Research Article

Liang Feng Zhang* and Reihaneh Safavi-Naini

Privacy-preserving verifiable delegation of polynomial and matrix functions

https://doi.org/10.1515/jmc-2018-0039
Received August 12, 2018; accepted August 27, 2019

Abstract: Outsourcing computation has gained significant popularity in recent years due to the development of cloud computing and mobile services. In a basic outsourcing model, a client delegates computation of a function f on an input $\mathbf{x}$ to a server. There are two main security requirements in this setting: guaranteeing the server performs the computation correctly, and protecting the client's input (and hence the function value) from the server. The verifiable computation model of Gennaro, Gentry and Parno achieves the above requirements, but the resulting schemes lack efficiency. This is due to the use of computationally expensive primitives such as fully homomorphic encryption (FHE) and garbled circuits, and the need to represent f as a Boolean circuit. Also, the security model does not allow verification queries, which implies the server cannot learn if the client accepts the computation result. This is a weak security model that does not match many real life scenarios. In this paper, we construct efficient (i.e., without using FHE, garbled circuits and Boolean circuit representations) verifiable computation schemes that provide privacy for the client's input, and prove their security in a strong model that allows verification queries. We first propose a transformation that provides input privacy for a number of existing schemes for verifiable delegation of multivariate polynomial f over a finite field. Our transformation is based on noisy encoding of $\mathbf{x}$ and keeps $\mathbf{x}$ semantically secure under the noisy curve reconstruction (CR) assumption. We then propose a construction for verifiable delegation of matrix-vector multiplication, where the delegated function f is a matrix and the input to the function is a vector. The scheme uses PRFs with amortized closed-form efficiency and achieves high efficiency. We outline applications of our results to outsourced two-party protocols.

Keywords: Cloud computing, outsourcing computation, verifiable computation, privacy preserving, polynomials, matrices

MSC 2010: 11T71, 94A60

1 Introduction

Outsourcing computation has gained significant popularity in recent years due to the development of cloud computing and mobile devices. Computationally weak devices such as smartphones and netbooks can outsource expensive computations to powerful cloud servers.

The first security concern that arises in outsourcing is to guarantee that the cloud server correctly performs the delegated computation. Cloud servers may have incentives, such as saving in computation time or other malicious goals, to produce results that may be incorrect. The *verifiable computation* (VC) of Gennaro, Gentry and Parno [18] allows a client to outsource the computation of a function f on an input $\mathbf{x}$ and then verify the correctness of the server's work. The outsourcing is especially meaningful as long as the client's work

***Corresponding author: Liang Feng Zhang,** School of Information Science and Technology, ShanghaiTech University, Shanghai, P. R. China, e-mail: zhanglf@shanghaitech.edu.cn
Reihaneh Safavi-Naini, Department of Computer Science, University of Calgary, Calgary, Canada, e-mail: rei@ucalgary.ca

spent on preparing $\mathbf{x}$ for delegation and verifying the server's results are substantially less than computing $f(\mathbf{x})$ locally. A second security concern is privacy of client's data, including the input $\mathbf{x}$ and the output $f(\mathbf{x})$.

Resolving both security issues simultaneously in an efficient way is a nontrivial problem. The proposals in [18] and several following works [2, 4, 13] address both security concerns but use expensive cryptographic primitives such as fully homomorphic encryption (FHE) and/or garbled circuits, and represent the function f as a Boolean circuit. The result is inefficient verifiable computation schemes. From a security view point, an important shortcoming is that these schemes can only tolerate adversaries that do not make verification queries, i.e., the adversary is not allowed to learn if the client has accepted the computation result.

1.1 Our work

In this paper, we develop the first verifiable computation schemes where the client's input is kept private from the server; both the client and the server computations are free of FHE, garbled circuits and Boolean circuit representations, and the security is proved in a strong model that allows verification queries. We achieve these properties for two types of functions: multivariate polynomials and functions that are represented by a matrix over a finite field.

A transformation for polynomial delegation schemes

Our first contribution is a transformation $\mathcal{T}$ that can be applied to a number of existing verifiable computation schemes, resulting in the input $\mathbf{x}$ and the output $f(\mathbf{x})$, to remain private (semantic security) from the server. Our transformation works for all schemes in [6, 10, 16, 37] where the function f is a multivariate polynomial over finite fields, does not use FHE, garbled circuits and Boolean circuits and allows verification queries if so does the underlying scheme.

Verifiable delegation of high-degree polynomial computations on private inputs is highly nontrivial. On one hand, the client has to provide a semantically secure encryption of $\mathbf{x}$ (say $\sigma_{\mathbf{x}}$) to the cloud server. On the other hand, the cloud server has to compute f on $\sigma_{\mathbf{x}}$ without knowing the decryption key and produce an encoding $\sigma_{\mathbf{y}}$ of the output $\mathbf{y} = f(\mathbf{x})$. A generic way to enable such computations is to use FHE, which however should be avoided in our schemes.

We resolve this difficulty using techniques from multivariate polynomial interpolation and reconstruction [12, 43]. Let $f(\mathbf{x}) = f(x_1, \ldots, x_h)$ be an h-variate polynomial of degree $\leq d$ over a finite field $\mathbb{F}$, and let $\mathbf{a} = (a_1, \ldots, a_h) \in \mathbb{F}^h$ be any input to the function. We observe that $f(\mathbf{a})$ can be learned from the restriction of f on a *random (parametric) curve* that passes through $\mathbf{a}$. More precisely, let $\gamma(z) = \mathbf{a} + \mathbf{r}_1 \cdot z + \cdots + \mathbf{r}_k \cdot z^k$ ($\mathbf{r}_1, \ldots, \mathbf{r}_k \in \mathbb{F}^h$) be a degree-$k$ parametric curve passing through $\mathbf{a}$, and let $g(z) = f(\gamma(z))$ be the restriction of f on the curve. Given any $t > kd$ points $\{(z_i, g(z_i))\}_{i=1}^t$, one can interpolate $g(z)$ and learn $f(\mathbf{a}) = g(0)$. The t points $\{(z_i, \gamma(z_i))\}_{i=1}^t$ can be regarded as a *noiseless encoding* of $\mathbf{a}$, where any $\leq k$ points perfectly hide $\mathbf{a}$. Distributing the t points to t different servers, one to each server, would enable each server to return a value $f(\gamma(z_i)) = g(z_i)$, and the t values jointly give $f(\mathbf{a})$ in a way such that any $\leq k$ servers learn no information about $\mathbf{a}$.

Unfortunately, we cannot use the noiseless encoding to attain input privacy in VC, where the single cloud server knows all t points $\{(z_i, g(z_i))\}_{i=1}^t$ and so is able to learn $\mathbf{a}$. To overcome this difficulty, we mix the t points of a noiseless encoding with $n - t$ random points $\{(z_j, \mathbf{u}_j)\}_{j=t+1}^n$ and form a *noisy encoding* of $\mathbf{a}$ that consists of n points (with locations randomly permuted). The values of f on these points suffice to compute $f(\mathbf{a})$. The problem of decoding $\mathbf{a}$ from its noisy encoding (known as noisy curve reconstruction) has been extensively studied in [7, 15, 23, 26, 40]. There is no known polynomial-time algorithm for the problem for $t \leq (nk^h)^{1/(h+1)} + k + 1$. The *noisy curve reconstruction (CR) assumption* [26] is that the noisy curve reconstruction problem is intractable when $t = o((nk^h)^{1/(h+1)} + k + 1)$.

While noisy encoding gives a "semantically secure encryption" of $\mathbf{a}$, it results in the long encoding of the input and so significant efficiency loss. Fortunately, the noisy encoding can be extended to "encrypt"

a polynomial number of inputs $\mathbf{a}_1, \ldots, \mathbf{a}_s$ at the same time, resulting in an encoding of size $O(n)$ elements for $s = O(n/t)$. With this extended noisy encoding, one can delegate computation of $f(\mathbf{a}_1), \ldots, f(\mathbf{a}_s)$ simultaneously and significantly reduce the *average cost* of delegation. This inspires our transformation $\mathcal{T}$ that extends polynomial delegation schemes to have input (and output) privacy: the cloud server computes f on $O(n)$ points of the noisy encoding of $\mathbf{a}_1, \ldots, \mathbf{a}_s$ and gives the results to the client; the client computes $f(\mathbf{a}_1), \ldots, f(\mathbf{a}_s)$ using polynomial interpolation. Using a verifiable polynomial delegation scheme Σ, the $O(n)$ computations of f will be verifiable by the client, and this determines if the server has worked correctly.

Applying $\mathcal{T}$ to the existing schemes [6, 10, 16, 37] (without data privacy), results in schemes with input (and output) privacy. Furthermore, $\mathcal{T}$ keeps additional properties, such as private/public verifiability of the underlying schemes, unchanged. For example, applying $\mathcal{T}$ to [37] results in a publicly verifiable scheme that allows efficient update of the function.

An input-private construction for matrix delegation

We interpret an $n \times n$ matrix $M = (M_{i,j})$ as a function that takes a vector $\mathbf{x} = (x_1, \ldots, x_n)$ as input and outputs $M \cdot \mathbf{x}'$, where $\mathbf{x}'$ is the transpose of $\mathbf{x}$. We propose an input-private verifiable computation of matrix-vector multiplications of the form $M \cdot \mathbf{x}'$. The scheme provides security assuming an adversary with access to verification queries and provides very efficient verification by avoiding the second level of amortization as used above.

