protocol on large polynomials. Assume that the outsourcing polynomial f is of m variables and degree at most d. We follow the idea of multi-labeled program and use a pair of labels $L = (\Delta, \tau_i)$ to identify input x_i, for all $i = 1, \ldots, m$. In the following, we will introduce our PRF first, then give a detailed verifiable computation protocol built on our PRF.

4.1 PRF with Amortized Closed-Form Efficiency

The randomness of our PRF is based on the subgroup decisional assumption.

PRF:

- KeyGen(1^n): Let $\varGamma_k = (N, \mathbb{G}_1, \ldots, \mathbb{G}_k, g_1, \ldots, g_k, e) \leftarrow \mathcal{G}(1^n, k)$. Choose two secret keys k_1, k_2 for PRFs $F'_{k_{1,2}} : \{0,1\}^n \rightarrow \mathbb{Z}_N$. Output $K = \{p, q, k_1, k_2\}$ and public parameter $param = \varGamma_k$.
- $F_K(x)$: On input x, generate a pair of values (a, b) according to its label $L = (\varDelta, \tau)$ such as: $a \leftarrow F'_{k_1}(\tau)$, and $b \leftarrow F'_{k_2}(\varDelta)$, where $\varDelta \in \{0,1\}^n$ and $\tau \in \{0,1\}^n$. Output $F_K(x) = g_1^{pab}$.

Theorem 1. *If F' is a pseudorandom function and the SDA assumption holds for $\varGamma_k$, then* PRF *is a pseudorandom function.*

Proof. The proof follows by a standard hybrid argument.

Game 0: this is the real game described above for PRF.
Game 1: this is Game 0 except that $F'_{k_1}(\tau)$ is replaced by a random function $\varPhi_1 : \{0,1\}^n \rightarrow \mathbb{Z}_N$. It is easy to argue that Game 1 is indistinguishable with Game 0.
Game 2: this is Game 1 except that $F'_{k_2}(\varDelta)$ is replaced by a random function $\varPhi_2 : \{0,1\}^n \rightarrow \mathbb{Z}_N$. Similarly to the previous case, one can easily argue that Game 2 is indistinguishable with Game 1.
Game$(\mathbf{3}, \mathbf{j})$: let $Q_\varDelta$ be the upper bound on the number of distinct $\varDelta$ queried by adversary $\mathcal{A}$. If $S = \{\varDelta_1, \ldots, \varDelta_{Q_\varDelta}\}$ is the ordered set of $\varDelta$ queried by $\mathcal{A}$, then, for $0 \leq j \leq Q_\varDelta$, we define the following partial sets of S: $S_{\leq j} = \{\varDelta_i \in S : i \leq j\}$ and $S_{>j} = \{\varDelta_i \in S : i > j\}$. Then we define Game $(3, j)$ same as Game 2 except that queries $(\varDelta, \tau)$ where $\varDelta \in S_{\leq j}$ are answered with a random value R chosen uniformly in $\mathbb{G}_1$, whereas queries $(\varDelta, \tau)$ where $\varDelta \in S_{>j}$ are answered with $R = g^{pab}$ where $a \leftarrow \varPhi_1(\tau)$ and $b \leftarrow \varPhi_2(\varDelta)$.

As one can notice, Game $(3, 0)$ is the same as Game 2, while Game $(3, Q_\varDelta)$ is the game where all queries are answered with freshly random values in $\mathbb{G}_1$, just like $\mathcal{A}$ is getting access to a truly random oracle from $\mathcal{X}$ to $\mathbb{G}_1$. If for every $1 \leq j \leq Q_\varDelta$, Game $(3, j-1)$ is computationally indistinguishable from Game $(3, j)$ under the subgroup decisional assumption holds for $\varGamma_k$, the proof can be done. So we prove the following lemma:

Lemma 1. *If subgroup decisional assumption holds for $\varGamma_k$, then $|Pr[G_{3,j-1}] - Pr[G_{3,j}]|$ is negligible for $1 \leq j \leq Q_\varDelta$.*

The key tool of our proof is the following lemma which shows the function $f_b(U) = U^{pb}$ is a weak PRF under the subgroup decisional assumption.