The construction uses three primitives: a somewhat homomorphic encryption (SHE) adapted from [8], a homomorphic hash from [17] and a PRF with closed-form efficiency from [17]. The input (and output) privacy is obtained by the client encrypting $\mathbf{x}$ using the SHE and giving it to the server. The server has the matrix M and a tag matrix $T = (T_{i,j})$ for the matrix elements, each computed using the PRF with closed-form efficiency. The SHE scheme allows the server to perform homomorphic scalar multiplications and additions on the elements of M (in clear) and the ciphertext of $\mathbf{x}$, which gives an encrypted version of $M \cdot \mathbf{x}'$ for the client. The server is able to compute the homomorphic hash digests on the SHE ciphertexts of $\mathbf{x}$ and combines these digests with the tags of M to generate a proof of correctness for computation. The homomorphic property of the hash, and the amortized closed-form efficiency of the PRF, makes the client's verification significantly faster than the computation of $M \cdot \mathbf{x}'$ from scratch. In particular, the verification can be done in constant time after a one-time computation that is substantially more efficient than computing $M \cdot \mathbf{x}'$.

Application

Our verifiable computation schemes could be used to outsourcing of two-party protocols. We show an example of such applications to the outsourcing of private information retrieval (PIR). PIR [12] allows a client to retrieve any block f_i of a database $f = (f_1, f_2, \ldots, f_N)$ from a server such that $i \in [N]$ is not revealed to the server. Outsourced PIR [25, 34] has been suggested to offload the PIR server computation [5] to cloud. Both of our constructions give outsourced PIR with security against malicious cloud servers.

1.2 Related work

Securely outsourcing computation dates back to the work on interactive proofs [3, 22], PCP-based efficient arguments [29, 30], CS proofs [35] and the muggle proofs [21]. While these schemes are either interactive or in the random oracle model, the verifiable computation of Gennaro, Gentry and Parno [18] is non-interactive and in the standard model.

The verifiable computation schemes of [2, 4, 13, 18] attain input (and output) privacy and thus resolve both security issues simultaneously. However, they have to use the expensive cryptographic primitives such as FHE and/or garbled circuits and occasionally represent the function f as a Boolean circuit. As a result,

these schemes are not efficient enough both in terms of server computation and in terms of client computation. Furthermore, these schemes are only secure against adversaries that do not make verification queries. Goldwasser et al. [20] show how to construct reusable garbled circuits and obtain private schemes but again make use of FHE. Ananth et al. [1] constructed a verifiable computation scheme achieving input (and output) privacy using *multiple servers*, where FHE is not used but the security requires at least one of the servers is honest.

Fiore, Gennaro and Pastro [17] consider verifiable computation schemes where the data on cloud server (the function f in our setting) is kept private. In our schemes, the data on server is not necessarily encrypted, but the client's input $\mathbf{x}$ should be kept semantically secure in order to achieve input (and output) privacy. That is, we are studying a problem orthogonal to [17]. The schemes of [27, 32] consider the same problem as [17].

The verifiable computation schemes of [6, 9, 11, 14, 16, 21, 39] require the client to send its input to the cloud server *in clear* and thus attain no input (or output) privacy.

2 Preliminaries

Let λ be a security parameter. We denote by $\mathrm{poly}(\lambda)$ an arbitrary polynomial function in λ. We denote by $\mathrm{negl}(\lambda)$ an arbitrary negligible function in λ, i.e., any function $\epsilon(\lambda)$ from the natural numbers to the non-negative real numbers such that, for any $c > 0$, there is an integer $\lambda_c > 0$ such that $\epsilon(\lambda) < \lambda^{-c}$ for all $\lambda \geq \lambda_c$. Let $\mathcal{A}(\cdot)$ be any probabilistic polynomial-time (p.p.t.) algorithm. We denote by "$y \leftarrow \mathcal{A}(x)$" the procedure of running $\mathcal{A}$ on input x and assigning the output to y. Let Ω be any finite set. We denote by "$y \leftarrow \Omega$" the procedure of choosing an element y from Ω uniformly and at random. For every integer $m > 0$, we denote $[m] = \{1, 2, \ldots, m\}$.

2.1 Verifiable computation

A verifiable computation scheme [6, 18] is a two-party protocol between a client and a server. The client provides a function f and an input $\mathbf{x}$ to the server. The server is expected to compute $f(\mathbf{x})$ and respond with the (possibly encoded) output together with a proof that the output is correct. The client then verifies the output is indeed correct. The goal of verifiable computation is to make the client's verification as efficient as possible, and in particular much faster than the computation of $f(\mathbf{x})$ from scratch. In the amortized model of [6, 18], the client is allowed to do an expensive preprocessing on f to produce a key pair and then use the key pair to efficiently verify the server's computation of f on many different inputs. The scheme is said to be outsourceable if each individual verification is much faster than the corresponding computation.

A *verifiable computation* scheme $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ for an admissible function family $\mathcal{F}$ consists of four polynomial-time algorithms defined below.

- $(\mathrm{PK}_f, \mathrm{SK}_f) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$: Based on the security parameter λ, the randomized *key generation* algorithm generates a public key that encodes the target function f and the matching secret key. The public key is provided to the server, and the secret key is kept private by the client.
- $(\sigma_{\mathbf{x}}, \tau_{\mathbf{x}}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x})$: The *problem generation* algorithm uses the secret key SK_f to encode the function input $\mathbf{x}$ as a public value $\sigma_{\mathbf{x}}$ which is given to the server, and a secret value $\tau_{\mathbf{x}}$ which is kept private by the client.
- $\sigma_{\mathbf{y}} \leftarrow \mathsf{Compute}(\mathrm{PK}_f, \sigma_{\mathbf{x}})$: Using the client's public key and the encoded input, the server *computes* an encoded version (i.e., $\sigma_{\mathbf{y}}$) of the function's output $\mathbf{y} = f(\mathbf{x})$.
- $\{\mathbf{y}, \perp\} \leftarrow \mathsf{Verify}(\mathrm{SK}_f, \tau_{\mathbf{x}}, \sigma_{\mathbf{y}})$: Using the secret key SK_f and the secret "decoding" value $\tau_{\mathbf{x}}$, the *verification* algorithm converts the server's encoded output into the output of the function, e.g., $\mathbf{y} = f(\mathbf{x})$, or outputs $\perp$ indicating that $\sigma_{\mathbf{y}}$ does not represent the valid output of f on $\mathbf{x}$.

We are interested in verifiable computation schemes that are correct, secure, private and outsourceable. The scheme is said to be correct if the problem generation algorithm produces values that allow an honest server to compute values that will verify successfully and be converted to the evaluation of f on the client's input $\mathbf{x}$.

Definition 1 (Correctness). The scheme $\mathcal{VC}$ is *correct* if, for any function f from the admissible function family $\mathcal{F}$, the key generation algorithm produces keys $(\mathrm{PK}_f, \mathrm{SK}_f) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$ such that, for all $\mathbf{x} \in \mathrm{Domain}(f)$, if $(\sigma_\mathbf{x}, \tau_\mathbf{x}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x})$ and $\sigma_\mathbf{y} \leftarrow \mathsf{Compute}(\mathrm{PK}_f, \sigma_\mathbf{x})$, then $\mathsf{Verify}(\mathrm{SK}_f, \tau_\mathbf{x}, \sigma_\mathbf{y}) = f(\mathbf{x})$.

Intuitively, a verifiable computation scheme is secure if a malicious server cannot persuade the verification algorithm to accept an incorrect output. In other words, for a given function f and input $\mathbf{x}$, a malicious server should not be able to convince the verification algorithm to output a value $\hat{\mathbf{y}}$ such that $\hat{\mathbf{y}} \neq f(\mathbf{x})$. This intuition can be formalized by the following experiment.

- $(\mathrm{PK}_f, \mathrm{SK}_f) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$;
- for $\ell = 1, \ldots, L = \mathrm{poly}(\lambda)$:
  (a) $\mathbf{x}_\ell \leftarrow \mathcal{A}(\mathrm{PK}_f, \mathbf{x}_1, \sigma_{\mathbf{x}_1}, b_1, \ldots, \mathbf{x}_{\ell-1}, \sigma_{\mathbf{x}_{\ell-1}}, b_{\ell-1})$;
  (b) $(\sigma_{\mathbf{x}_\ell}, \tau_{\mathbf{x}_\ell}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x}_\ell)$;
  (c) $\hat{\sigma}_{\mathbf{y}_\ell} \leftarrow \mathcal{A}(\mathrm{PK}_f, \mathbf{x}_1, \sigma_{\mathbf{x}_1}, b_1, \ldots, \mathbf{x}_{\ell-1}, \sigma_{\mathbf{x}_{\ell-1}}, b_{\ell-1}, \mathbf{x}_\ell, \sigma_{\mathbf{x}_\ell})$;
  (d) $\hat{\mathbf{y}}_\ell \leftarrow \mathsf{Verify}(\mathrm{SK}_f, \tau_{\mathbf{x}_\ell}, \hat{\sigma}_{\mathbf{y}_\ell})$;
  (e) $b_\ell = 1$ if $\hat{\mathbf{y}}_\ell \neq \perp$; otherwise, $b_\ell = 0$;
- if there exists $\ell \in [L]$ such that $b_\ell = 1$ but $\hat{\mathbf{y}}_\ell \neq f(\mathbf{x}_\ell)$, output "1"; otherwise, output "0".

Figure 1: Experiment $\mathrm{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\mathcal{VC}, f, \lambda)$.

In the experiment $\mathsf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\mathcal{VC}, f, \lambda)$, the adversary $\mathcal{A}$ is given a polynomial number (i.e., L) of opportunities to persuade the verification algorithm to accept the wrong output value for an input value. In each trial, the adversary is given oracle access to generate the encoding of a problem instance, and also oracle access to the result of the verification algorithm on an arbitrary string on that instance. The adversary succeeds if it ever convinces the verification algorithm in a trial to accept the wrong output value for the input value. The security of $\mathcal{VC}$ requires that the adversary succeeds only with negligible probability.

Definition 2 (Security). The scheme $\mathcal{VC}$ is *secure* if, for any function $f \in \mathcal{F}$ and for any probabilistic polynomial-time adversary $\mathcal{A}$, there is a negligible function negl such that

$$\Pr[\mathsf{Exp}^{\mathsf{Ver}}_{\mathcal{A}}(\mathcal{VC}, f, \lambda) = 1] \leq \mathrm{negl}(\lambda).$$

Intuitively, a verifiable computation scheme is (input) private when the public outputs of the problem generation algorithm $\mathsf{ProbGen}$ for two different inputs are indistinguishable; i.e., nobody can decide which encoding is the correct one for a given input. The input privacy can be defined based on a typical indistinguishability argument and yields output privacy. Let $\mathsf{PubProbGen}(\mathrm{SK}_f, \cdot)$ be an oracle that computes $(\sigma_\mathbf{x}, \tau_\mathbf{x}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x})$ on any input $\mathbf{x}$ and returns only the public value $\sigma_\mathbf{x}$. We formalize the intuition (on input privacy) with the following experiment.

- $(\mathrm{PK}_f, \mathrm{SK}_f) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$;
- $(\mathbf{x}_0, \mathbf{x}_1) \leftarrow \mathcal{A}^{\mathsf{PubProbGen}(\mathrm{SK}_f, \cdot)}(\mathrm{PK}_f)$;
- $b \leftarrow \{0, 1\}$;
- $(\sigma_{\mathbf{x}_b}, \tau_{\mathbf{x}_b}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x}_b)$;
- $\hat{b} \leftarrow \mathcal{A}^{\mathsf{PubProbGen}(\mathrm{SK}_f, \cdot)}(\mathrm{PK}_f, \mathbf{x}_0, \mathbf{x}_1, \sigma_{\mathbf{x}_b})$;
- if $\hat{b} = b$, output "1", else "0".

Figure 2: Experiment $\mathrm{Exp}^{\mathsf{Pri}}_{\mathcal{A}}(\mathcal{VC}, f, \lambda)$.

Definition 3 (Privacy). The scheme $\mathcal{VC}$ is *private* if, for any function $f \in \mathcal{F}$ and for any probabilistic polynomial-time adversary $\mathcal{A}$, there is a negligible function "negl" such that

$$\left|\Pr[\mathsf{Exp}^{\mathsf{Pri}}_{\mathcal{A}}(\mathcal{VC}, f, \lambda) = 1] - \frac{1}{2}\right| \leq \mathrm{negl}(\lambda).$$

Informally, a verifiable computation scheme is outsourceable if the time to encode the input and verify the output must be smaller than the time to compute the function from scratch.

Definition 4 (Outsourceable). The scheme $\mathcal{VC}$ is *outsourceable* if it permits efficient problem generation and output verification. That is, for any $\mathbf{x}$ and any $\sigma_{\mathbf{y}}$, the time required for $\mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x})$ plus the time required for $\mathsf{Verify}(\mathrm{SK}_f, \sigma_{\mathbf{y}})$ is $o(T)$, where T is the time required to compute $f(\mathbf{x})$ from scratch.

We work in the *amortized* model of [6, 18], where the time required for $\mathsf{KeyGen}(1^\lambda, f)$ is not included in the above definition. In this model, computing the key pair $(\mathrm{PK}_f, \mathrm{SK}_f)$ is a one-time operation (per function) that can be amortized over the computation of on many (in fact, any $\mathrm{poly}(\lambda)$ number of) different inputs. Apart from this amortization, we also consider a *second level* of amortization occasionally, where a number of different inputs, say $\mathbf{x}_1, \ldots, \mathbf{x}_s$, are processed by $\mathsf{ProbGen}$ together and the delegation and verification of the computations $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_s)$ are done simultaneously.

3 Adding privacy to polynomial delegation

In this section, we show a transformation that can add (input and output) privacy to a verifiable computation scheme whose admissible function family consists of multivariate polynomials over a finite field. Our transformation is based on the noisy curve reconstruction assumption.

3.1 Noisy curve reconstruction assumption

The noisy curve reconstruction assumption generalizes the noisy polynomial reconstruction assumption [28, 36], which was widely used in protocol design [26, 42] and is based on the hardness of noisy polynomial list reconstruction problems.

Definition 5 (Noisy polynomial list reconstruction). Let $\mathbb{F}$ be a finite field, and let $n, k, t > 0$ be integers. Let $(z_1, y_1), \ldots, (z_n, y_n) \in \mathbb{F}^2$. The *noisy polynomial list reconstruction* problem with input $(n, k, t, \{(z_i, y_i)\}_{i=1}^n)$ is the problem of finding all polynomials $y(z)$ of degree $\leq k$ such that $y(z_i) = y_i$ for $\geq t$ values of $i \in [n]$.

When $t \geq \frac{n+k}{2}$, the noisy polynomial list reconstruction problem has a unique solution and can be solved in polynomial time by Berlekamp and Massey's algorithm [33]. Goldreich, Rubinfeld and Sudan [19] showed that, for $t > \sqrt{kn}$, the noisy polynomial list reconstruction problem has $\leq \mathrm{poly}(n)$ solutions. Sudan [41] and Guruswami and Sudan [24] proposed polynomial-time algorithms for $t \geq \sqrt{2kn}$ and $t \geq \sqrt{kn}$, respectively. For $t \leq \sqrt{kn}$, no polynomial-time algorithms are known. Naor and Pinkas [36] introduced the *noisy polynomial reconstruction assumption*, which asserts that, for appropriately chosen $n = n(\lambda)$, $k = k(\lambda)$, $t = t(\lambda)$ and $\mathbb{F} = \mathbb{F}(\lambda)$, the output distribution of the following procedure keeps $a \in \mathbb{F}$ semantically secure:

- randomly choose
  - a polynomial $y(z) \in \mathbb{F}[z]$ of degree $\leq k$ such that $y(0) = a$,
  - n nonzero field elements $z_1, z_2, \ldots, z_n \in \mathbb{F}$ such that they are distinct,
  - a subset $T \subseteq [n]$ of cardinality t and set $y_i = y(z_i)$ for every $i \in T$,
  - a field element $y_i \in \mathbb{F}$ for every $i \in [n] \setminus T$;
- output $\{(z_i, y_i)\}_{i=1}^n$.

Ishai, Kushilevitz, Ostrovsky and Sahai [26] considered a multi-dimensional variant of the noisy polynomial list reconstruction problem and introduced the *noisy curve reconstruction (CR) assumption*.

Definition 6 (CR assumption). Let k be a degree parameter, which will also serve as a security parameter. Given functions $\mathbb{F}(k)$ (field), $h(k)$ (dimension), $t(k)$ (the number of points on the curve) and $n(k)$ (the total number of points), the *CR assumption* holds with parameters $(\mathbb{F}, h, t, n)$ if the output distribution $\mathcal{D}^{\mathbf{a}}_{n,k,t,h}$ of the following procedure keeps $\mathbf{a} = (a_1, \ldots, a_h) \in \mathbb{F}(k)^{h(k)}$ semantically secure:

- randomly choose
  - h polynomials $\gamma_1(z), \ldots, \gamma_h(z) \in \mathbb{F}[z]$ of degree $\le k$ such that $(\gamma_1(0), \ldots, \gamma_h(0)) = (a_1, \ldots, a_h)$,
  - n nonzero field elements $z_1, z_2, \ldots, z_n \in \mathbb{F} \setminus \{0\}$ such that they are distinct,
  - a subset $T \subseteq [n]$ of cardinality t, and for every $i \in T$, set $\mathbf{y}_i = (\gamma_1(z_i), \ldots, \gamma_h(z_i))$,
  - a vector $\mathbf{y}_i \in \mathbb{F}^h$ for every $i \in [n] \setminus T$;
- output $\{\mathbf{y}_i\}_{i=1}^n$.