Lemma 2. *If the subgroup decisional assumption holds for $\varGamma_k$ then function $f_b(U) = U^{pb}$, where $b \overset{U}{\longleftarrow} \mathbb{Z}_N$, is a weak PRF.*

Proof. For a tuple (g_1, g_1^a, g_1^{pab}), we rename g_1^a as U and g_1^{pab} as V. Given such (U, V), challenger can create polynomially-many binary pairs (U_i, V_i) which have the same form, all V_i are random values in subgroup $\mathbb{G}_1^q$. If there exist a PPT adversary who can distinguish $f_b(U_i)$ with a random function, whose output is a random value in $\mathbb{G}_1$, in a non-negligible probability, then the challenger can solve subgroup decisional problem with the same probability.

Proof (Lemma 1). Now we show that any PPT adversary $\mathcal{A}$ who has non-negligible probability in distinguish Game $(3, j-1)$ with Game $(3, j)$ can build a PPT challenger $\mathcal{C}$ who distinguishes the weak PRF $f_b(U) = U^{pb}$ with a random function in the same probability.

$\mathcal{C}$ receives as input $param = \Gamma_k$ and gets access to an oracle which outputs a binary pair (U, V) on each query. Recall that if $\mathcal{O} = \mathcal{O}_f$, then $V = U^{pb}$ where b is the secret key of the weak PRF f. Otherwise, if $\mathcal{O} = \mathcal{O}_R$, then V is randomly chosen in $\mathbb{G}_1$. In both case, U is randomly chosen at every new query.

$\mathcal{C}$ runs the simulation for $\mathcal{A}$ as follows.

Assume that Q_τ is the upper bound on the number of distinct τ queried by $\mathcal{A}$. Let (Δ, τ) be query from $\mathcal{A}$, and assume that $(\Delta, \tau) = (\Delta_k, \tau_i)$ for $1 \leq k \leq Q_\Delta$ and $1 \leq i \leq Q_\tau$. $\mathcal{C}$ answers (Δ_k, τ_i) as follows.

- If $k \leq j-1$, then $\mathcal{C}$ chooses $R \overset{U}{\longleftarrow} \mathbb{G}_1$ uniformly and returns R.
- If $k > j$, then $\mathcal{C}$ chooses $b_k \overset{U}{\longleftarrow} \mathbb{Z}_N$ and queries the oracle $\mathcal{O}_f$. Return $R = f_{b_k}(U_i)$.
- If $k = j$, then $\mathcal{C}$ returns $R = V_i$

Basically, the simulator is implicitly setting $b_j = b$ where b is the secret key of the weak PRF f. Let $G_{3,j}$ be the event that Game $(3, j)$ outputs 1 which is run by adversary $\mathcal{A}$. Finally, $\mathcal{C}$ outputs the same bit b as $\mathcal{A}$ outputs b.

When $\mathcal{C}$ gets access to the weak PRF, where $V_i = f_b(U_i)$, then $\mathcal{C}$ is simulating Game $(3, j-1)$. On the other hand, when $\mathcal{C}$ gets access to a random function, where V_i is random and independent of U_i, then $\mathcal{C}$ simulates the view of Game $(3, j)$. That are $Pr[\mathcal{C}^{\mathcal{O}_f} = 1] = Pr[G_{3,j-1}]$ and $Pr[\mathcal{C}^{\mathcal{O}_R} = 1] = Pr[G_{3,j}]$. We have:

$$|Pr[\mathcal{C}^{\mathcal{O}_f} = 1] - Pr[\mathcal{C}^{\mathcal{O}_R} = 1]| = |Pr[G_{3,j-1}] - Pr[G_{3,j}]|$$

The simulation is perfect, and Lemma 1 has been proved.

The PRF helps to amortize the pre-computation cost. For a specific polynomial f, which is of m variables and in degree d, the client performs the pre-computation in two steps. In Step 1, the client transforms this polynomial to a one variable, degree d polynomial ρ in a cost $O((m+1)^d)$, the same as the cost of performing the computation on f. In Step 2, the client performs a computation on ρ with cost $O(d)$. Details as follows:

Step 1.

This is off-line pre-computation. Generate $a_i \leftarrow F'_{k_1}(\tau_i)$ according to input identifier τ_i for $i = 1, \ldots, m$, where $F'_{k_1}(\cdot)$ is the pseudorandom function to produce an exponent a. Set $\rho_i(z) = pa_i \cdot z$ for $i = 1, \ldots, m$. Obviously, all $\rho_i(z)$

are degree-1 polynomial on variable z with no constant. Perform the computation of f on $\rho_1(z), \ldots, \rho_m(z)$ to get a new one variable, degree d polynomial $\rho(z)$:

$$\rho(z) = f(\rho_1(z), \ldots, \rho_m(z)).$$

It is worth noting that the above computation can be done off-line by client as it is only related to function. The input identifier can be reused many times for a specific polynomial f as long as data set identifier is different. The cost of this step is $O((m+1)^d)$.

STEP 2.

This is on-line pre-computation. Generate $b \leftarrow F'_{k_2}(\Delta)$ according to data set identifier, where $F'_{k_2}(\cdot)$ is the pseudorandom function to produce an exponent b. Perform the computation of $\rho(z)$ on b, the result is $\rho(b)$ and computation cost is $O(d)$.

When performing STEP 2, the input of polynomial f has been identified. STEP 2 can be performed many times on different inputs for a specific polynomial f, the cost of off-line pre-computation can be amortized if this polynomial f will be performed many times on different inputs. So, the cost of pre-computation will be low on average.

4.2 Construction

Our verifiable computation protocol on large polynomials utilizes the PRF above. Let f be the outsourcing polynomial, assume it is a polynomial of m variables and in degree d. Details as follows:

- **KeyGen**$(1^n, k, f) \rightarrow (SK, PK, EK)$. This is key generation algorithm run by client. Generate a k-linear map, $\Gamma_k = (N, \mathbb{G}_1, \ldots, \mathbb{G}_k, g_1, \ldots, g_k, e) \leftarrow \mathcal{G}(1^n, k)$, where $k = d+2$. Choose $\alpha \xleftarrow{U} \mathbb{Z}_N$ uniformly. Choose secret keys of PRF as described before, $K = (k_1, k_2)$. Run STEP 1 to generate a one variable, degree d polynomial $\rho(z)$. Set $ek = (ek_0, ek_1, \ldots, ek_i, \ldots, ek_d)$ where $ek_i = g_{d-i+1}^{\alpha^i}$.
  The secret key $SK = (k_1, k_2, p, q, \alpha)$, the public key $PK = \Gamma_k$. The evaluation key $EK = ek$, send it to server.
- **ProbGen**$(SK, PK, x) \rightarrow (\sigma_x, VK_x)$. This is problem generation algorithm run by client. Run STEP 2 to get the result $\rho(b)$ and set the public verification key as $VK_x = g_{d+2}^{\rho(b)}$.
  Run PRF to get $R_i = g_1^{pa_i b}$ for each input x_i, $i = 1, \ldots, m$. Set $\sigma_i = (y_0^{(i)}, Y_1^{(i)}, Y_2^{(i)})$, where $y_0^{(i)} = x_i \in \mathbb{Z}_N$, $Y_1^{(i)} = (R_i \cdot g_1^{-x_i})^{\frac{1}{\alpha}} \in \mathbb{G}_1$, $Y_2^{(i)} = 1 \in \mathbb{G}_1$. Set $\sigma_x = (\sigma_1, \ldots, \sigma_m)$, send it to server.
- **Compute**$(PK, EK, f, \sigma_x) \rightarrow \sigma_y$. Given the evaluation key EK, σ_x, PK and the outsourcing polynomial f, server computes a σ_y as follows. For our convenience to describe, we interpret $f(x)$ as $f(x) = \sum_{i=1}^{s} f_i p_i(x)$, where for each monomial $f_i p_i(x)$ we interpret it further as $f_i p_i(x) = f_i \prod_{j=1}^{d} x_{i_j}$, where $0 \le i_1, \ldots, i_d \le m$, x_0 denotes constant 1, while $x_1, \ldots, x_m$ denote the m