Formally, the *CR assumption* holds if, for any points $\mathbf{a}_0, \mathbf{a}_1 \in \mathbb{F}(k)^{h(k)}$, for any probabilistic polynomial-time algorithm $\mathcal{A}$, there is a negligible function "negl" such that

$$|\Pr[\mathcal{A}(\mathcal{D}^{\mathbf{a}_0}_{n,k,t,h}) = 1] - \Pr[\mathcal{A}(\mathcal{D}^{\mathbf{a}_1}_{n,k,t,h}) = 1]| \le \mathrm{negl}(k).$$

An augmented version of the CR problem requires one to learn a from $\{(z_i, \mathbf{y}_i)\}_{i=1}^n$ (instead of $\{\mathbf{y}_i\}_{i=1}^n$) and was resolved in [15] when $t > (nk^h)^{1/(h+1)} + k + 1$. The problem remains hard when $t = o((nk^h)^{1/(h+1)})$, and the CR assumption remains plausible despite of the progress in list decoding [7, 15, 23, 26, 40].

3.2 Multivariate polynomial interpolation and noisy encoding

Multivariate polynomial interpolation allows one to learn the value of a multivariate polynomial at a point, given its restriction on a parametric curve passing through that point. Let $h, d > 0$ be integers. For any vector $\mathbf{i} = (i_1, \ldots, i_h)$ of non-negative integers, we denote $\mathrm{wt}(\mathbf{i}) = i_1 + \cdots + i_h$ as the *weight* of $\mathbf{i}$. Let $\mathbb{F}$ be any finite field. We denote by $\mathbb{F}[\mathbf{x}] = \mathbb{F}[x_1, \ldots, x_h]$ the ring of all polynomials in the h variables $\mathbf{x} = (x_1, \ldots, x_h)$ and denote by $\mathbf{x}^{\mathbf{i}} = x_1^{i_1} \cdots x_h^{i_h}$ the monomial of *multidegree* $\mathbf{i}$ (and *degree* $\mathrm{wt}(\mathbf{i})$) in $\mathbf{x}$. Let $f(\mathbf{x}) = \sum_{\mathbf{i}:\mathrm{wt}(\mathbf{i}) \le d} f_{\mathbf{i}} \cdot \mathbf{x}^{\mathbf{i}}$ be any h-variate polynomial of degree $\le d$ over $\mathbb{F}$, and let $\mathbf{a} = (a_1, \ldots, a_h) \in \mathbb{F}^h$. The multivariate polynomial interpolation technique of learning $f(\mathbf{a})$ can be described as the following procedure:

- choose $\mathbf{r}_1, \ldots, \mathbf{r}_k \leftarrow \mathbb{F}^h$; define a parametric curve $\gamma(z) = \mathbf{a} + \mathbf{r}_1 \cdot z + \cdots + \mathbf{r}_k \cdot z^k$;
- learn $f(\gamma(z_i))$ for $t \ge kd + 1$ distinct nonzero field elements $z_1, \ldots, z_t \in \mathbb{F} \setminus \{0\}$;
- interpolate the polynomial $g(z) = f(\gamma(z))$ of degree $\le kd$ with $\{(z_i, f(\gamma(z_i)))\}_{i=1}^t$;
- output $g(0)$ (which is equal to $f(\gamma(0)) = f(\mathbf{a})$).

This procedure allows one to hide $\mathbf{a}$ from a subset of the players in distributed protocols for evaluating a multivariate polynomial $f(\mathbf{x})$, such as in the *private information retrieval* (PIR) protocols [12, 43], where a client gives t points $\gamma(z_1), \ldots, \gamma(z_t)$ to t servers such that no k or less servers can learn any information about $\mathbf{a}$, the i-th server returns $g(z_i)$ and the client recovers $f(\mathbf{a})$ from the t values $g(z_1), \ldots, g(z_t)$. We consider the t points $\gamma(z_1), \ldots, \gamma(z_t)$ as a *noiseless encoding* of $\mathbf{a}$, which leaks absolutely no information about $\mathbf{a}$ to any adversary that observes $\le k$ of the t points.

We shall construct verifiable computation schemes where the client's input $\mathbf{a}$ is kept private from a single cloud server. While sending a noiseless encoding $(\gamma(z_1), \ldots, \gamma(z_t))$ of $\mathbf{a}$ to the server simply reveals $\mathbf{a}$ to that server, the CR assumption allows us to develop a *noisy encoding* $\{\mathbf{y}_i\}_{i=1}^n$ of $\mathbf{a}$ (as in the procedure of Definition 6) that keeps $\mathbf{a}$ semantically secure. Unfortunately, we cannot directly use this noisy encoding in the constructions due to efficiency loss. On one hand, the CR assumption requires that $t \le (nk^h)^{1/(h+1)} + k + 1$. On the other hand, one has to choose $t \ge kd + 1$ to enable the interpolation of $g(z) = f(\gamma(z))$. As a result, $n \ge (d-1)^{h+1} \cdot k$ and is comparable to $\binom{h+d}{d}$, the number coefficients of f. And the noisy encoding only yields a scheme that is not outsourceable.

We bypass this difficulty with a second level of *amortization*, i.e., by processing multiple function inputs $\mathbf{a}_1, \ldots, \mathbf{a}_s$ together such that the average encoding length of each input is short and thus results in outsourceable schemes. In [26], it was shown that if $n - t$ noisy points suffice to keep one point semantically secure, then, for any $s = \mathrm{poly}(k)$, they suffice to keep s points semantically secure. With this observation, we describe an *extended* noisy encoding algorithm $(\mathsf{pk}_{\vec{\mathbf{a}}}, \mathsf{rk}_{\vec{\mathbf{a}}}) \leftarrow \mathsf{NEnc}(k, \vec{\mathbf{a}})$ that takes $\vec{\mathbf{a}} = (\mathbf{a}_1, \ldots, \mathbf{a}_s) \in (\mathbb{F}^h)^s$ as input and outputs a public noisy encoding $\mathsf{pk}_{\vec{\mathbf{a}}}$ and a private value $\mathsf{rk}_{\vec{\mathbf{a}}}$ for reconstruction use as follows.

- for every $\ell \in [s]$, randomly choose h polynomials $\gamma_{\ell,1}(z), \ldots, \gamma_{\ell,s}(z) \in \mathbb{F}[z]$ of degree $\leq k$ such that $\mathbf{a}_\ell = (\gamma_{\ell,1}(0), \ldots, \gamma_{\ell,h}(0))$;
- randomly choose $m = ts + n - t$ nonzero distinct field elements $z_1, \ldots, z_m \in \mathbb{F} \setminus \{0\}$;
- randomly choose s pairwise disjoint subsets $T_1, \ldots, T_s \subset [m]$, each of cardinality t;
- for every $\ell \in [s]$ and $j \in T_\ell$, set $\mathbf{c}_j = (\gamma_{\ell,1}(z_j), \ldots, \gamma_{\ell,h}(z_j))$;
- set $T_0 = [m] \setminus (T_1 \cup T_2 \cup \cdots \cup T_s)$; for every $j \in T_0$, randomly choose a vector $\mathbf{c}_j \in \mathbb{F}^h$;
- output $\mathsf{pk}_{\vec{\mathbf{a}}} = \{\mathbf{c}_j\}_{j=1}^m$ and $\mathsf{rk}_{\vec{\mathbf{a}}} = \{T_i\}_{i=0}^s$.

3.3 The transformation

The algorithm NEnc allows one to hide s function inputs, say $\vec{\mathbf{a}} = (\mathbf{a}_1, \ldots, \mathbf{a}_s)$, with a public noisy encoding $\mathsf{pk}_{\vec{\mathbf{a}}}$ such that no information about $\vec{\mathbf{a}}$ will be leaked (under the CR assumption). Let Σ be a non-private verifiable computation scheme [6, 10, 16, 37] with an admissible function family of multivariate polynomials over a finite field. We shall present a transformation $\mathcal{T}$ that adds (input and output) privacy to Σ. The idea of our transformation is letting the client encode $\vec{\mathbf{a}}$ as $\mathsf{pk}_{\vec{\mathbf{a}}}$ and give $\mathsf{pk}_{\vec{\mathbf{a}}}$ to the server; the server runs Σ.Compute on every element (which is a point) of $\mathsf{pk}_{\vec{\mathbf{a}}}$ and provides the public values of evaluating the polynomial f on all points to the client; at last the client runs Σ.Verify to both verify the server's work and recover the results $f(\mathbf{a}_1), \ldots, f(\mathbf{a}_s)$. This idea gives a new scheme $\Pi = \mathcal{T}(\Sigma)$ as below.