variables. Server computes $\sigma_y = (y_0, Y_1, Y_2)$ according to each monomial first and then adds the s triples (y_0, Y_1, Y_2) together, details as follows:
Initiate $y_0 = 0$, $Y_1 = 1 \in \mathbb{G}_{d+1}$, $Y_2 = 1 \in \mathbb{G}_{d+1}$.

For $i = 1, \ldots, s$:

```
If $i_1 = \ldots = i_d = 0$, then:
    $y_{0i} = f_i, Y_{1i} = ek_0, Y_{2i} = ek_0$;
Else, let $\overline{j}$ be such that $i_{\overline{j}} \geq 1$ and $i_{\overline{j}+1} = \cdots = i_d = 0$:
    $Y_{2i} = e(Y_1^{(i_1)}, Y_1^{(i_2)}, \ldots, Y_1^{(i_{\overline{j}})}, ek_{\overline{j}})$,

    $Y_{1i} = e(Y_1^{(i_1)}, \ldots, Y_1^{(i_{\overline{j}-1})}, ek_{\overline{j}-1})^{y_0^{(i_{\overline{j}})}} \cdot e(Y_1^{(i_1)}, \ldots, Y_1^{(i_{\overline{j}-2})}, Y_1^{(i_{\overline{j}})}, ek_{\overline{j}-1})^{y_0^{(i_{\overline{j}-1})}}$
        $\cdots e(Y_1^{(i_1)}, Y_1^{(i_3)} \ldots, Y_1^{(i_{\overline{j}})}, ek_{\overline{j}-1})^{y_0^{(i_2)}} \cdot e(Y_1^{(i_2)}, \ldots, Y_1^{(i_{\overline{j}})}, ek_{\overline{j}-1})^{y_0^{(i_1)}}$
        $\cdot e(Y_1^{(i_1)}, \ldots, Y_1^{(i_{\overline{j}-2})}, ek_{\overline{j}-2})^{y_0^{(i_{\overline{j}-1})} \cdot y_0^{(i_{\overline{j}})}} \cdots e(Y_1^{(i_3)}, \ldots, Y_1^{(i_{\overline{j}})}, ek_{\overline{j}-2})^{y_0^{(i_1)} \cdot y_0^{(i_2)}}$
        $\cdots$
        $e(Y_1^{(i_1)}, ek_1)^{y_0^{(i_2)} \cdots y_0^{(i_{\overline{j}})}} \cdots e(Y_1^{(i_{\overline{j}})}, ek_1)^{y_0^{(i_1)} \cdots y_0^{(i_{\overline{j}-1})}}$,

    $y_{0i} = y_0^{(i_1)} \cdots y_0^{(i_{\overline{j}})}$,
    set $y_{0i} = f_i y_{0i}$, $Y_{1i} = (Y_{1i})^{f_i}$, and $Y_{2i} = (Y_{2i})^{f_i}$;
```

set $y_0 = y_0 + y_{0i}$, $Y_1 = Y_1 \cdot Y_{1i}$, and $Y_2 = Y_2 \cdot Y_{2i}$.
Server sets $\sigma_y = (y_0, Y_1, Y_2)$ and returns σ_y to verifier.

– **Verify**$(PK, VK_x, \sigma_y) \rightarrow y/\perp$. Any third party who wants to verify the result checks the following equation:

$$g_{d+2}^{y_0} \cdot e(Y_1, g_1) \cdot e(Y_2, g_1) = VK_x \tag{2}$$

If the equation holds, verifier outputs y_0 as the correct computation result. Otherwise, outputs an error symbol $\perp$.

First we show the correctness of the protocol briefly. Recall that $ek_i = g_{d-i+1}^{\alpha^i}$, if σ_y is honestly calculated by server, there is

$$g_{d+2}^{y_0} \cdot e(Y_1, g_1) \cdot e(Y_2, g_1) = g_{d+2}^{\rho(b)}, \tag{3}$$

Notice that $VK_x = g_{d+2}^{\rho(b)}$, then Eq. (2) holds. The honest result returned from server can be verified correctly.

Now we show the soundness of our protocol. If (k, l)-MDHI assumption holds in Γ_k, any PPT adversary can't get any secret keys from public key PK and evaluation key EK.

Theorem 2. *If co-CDH assumption holds in Γ_k, then any PPT adversary $\mathcal{A}$ making at most $l = poly(n)$ queries has advantage*

$$Adv_{\mathcal{A}}^{PubVer}(\mathcal{VC}, f, l, n) \leq neg(n),$$

where $neg(\cdot)$ is a negligible function.

Proof. The proof follows by a standard hybrid argument based on the following games:

Game 0: this is the real game same as $\mathbf{Exp}_{\mathcal{A}}^{PubVer}(\mathcal{VC}, f, l, n)$.

Game 1: this is Game 0 except for the following change in the evaluation of $\rho(b)$. For any x asked by the adversary during the game, instead of computing $\rho(b)$ using the Step 1 and Step 2, which is efficient in an amortized notion, an inefficient one step evaluation $\rho(b) = f(\rho_1(b), \dots, \rho_m(b))$ is used. One can easily argue that Game 1 is indistinguishable with Game 0.

Game 2: this is Game 1 except that PRF is replaced by a truly random function $R : \{0,1\}^n \times \{0,1\}^n \to \mathbb{G}_1$. Let R be a set of m random values generated by this random function where R is a set of m numbers. One can easily argue that Game 2 is indistinguishable with Game 1 as the randomness of our PRF.

Now we show if there exists a PPT adversary $\mathcal{A}$ who can win in Game 2 with a non-negligible probability, then there is a challenger $\mathcal{C}$ who can solve the *co*-CDH problem with the same probability.

$\mathcal{C}$ takes as input a group description Γ_k, chooses $r \xleftarrow{U} \mathbb{Z}_N$. For a query $x = (x_1, \dots, x_m)$ from $\mathcal{A}$, $\mathcal{C}$ chooses m random values $\beta_1, \dots, \beta_m \in \mathbb{Z}_N$, sets $R^{(i)} = g_1^{\beta_i}$, for $i = 1, \dots, m$. all $R^{(i)} = g_1^{\beta_i}$ are random values in $\mathbb{G}_1$. Set $\sigma_x = (\sigma_1, \dots, \sigma_m)$ where $\sigma_i = (y_0^{(i)}, Y_1^{(i)}, Y_2^{(i)})$, $y_0^{(i)} = x_i$, $Y_1^{(i)} = (R^{(i)} \cdot g_1^{-x_i})^{\frac{1}{r}}$, $Y_2^{(i)} = 1 \in \mathbb{G}_1$. Set $ek = (ek_0, ek_1, \dots, ek_i, \dots, ek_d)$ where $ek_i = g_{d-i+1}^{r^i}$. $\mathcal{C}$ computes $VK_x = g_{d+2}^{f(\beta_1, \dots, \beta_m)}$ and returns VK_x and σ_x to $\mathcal{A}$. The distribution of VK_x and σ_x are exactly the same as the one in Game 2.