- $(\mathrm{PK}_f, \mathrm{SK}_f) \leftarrow \Pi.\mathsf{KeyGen}(1^k, f)$: Given $f = f(\mathbf{x}) \in \mathbb{F}[x_1, \ldots, x_h]$, an h-variate polynomial of degree $\leq d$, run $\Sigma.\mathsf{KeyGen}(1^k, f)$ to generate a public key pk_f and the matching secret key sk_f; output $\mathrm{PK}_f = \mathrm{pk}_f$ and $\mathrm{SK}_f = \mathrm{sk}_f$.
- $(\sigma_{\vec{\mathbf{a}}}, \tau_{\vec{\mathbf{a}}}) \leftarrow \Pi.\mathsf{ProbGen}(\mathrm{SK}_f, \vec{\mathbf{a}})$: Given $\vec{\mathbf{a}} = (\mathbf{a}_1, \ldots, \mathbf{a}_s) \in (\mathbb{F}^h)^s$, a set of s inputs from Domain(f), run $\mathsf{NEnc}(k, \vec{\mathbf{a}})$ to generate both a public noisy encoding $\mathsf{pk}_{\vec{\mathbf{a}}}$ of $\vec{\mathbf{a}}$ and a private value $\mathsf{rk}_{\vec{\mathbf{a}}}$ for the reconstruction use. Parse $\mathsf{pk}_{\vec{\mathbf{a}}}$ as $\{\mathbf{c}_j\}_{j=1}^m \subseteq \mathbb{F}^h$, a set of m points. For every $j \in [m]$, run $\Sigma.\mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{c}_j)$ to generate both a public encoding $\sigma_{\mathbf{c}_j}$ of $\mathbf{c}_j$ and a private value $\tau_{\mathbf{c}_j}$ for verification use. At last, output $\sigma_{\vec{\mathbf{a}}} = \{\sigma_{\mathbf{c}_j}\}_{j=1}^m$ as the public encoding of $\vec{\mathbf{a}}$, and output $\tau_{\vec{\mathbf{a}}} = (\mathsf{rk}_{\vec{\mathbf{a}}}, \{\tau_{\mathbf{c}_j}\}_{j=1}^m)$, the private values for verification and reconstruction.
- $\sigma_{\mathbf{y}} \leftarrow \Pi.\mathsf{Compute}(\mathrm{PK}_f, \sigma_{\vec{\mathbf{a}}})$: Parse $\sigma_{\vec{\mathbf{a}}}$ as $\{\sigma_{\mathbf{c}_j}\}_{j=1}^m$, the set of s public encodings, one for each element in $\mathsf{pk}_{\vec{\mathbf{a}}}$. For every $j \in [m]$, run $\Sigma.\mathsf{Compute}(\mathrm{PK}_f, \sigma_{\mathbf{c}_j})$ to compute an encoded version $\sigma_{f(\mathbf{c}_j)}$ of the function's output $f(\mathbf{c}_j)$. At last, output $\sigma_{\mathbf{y}} = \{\sigma_{f(\mathbf{c}_j)}\}_{j=1}^m$ as the encoded version of the function's outputs on all s inputs, i.e., $f(\mathbf{a}_1), \ldots, f(\mathbf{a}_s)$.
- $\{\mathbf{y}, \perp\} \leftarrow \Pi.\mathsf{Verify}(\mathrm{SK}_f, \tau_{\vec{\mathbf{a}}}, \sigma_{\mathbf{y}})$: Parse $\tau_{\vec{\mathbf{a}}}$ as $(\mathsf{rk}_{\vec{\mathbf{a}}}, \{\tau_{\mathbf{c}_j}\}_{j=1}^m)$, where $\{\tau_{\mathbf{c}_j}\}_{j=1}^m$ is for verification use and $\mathsf{rk}_{\vec{\mathbf{a}}}$ is for reconstruction use. Parse $\sigma_{\mathbf{y}}$ as $\{\sigma_{f(\mathbf{c}_j)}\}_{j=1}^m$, an encoded version of the s function outputs $f(\mathbf{a}_1), \ldots, f(\mathbf{a}_s)$. For every $j \in [m]$, run $\Sigma.\mathsf{Verify}(\mathrm{SK}_f, \tau_{\mathbf{c}_j}, \sigma_{f(\mathbf{c}_j)})$ to verify the server's work of computing $f(\mathbf{c}_j)$ and output v_j, where $v_j = f(\mathbf{c}_j)$ or $v_j = \perp$ (indicating that $\sigma_{f(\mathbf{c}_j)}$ is not a valid encoding of $f(\mathbf{c}_j)$). If there exists $j \in [m]$ such that $v_j = \perp$, then output $\perp$ to indicate that $\sigma_{\mathbf{y}}$ is not a valid encoding of the s function outputs. Otherwise, parse $\mathsf{rk}_{\vec{\mathbf{a}}}$ as $(T_0, T_1, \ldots, T_s)$, where $T_1, \ldots, T_s \subseteq [m]$ are pairwise disjoint t-subsets and $T_0 = [m] \setminus T_1 \cup \cdots \cup T_s$; for every $\ell \in [s]$, interpolate a polynomial $Q_\ell(z) = f(\gamma_{\ell,1}(z), \ldots, \gamma_{\ell,h}(z))$ of degree $\leq t$ from the t points $\{(z_j, v_j)\}_{j \in T_\ell}$. At last, output $\mathbf{y} = (Q_1(0), \ldots, Q_s(0))$.

Correctness: The correctness of Σ implies that $v_j = f(\mathbf{c}_j)$ for every $j \in [m]$. For every $\ell \in [s]$, the points $\{\mathbf{c}_j\}_{j \in T_\ell}$ are on the parametric curve $\gamma_\ell(z) = (\gamma_{\ell,1}(z), \ldots, \gamma_{\ell,h}(z))$. Then $\{v_j\}_{j \in T_\ell}$ are values of $Q_\ell(z) = f(\gamma_\ell(z))$ at $t = |T_\ell|$ distinct points $\{z_j\}_{j \in T_\ell}$, where $\deg(Q_\ell(z)) \leq k \cdot \deg(f) \leq kd$. If the parameters t, k, d are chosen such that $t \geq kd + 1$, then the t points $\{(z_j, v_j)\}_{j \in T_\ell}$ suffice to interpolate the univariate polynomial $Q_\ell(z)$ of degree $\leq kd$ and give

$$Q_\ell(0) = f(\gamma_\ell(0)) = f((\gamma_{\ell,1}(0), \ldots, \gamma_{\ell,h}(0))) = f(\mathbf{a}_\ell).$$

Hence, the scheme $\Pi = \mathcal{T}(\Sigma)$ is correct when $t \geq kd + 1$.

Privacy: In the scheme Π, $\vec{\mathbf{a}} = (\mathbf{a}_1, \ldots, \mathbf{a}_s)$ is encoded with NEnc and then given to the server. The CR assumption implies that $\vec{\mathbf{a}}$ will be kept semantically secure against the server as long as $t = o((nk^h)^{1/(h+1)} + k + 1)$. Therefore, Π achieves input privacy (and thus output privacy) under the CR assumption.

Security: The security of Π requires that no adversary running in probabilistic polynomial-time should be able to persuade the verification algorithm to accept and output incorrect values on the input values. The proof of the following theorem is straightforward and left to Appendix A.

Theorem 1. *If the scheme Σ is secure under Definition 2, then Π is a secure verifiable computation scheme under this security definition.*

Efficiency: A verifiable computation scheme is outsourceable if the time to encode the input and verify the output is smaller than the time to compute the function from scratch. The existing verifiable computation schemes [6, 18] are in an amortized model, where the one-time cost of $\mathsf{KeyGen}(1^\lambda, f)$ is amortized over many different inputs. And for each input $\mathbf{x}$, the total time required for $\mathsf{ProbGen}(\mathrm{SK}_f, \mathbf{x})$ and $\mathsf{Verify}(\mathrm{SK}_f, \tau_{\mathbf{x}}, \sigma_{\mathbf{y}})$ is substantially less than the time required for computing $f(\mathbf{x})$ from scratch.

The scheme Π works in *two levels* of amortization. At the first level, the one-time cost of $\Pi.\mathsf{KeyGen}(1^\lambda, f)$ is amortized over the executions of the scheme on many different sets of inputs. At the second level, in every execution of $\Pi.\mathsf{ProbGen}(\mathrm{SK}_f, \vec{\mathbf{a}})$ and $\Pi.\mathsf{Verify}(\mathrm{SK}_f, \tau_{\vec{\mathbf{a}}}, \sigma_{\mathbf{y}})$, the client can process s function inputs $\vec{\mathbf{a}} = (\mathbf{a}_1, \ldots, \mathbf{a}_s)$ together; and the total time required for both algorithms, when averaged over the s function inputs, allows Π to be outsourceable. More precisely, $\vec{\mathbf{a}}$ is encoded as a set of $m = n - t + ts$ points. The time spent on $\Pi.\mathsf{ProbGen}(\mathrm{SK}_f, \vec{\mathbf{a}})$ is equal to the time spent on $\mathsf{NEnc}(k, \vec{\mathbf{a}})$, which is dominated by m computations of h-dimensional curve of degree k plus the total time spent on m executions of $\Sigma.\mathsf{ProbGen}$. The average time spent on processing each function input is dominated by m/s curve computations and m/s executions of $\Sigma.\mathsf{ProbGen}$. The time spent on $\Pi.\mathsf{Verify}(\mathrm{SK}_f, \tau_{\vec{\mathbf{a}}}, \sigma_{\mathbf{y}})$ is equal to the total time spent on m executions of $\Sigma.\mathsf{Verify}$ plus the total time spent on interpolations of s polynomials of degree $\leq t$. Therefore, the average time spent on verifying each output is dominated by the time spent on m/s executions of $\Sigma.\mathsf{Verify}$ plus the time spent on interpolation of a polynomial of degree $\leq t$.

There is a tradeoff between the number s of simultaneously delegated function inputs and the average time $\bar{T}$ spent on processing each function input and verifying its output. If we choose $s = O(n/t)$, then $\bar{T}$ will be dominated by $O(t)$ curve evaluations, $O(t)$ executions of $\Sigma.\mathsf{ProbGen}$, $O(t)$ executions of $\Sigma.\mathsf{Verify}$ and interpolation of a degree $\leq t$ univariate polynomial. For Π to be correct and secure, n, k, t, h, d should be chosen such that $kd < t \leq (nk^h)^{1/(h+1)} + k + 1$. As a result, one must have that $n \geq (d-1)^{h+1} \cdot k$, which is comparable to $N = \binom{h+d}{d}$, the number of coefficients of f. There are many ways to choose n, k, t, h, d such that $\bar{T} = o(N)$. As an example, if we choose

$$h = O(1), \quad d = \mathrm{poly}(k), \quad t = O(kd \log k) = o(N), \quad n = O(k^{h+2} d^{h+1} \log^{h+1} k),$$

$\bar{T}$ will be dominated by $O(kd \log k)$ curve computations, $O(kd \log k)$ executions of $\Sigma.\mathsf{ProbGen}$, $O(kd \log k)$ executions of $\Sigma.\mathsf{Verify}$ and interpolation of a polynomial of degree $\leq t$. Since Σ is outsourceable, Π is outsourceable as well.

Our transformation gives *efficient* verifiable computation schemes that enable the delegation of *high-degree* polynomial computations on *private (encrypted)* function inputs. In particular, our scheme neither relies on the expensive primitives such as fully homomorphic encryption (FHE) and garbled circuits nor has to represent the function f as a Boolean circuit. Even for very small k and d, our schemes are the *first* ensuring security and privacy without using expensive primitives. We can easily extend $\mathcal{T}$ such that it is not only applicable to privately verifiable schemes [6] but also applicable to publicly verifiable schemes [10, 16, 37]. Furthermore, $\mathcal{T}$ never changes the verifiability of the underlying scheme Σ.

Implementation: Applying our transformation to the privately delegatable and verifiable computation scheme Σ_{bgv} for multivariate polynomials of bounded total degree from Benabbas, Gennaro and Vahlis [6] gives a new scheme Π_{bgv} that achieves input and output privacy for the client. We implemented Π_{bgv} with a cyclic group of order $\geq 2^{1024}$, where strong DDH [6] is supposed true. Let $T_c(\cdot)$ and $T_s(\cdot)$ denote the average client running time and server running time. Our implementation shows that $T_c(\Pi_{\mathrm{bgv}}) = O(T_c(\Sigma_{\mathrm{bgv}}))$ and $T_s(\Pi_{\mathrm{bgv}}) = O(T_s(\Sigma_{\mathrm{bgv}}))$, where the constants hidden in O depend on k and d. The moderate efficiency loss stems from $\mathcal{T}$, which adds privacy to Σ_{bgv}. In contrast, the FHE-based schemes [2, 4, 13, 18] achieve input privacy but provide *no* implementations for polynomial computations. More precisely, we implemented Π_{bgv}

on a Dell Optiplex 9020 desktop with Intel Core i7-4790 Processor running at 3.6 GHz, on which we run Ubuntu 16.04.1 with 4 GB of RAM and the g++ compiler version 5.4.0. All our programs are single-threaded and built on top of NTL (and GMP). In order to achieve 128-bit security, the underlying scheme Π_{bgv} requires a cyclic group of order $\geq 2^{1024}$, where strong DDH assumption [6] is supposed to be true. We consider the computation of a 4-variate polynomial of total degree ≤ 6 at 504 points. The test shows that the client-side computations (Π_{bgv}.ProbeGen and Π_{bgv}.Verify) can be done very efficiently with total running time 27.616 seconds, and the average client's work for each of the 504 delegated computations is ≤ 0.88 milliseconds; the one-time work of running Π_{bgv}.KeyGen takes 0.636 seconds. On the other hand, the server's work of running Π_{bgv}.Compute takes 4444.24 seconds, which gives an amortized cost of 8.818 seconds for each of the 504 function inputs. Compared with the cost of 0.142 seconds in the non-private scheme, this high cost is the *price* of converting a non-private scheme to one that achieves privacy. This cost will become reasonable if the work of executing Π_{bgv}.Compute is done in parallel. The performance of our implementation shows that the resulting schemes of our transformation in this section is potentially practical.

4 Private delegation of matrix-vector multiplication

We interpret any matrix $M = (M_{i,j})$ as a function that takes a vector $\mathbf{x}$ as input and outputs $M \cdot \mathbf{x}'$, where $\mathbf{x}'$ is the transpose of $\mathbf{x}$. In this section, we present a verifiable computation scheme with an admissible function family of all matrix functions over a finite field, where the function input and output are kept private. Our construction is based on the somewhat homomorphic encryption, homomorphic hash and PRF with amortized closed-form efficiency.

4.1 Somewhat homomorphic encryption

A somewhat homomorphic encryption scheme allows one to evaluate low-degree polynomials on encrypted data. Fiore, Gennaro and Pastro [17] described a slight variation HE = (ParamGen, KeyGen, Eval, Enc, Dec) of the somewhat homomorphic encryption scheme by Brakerski and Vaikuntanathan [8], based on the hardness of the polynomial learning with error (LWE) problem. The variation is specialized to evaluate circuits of multiplicative depth 1 and sketched as below:

- HE.ParamGen(λ): Given the security parameter λ, generate
  - a message space $\mathcal{M} = \mathbb{Z}_p[X]/\Phi_m(X)$, where $\Phi_m(X) \in \mathbb{Z}[X]$ is the m-th cyclotomic polynomial of degree $\phi(m)$, where $\phi(\cdot)$ is the Euler totient function,
  - a ciphertext space $\mathcal{C} \subseteq \mathbb{Z}_q[X, Y]$ that consists of two kinds of elements:
    - level-0 ciphertext: $c = c_0 + c_1 Y$ with $c_0, c_1 \in \mathbb{Z}_q[X]/\Phi_m(X)$, where $q > p$, $\gcd(p, q) = 1$ and $\deg_X(c_i) \leq \phi(m) - 1$ for $i \in \{0, 1\}$,
    - level-1 ciphertext: $c = c_0 + c_1 Y + c_2 Y^2$, where $c_0, c_1, c_2 \in \mathbb{Z}_q[X]$ and $\deg_X(c_i) \leq 2(\phi(m) - 1)$ for $i \in \{0, 1, 2\}$,
  - two distributions: $D_{\mathbb{Z}^n,\sigma}$ and ZO_n.
- HE.KeyGen(1^λ): Choose $a \leftarrow \mathbb{Z}_q[X]/\Phi_m(X)$ and $s, e \leftarrow D_{\mathbb{Z}^n,\sigma}$; compute $b \leftarrow as + pe$; output $\mathsf{dk} = s$ and $\mathsf{pk} = (a, b)$.
- $\text{HE.Enc}_{\mathsf{pk}}(m, r)$: Given $m \in \mathcal{M}$ and $r = (u, v, w) \leftarrow (ZO_n, D_{\mathbb{Z}^n,\sigma}, D_{\mathbb{Z}^n,\sigma})$, compute $c_0 \leftarrow bu + pw + m$ and $c_1 \leftarrow au + pv$; output $c = c_0 + c_1 Y$.
- $\text{HE.Eval}_{\mathsf{pk}}(f, a, b)$: Given $a, b \in \mathcal{C}$, where $a = a_0 + a_1 Y + a_2 Y^2$, $b = b_0 + b_1 Y + b_2 Y^2$, homomorphic additions and multiplications (when $a_2 = b_2 = 0$) are done over $\mathbb{Z}_q[X, Y]$:
  - $(a_0 + a_1 Y + a_2 Y^2) + (b_0 + b_1 Y + b_2 Y^2) = (a_0 + b_0) + (a_1 + b_1)Y + (a_2 + b_2)Y^2$,
  - $(a_0 + a_1 Y) \cdot (b_0 + b_1 Y) = a_0 b_0 + (a_0 b_1 + a_1 b_0)Y + a_1 b_1 Y^2$.
- $\text{HE.Dec}_{\mathsf{dk}}(c)$: Given $c = c_0 + c_1 Y + c_2 Y^2 \in \mathcal{C}$, compute $c'_i = c_i \bmod \Phi_m(X)$ for $i = 0, 1, 2$; compute $t \in R_q$ as $t \leftarrow c'_0 - s \cdot c'_1 - s^2 \cdot c'_2$; output $(t \bmod p)$.

4.2 Homomorphic hash

A keyed homomorphic hash ($\mathsf{H.KeyGen}$, H, $\mathsf{H.Eval}$) is defined by three algorithms, where $\mathsf{H.KeyGen}$ generates two keys K (public) and κ (private), H uses K or κ to map any input $\mu \in \mathcal{D}$ to a digest $\mathsf{H}_K(\mu) \in \mathcal{R}$ and $\mathsf{H.Eval}$ allows homomorphic computations (addition "+", multiplication "$*$" and scalar multiplication "$\cdot$") over $\mathcal{R}$. Let $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be a tuple of bilinear group parameters, and let

$$\mathcal{D} = \{\mu \in \mathbb{Z}_q[X, Y] : \deg_X(\mu) \le 2(\phi(m) - 1),\ \deg_Y(\mu) \le 2\}.$$

The following homomorphic hash with domain $\mathcal{D}$ and range $\mathcal{R} = \mathbb{G}_1 \times \mathbb{G}_2$ (or $\mathbb{G}_T$) is from [17].