Finally, let $\sigma_y^* = (y_0^*, Y_1^*, Y_2^*, W^*)$ be the output of $\mathcal{A}$ at the end of the game, such that for some x^* chosen by $\mathcal{A}$ it holds $\mathbf{Verify}(PK, VK_{x^*}, \sigma_{y^*}) = y^*$, $y^* \neq \perp$ and $y^* \neq f(x^*)$. By verification, this means that

$$g_{d+2}^{y_0^*} \cdot e(Y_1^* \cdot Y_2^*, g_1) = VK_x. \tag{4}$$

Let $\sigma_y = (y_0, Y_1, Y_2, W)$ be the correct output of the computation. Then, by correctness it also holds:

$$g_{d+2}^{y_0} \cdot e(Y_1 \cdot Y_2, g_1) = VK_x. \tag{5}$$

Dividing the verification Eq. (4) by (5),

$$g_{d+2}^{y_0^* - y_0} = e(Y_1 / Y_1^* \cdot Y_2 / Y_2^*, g_1). \tag{6}$$

That is, for a false y_0^*, $\mathcal{A}$ can find a Y_1^* and a Y_2^* to satisfy Eq. (6) in a non-negligible probability, then $\mathcal{B}$ solves the co-CDH problem with the same probability.

5 Conclusion

In this paper, we propose a delegated computation protocol on high degree polynomials over a large amount of variables which allows public verification. Assume that the delegated polynomial is of m variables and degree at most d. The off-line pre-computation cost is $O((m+1)^d)$, same as the cost of performing the outsourcing polynomial computation. The on-line pre-computation cost is $O(d)$ in addition with a multilinear map operation. Using the notion of amortization, off-line pre-computation cost can be amortized if the client delegates the same function f several times on different inputs. This protocol is efficient in average.

Acknowledgment. This work is supported by the National Natural Science Foundation of China (No.61379140) and the National Basic Research Program of China (973 Program) (No. 2013CB338001). The authors wish to acknowledge the anonymous referees for helpful suggestions.

References

1. 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)
2. Backes, M., Fiore, D., Reischuk., R. M.: Verifiable delegation of computation on outsourced data. In: CCS 2013, pp. 863–874. ACM press (2013). A full version is avaliable at http://eprint.iacr.org/2013/469 (2013)
3. 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)
4. 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)
5. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
6. Catalano, Dario, Fiore, Dario, Gennaro, Rosario, Nizzardo, Luca: Generalizing homomorphic MACs for arithmetic circuits. In: Krawczyk, Hugo (ed.) PKC 2014. LNCS, vol. 8383, pp. 538–555. Springer, Heidelberg (2014)
7. 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)
8. 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)
9. Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013)
10. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical Verified Computation with Streaming Interactive Proofs. In: ITCS 2012, pp. 90–112. ACM press, New York (2012)

11. Cormode, G., Thaler, J., Yi, K.: Verifying computations with streaming interactive proofs. Proc. VLDB Endowment **5**(1), 25–36 (2011)
12. Fiore, D., Gennaro, R.: Publicly Verification delegation of large polynomials and matrix computations, with applications. In: CCS 2012, pp. 501–512. ACM press, New York (2012)
13. Gentry, C.: A fully homomorphic encryption scheme. In: Stanford University (2009)
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. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In STOC 2008, pp. 113–122. ACM press, New York (2008)
18. Goldwasser, S., Lin, H., Rubinstein, A.: Delegation of computation without rejection problem from designated verifier cs-proofs. In: IACR Cryptology ePrint Archive, avaliable at http://eprint.iacr.org/2011/456 (2011)
19. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg (2005)
20. Hohenberger, S., Sahai, A., Waters, B.: Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 494–512. Springer, Heidelberg (2013)
21. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC 2012, pp. 1219–1234. ACM press (2012)
22. Lewko, A.B., Waters, B.: Efficient pseudorandom functions from the dicisional linear assumption and weaker variants. In: CCS 2009, pp. 112–120. ACM press, New York (2009)
23. Mohassel, P.: Efficient and secure delegation of linear algebra. In: IACR Cryptology ePrint Archive, avaliable at http://eprint.iacr.org/2011/605, (2011)
24. 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)
25. Rothblum, G.N., Vadhan, S., Wigderson, A.: Interactive proofs of proximity: delegating computation in sublinear time. In: STOC 2013, pp. 793–802. ACM press, New York (2013)
26. Zhang, L.F., Safavi-Naini, R.: Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 329–348. Springer, Heidelberg (2013)