- $\mathsf{H.KeyGen}(1^\lambda)$: Choose $\alpha, \beta \leftarrow \mathbb{Z}_q$; output a public key

$$K = \{(g^{\alpha^i\beta^j}, h^{\alpha^i\beta^j}) : i \in \{0, 1, 2\},\ j \in \{0, 1, \ldots, 2(\phi(m) - 1)\}\}$$

  and a matching secret key $\kappa = (\alpha, \beta)$; both allow the computation of hash digest, and the latter usually makes the computation more efficient.
- $\mathsf{H}_K(\mu)$: Given an input $\mu \in \mathcal{D}$, if $\deg_Y(\mu) \le 1$, then output $(g^{\mu(\beta,\alpha)}, h^{\mu(\beta,\alpha)})$ as the digest; if $\deg_Y(\mu) = 2$, then output $e(g, h)^{\mu(\beta,\alpha)}$ as the digest. In particular, when $\deg_Y(\mu) \le 1$, we denote $[\mathsf{H}_K(\mu)]_1 = g^{\mu(\beta,\alpha)}$ and $[\mathsf{H}_K(\mu)]_2 = h^{\mu(\beta,\alpha)}$.
- $\mathsf{H.Eval}(f, v_1, v_2)$: This algorithm enables the homomorphic computations of arithmetic circuits f of degree ≤ 2 as below:
  - $v_1 = (t_1, u_1),\ v_2 = (t_2, u_2) \in \mathbb{G}_1 \times \mathbb{G}_2$, $f =$ "+": output (t_1t_2, u_1u_2);
  - $v_1 = (t_1, u_1) \in \mathbb{G}_1 \times \mathbb{G}_2$, $v_2 = c \in \mathbb{Z}_q$, $f =$ "$\cdot$": output (t_1^c, u_1^c);
  - $v_1 = (t_1, u_1),\ v_2 = (t_2, u_2) \in \mathbb{G}_1 \times \mathbb{G}_2$, $f =$ "$*$": output $e(t_1, u_2) \in \mathbb{G}_T$;
  - $v_1, v_2 \in \mathbb{G}_T$, $f =$ "+": output $v_1v_2 \in \mathbb{G}_T$;
  - $v_1 \in \mathbb{G}_T$, $v_2 = c \in \mathbb{Z}_q$, $f =$ "$\cdot$": output $v_1^c \in \mathbb{G}_T$.

The homomorphic hash H was shown collision-resistant under the ℓ-BDHI assumption. That is, when $\ell \ge \max\{2(\phi(m) - 1), 2\}$, for any $(K, \kappa) \leftarrow \mathsf{H.KeyGen}(1^\lambda)$, for any adversary $\mathcal{A}$ running in probabilistic polynomial time,

$$\Pr[(\mu \ne \mu') \wedge (\mathsf{H}_K(\mu) = \mathsf{H}_K(\mu')) : (\mu, \mu') \leftarrow \mathcal{A}(K)] \le \mathrm{negl}(\lambda).$$

4.3 PRFs with amortized closed-form efficiency

A pseudorandom function ($\mathsf{F.KG}$, F) is defined by two algorithms, where the key generation algorithm $\mathsf{F.KG}$ takes as input the security parameter 1^λ and outputs a secret key k and some public parameters pp that specify domain $\mathcal{X}$ and range $\mathcal{R}$ of the function, and the function $\mathsf{F}_k(x)$ takes input $x \in \mathcal{X}$ and uses the secret key k to compute a value $R \in \mathcal{R}$. The PRF ($\mathsf{F.KG}$, F) is said to be *secure* (satisfy the pseudorandomness property) if, for any p.p.t. adversary $\mathcal{A}$,

$$|\Pr[\mathcal{A}^{\mathsf{F}_k(\cdot)}(1^\lambda, \mathsf{pp}) = 1] - \Pr[\mathcal{A}^{\Phi(\cdot)}(1^\lambda, \mathsf{pp}) = 1]| \le \mathrm{negl}(\lambda),$$

where $(k, \mathsf{pp}) \leftarrow \mathsf{F.KG}(1^\lambda)$ and $\Phi : \mathcal{X} \to \mathcal{R}$ is a random function.

Let C be a computation that takes as input n random values $R_1, \ldots, R_n \in \mathcal{R}$ and a vector of m arbitrary values $\boldsymbol{z} = (z_1, \ldots, z_m)$, and assume that the computation of $C(R_1, \ldots, R_n; z_1, \ldots, z_m)$ requires time $t(n, m)$. Let $\boldsymbol{L} = ((\xi, \eta_1), \ldots, (\xi, \eta_n)) \in \mathcal{X}^n$ and $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$. The PRF ($\mathsf{F.KG}$, F) is said to satisfy the *amortized closed-form efficiency* for $(C, \boldsymbol{L})$ if there exist two polynomial-time algorithms $\mathsf{CFEval}^{\mathrm{off}}_{C,\boldsymbol{\eta}}$ and $\mathsf{CFEval}^{\mathrm{on}}_{C,\xi}$ such that

(1) for any $\omega \leftarrow \mathsf{CFEval}^{\mathrm{off}}_{C,\boldsymbol{\eta}}(k, \boldsymbol{z})$, $\mathsf{CFEval}^{\mathrm{on}}_{C,\xi}(k, \omega) = C(\{\mathsf{F}_k(\xi, \eta_j)\}_{j=1}^n; \boldsymbol{z})$,

(2) the running time of $\mathsf{CFEval}^{\mathrm{on}}_{C,\xi}(k, \omega)$ is $o(t)$.

Let $f(x_1, \ldots, x_n) = \sum_{i,j=1}^n \alpha_{i,j}x_ix_j + \sum_{i=1}^n \beta_ix_i$ be a degree-2 arithmetic circuit defined by $\boldsymbol{f} = \{\alpha_{i,j}, \beta_i\}_{i,j=1}^n$. Let $C : (\mathbb{G}_1 \times \mathbb{G}_2)^n \times \mathbb{Z}_q^{n^2+n} \to \mathbb{G}_T$ be a computation defined by

$$C(\{(X_i, Y_i)\}_{i=1}^n, \boldsymbol{f}) = \prod_{i,j=1}^n e(X_i, Y_j)^{\alpha_{i,j}} \cdot \prod_{i=1}^n e(X_i, h)^{\beta_i}.$$

Let $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be a tuple of bilinear group parameters, and let F' be a PRF with domain $\{0,1\}^*$ and range $\mathbb{Z}_q^2$. Fiore, Gennaro and Pastro [17] proposed a PRF with amortized closed-form efficiency for $(C, \boldsymbol{L})$.

- $\mathsf{F.KG}(1^\lambda)$: Choose two secret keys k_1, k_2 for the PRF F'; output $k = (k_1, k_2)$ and pp, where pp defines the domain $\mathcal{X} = (\{0,1\}^*)^2$ and the range $\mathcal{R} = \mathbb{G}_1 \times \mathbb{G}_2$ (or $\mathbb{G}_T$).
- $\mathsf{F}_k(\xi, \eta)$: Compute $(u, v) \leftarrow \mathsf{F}'_{k_1}(\eta)$, $(a, b) \leftarrow \mathsf{F}'_{k_2}(\xi)$; output (g^{ua+vb}, h^{ua+vb}); in particular, we denote
$$[\mathsf{F}_k(\xi, \eta)]_1 = g^{ua+vb}, \quad [\mathsf{F}_k(\xi, \eta)]_2 = h^{ua+vb}.$$
- $\mathsf{CFEval}^{\mathrm{off}}_{C,\boldsymbol{\eta}}(k, \boldsymbol{f})$: Compute $(u_i, v_i) \leftarrow \mathsf{F}'_{k_1}(\eta_i)$ for all $i \in [n]$; let
$$\omega(z_1, z_2) = f(u_1 \cdot z_1 + v_1 \cdot z_2, \ldots, u_n \cdot z_1 + v_n \cdot z_2);$$
output the bivariate polynomial ω.
- $\mathsf{CFEval}^{\mathrm{on}}_{C,\xi}(k, \omega)$: Compute $(a, b) \leftarrow \mathsf{F}'_{k_2}(\xi)$; output $e(g, h)^{\omega(a,b)}$.

4.4 The construction

In this section, we present a private verifiable computation scheme Γ with an admissible function family of all matrix functions over a finite field. In this scheme, the function to be delegated is a square matrix $M = (M_{i,j})$ of order n, and the input is a vector $\mathbf{x} = (x_1, \ldots, x_n)$ of dimension n; the server is required to compute and reply with an encoding of $M \cdot \mathbf{x}'$, where $\mathbf{x}'$ is the transpose of $\mathbf{x}$. The input (and output) privacy of Γ is attained by the client encrypting $\mathbf{x}$ (as $\mathsf{HE.Enc}(\mathbf{x})$) and then giving it to the server. The somewhat homomorphic encryption scheme used here allows the server to perform homomorphic scalar multiplications and additions on the elements of M (in clear) and the ciphertext of $\mathbf{x}$, which gives an encrypted version of $M \cdot \mathbf{x}'$ for the client. The server is able to compute the homomorphic hash digests of $\mathsf{HE.Enc}(\mathbf{x})$ and combine these digests with the tags of M to generate a proof that its computation is correct. The homomorphic property of the hash and the amortized closed-form efficiency property of the PRF makes the client's verification significantly faster than the computation of $M \cdot \mathbf{x}'$ from scratch. Below is the description of Γ.

- $(\mathrm{PK}_M, \mathrm{SK}_M) \leftarrow \Gamma.\mathsf{KeyGen}(1^\lambda, M)$:
  - run $\mathsf{HE.ParamGen}(1^\lambda)$ to choose message and ciphertext spaces $\mathcal{M} = \mathbb{Z}_p[X]/\Phi_m(X)$ and $\mathcal{C} \subseteq \mathbb{Z}_q[X, Y]$.
  - run $\mathsf{HE.KeyGen}(1^\lambda)$ to generate an encryption key pk and a decryption key dk for the encryption scheme HE;
  - choose a tuple $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ of bilinear map parameters;
  - run $\mathsf{H.KeyGen}(1^\lambda)$ to choose two keys K (public) and κ (private) for the homomorphic hash H;
  - run $\mathsf{F.KG}(1^\lambda)$ to generate a secret key $k = (k_1, k_2)$ for the PRF F; choose $a \leftarrow \mathbb{Z}_q$;
  - compute $T_{i,j} = g^{aM_{i,j}} \cdot [\mathsf{F}_k(i, j)]_1$ for all $(i, j) \in [n]^2$;
  - output $\mathrm{PK}_M = (p, m, n, \mathsf{bgpp}, \mathsf{pk}, K, M, T = (T_{i,j}))$, $\mathrm{SK}_M = (\mathsf{dk}, \kappa, k, a)$.
- $(\sigma_{\mathbf{x}}, \tau_{\mathbf{x}}) \leftarrow \Gamma.\mathsf{ProbGen}(\mathrm{SK}_M, \mathbf{x})$: let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_q^n$;
  - for every $j \in [n]$, compute $\mu_j \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(x_j)$;
  - parse κ as $(\alpha, \beta) \in \mathbb{Z}_q^2$; compute $\omega = \sum_{j=1}^n \mu_j(\beta, \alpha) \cdot \mathsf{F}'_{k_1}(j) \in \mathbb{Z}_q^2$;
  - let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$; output $\sigma_{\mathbf{x}} = \boldsymbol{\mu}$ and $\tau_{\mathbf{x}} = \omega$.
- $\sigma_{\mathbf{y}} \leftarrow \Gamma.\mathsf{Compute}(\mathrm{PK}_M, \sigma_{\mathbf{x}})$: parse $\sigma_{\mathbf{x}}$ as $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n) \in \mathcal{C}^n$;
  - for every $i \in [n]$, compute $\gamma_i = \sum_{j=1}^n M_{i,j} \cdot \mu_j$;
  - for every $i \in [n]$, compute $\delta_i = \prod_{j=1}^n e(T_{i,j}, [\mathsf{H}_K(\mu_j)]_2)$;
  - let $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n)$; let $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_n)$; output $\sigma_{\mathbf{y}} = (\boldsymbol{\gamma}, \boldsymbol{\delta})$.
- $\{\mathbf{y}, \perp\} \leftarrow \Gamma.\mathsf{Verify}(\mathrm{SK}_M, \tau_{\mathbf{x}}, \sigma_{\mathbf{y}})$:
  - for every $i \in [n]$, let $W_i = e(g, h)^{\langle \omega, \mathsf{F}'_{k_2}(i) \rangle}$, where $\langle \omega, \mathsf{F}'_{k_2}(i) \rangle$ is the dot product of ω and $\mathsf{F}'_{k_2}(i)$; check if the following identity holds:
$$\delta_i = e(g, h)^{a \cdot \gamma_i(\beta, \alpha)} \cdot W_i; \tag{$\star$}$$
  - if there is an $i \in [n]$ such that ($\star$) does not hold, then output $\perp$; otherwise, output
$$\mathbf{y} = (\mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_1), \ldots, \mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_n)).$$

Correctness: The correctness of Γ requires that, for any matrix function M, any key pair

$$(\mathrm{PK}_M, \mathrm{SK}_M) \leftarrow \Gamma.\mathsf{KeyGen}(1^\lambda, M),$$

any function input $\mathbf{x}$, any $(\sigma_\mathbf{x}, \tau_\mathbf{x}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_M, \mathbf{x})$, if $\sigma_\mathbf{y}$ is output by the algorithm $\mathsf{Compute}(\mathrm{PK}_M, \sigma_\mathbf{x})$, then $\mathsf{Verify}(\mathrm{SK}_M, \tau_\mathbf{x}, \sigma_\mathbf{y})$ will always accept and output $M \cdot \mathbf{x}'$. For every $i \in [n]$, if Compute was honestly executed, then it is not hard to verify that

$$\delta_i = \prod_{j=1}^{n} e(g^{aM_{i,j}} \cdot g^{\langle \mathsf{F}'_{k_1}(j), \mathsf{F}'_{k_2}(i)\rangle}, h^{\mu_j(\beta,\alpha)}) = e(g,h)^{a\cdot\gamma_i(\beta,\alpha)} \cdot e(g,h)^{\langle \omega, \mathsf{F}'_{k_2}(i)\rangle}.$$

Hence, the n equalities always hold, and $\sigma_\mathbf{y}$ will be accepted. Then the decryption correctness of HE gives $(\mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_1), \ldots, \mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_n)) = M \cdot \mathbf{x}'$.

Privacy: In the scheme Γ, the client's input $\mathbf{x}$ is encrypted using HE, the slight variation of the somewhat homomorphic encryption scheme by Brakerski and Vaikuntanathan [8]. The encryption scheme is semantically secure based on the hardness of the polynomial learning with error (LWE) problem. The input (and output) privacy of Γ follows from HE's semantic security.

Security: The security of Γ requires that no adversary running in probabilistic polynomial time would be able to persuade the verification algorithm to accept and output wrong results.

Theorem 2. *If* F *is a secure PRF and* H *is a collision-resistant homomorphic hash, then* Γ *is a secure verifiable computation scheme.*

Proof. Let λ be any security parameter. Let $M = (M_{i,j})$ be any $n \times n$ matrix. Let $\mathcal{A}$ be any p.p.t. adversary. We define the following security experiments.

E_0: This is the standard security experiment $\mathsf{Exp}^{\mathsf{Ver}}_{\Gamma,\mathcal{A}}(M, \lambda)$ of Definition 2.

E_1: This experiment is identical to E_0, except that, at step (d) of the standard security experiment, the W_i is computed as $W_i = \prod_{j=1}^{n} e(\mathsf{F}_k(i,j), h)^{\mu_j(\beta,\alpha)}$ for every $i \in [n]$, instead of using the key ω for efficient verification (and therefore avoid the use of $\mathsf{CFEval}^{\mathrm{on}}$).

E_2: This is identical to E_1, except that the F_k is replaced with a random function $R\colon (\{0,1\}^*)^2 \to \mathbb{G}_1 \times \mathbb{G}_2$.

Below is the description of E_2.

- $(\mathrm{PK}_M, \mathrm{SK}_M) \leftarrow \mathsf{KeyGen}(1^\lambda, M)$:
  - run $\mathsf{HE.ParamGen}(1^\lambda)$ to generate $(p, q, \mathcal{M}, \mathcal{C})$, where $q > p$, $\gcd(p,q) = 1$, $\mathcal{M} = \mathbb{Z}_p[X]/\Phi_m(X)$ and $\mathcal{C} \subseteq \mathbb{Z}_q[X, Y]$ is the ciphertext space;
  - run $\mathsf{HE.KeyGen}(1^\lambda)$ to generate $(\mathsf{pk}, \mathsf{dk})$;
  - choose $\mathsf{bgpp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h) \leftarrow \mathcal{G}(1^\lambda)$;
  - run H.KeyGen to generate keys (K, κ) for the homomorphic hash H;
  - choose a random function $R\colon (\{0,1\}^*)^2 \to \mathbb{G}_1 \times \mathbb{G}_2$; choose $a \leftarrow \mathbb{Z}_q$;
  - compute $T_{i,j} = g^{aM_{i,j}} \cdot [R(i,j)]_1$ for all $(i,j) \in [n]^2$; let $T = (T_{i,j})$;
  - output $\mathrm{PK}_M = (p, m, n, \mathsf{bgpp}, \mathsf{pk}, K, M, T)$ and $\mathrm{SK}_M = (\mathsf{dk}, \kappa, R, a)$.
- For $\ell = 1$ to $L = \mathrm{poly}(\lambda)$:
  - (a) $\mathbf{x}_\ell \leftarrow \mathcal{A}(\mathrm{PK}_M, \{(\mathbf{x}_u, \sigma_{\mathbf{x}_u}, b_u)\}_{u=1}^{\ell-1})$: Based on the current view, $\mathcal{A}$ chooses a new function input
  $$\mathbf{x}_\ell = (x_{\ell,1}, \ldots, x_{\ell,n}) \in \mathbb{Z}_q^n.$$
  - (b) $(\sigma_{\mathbf{x}_\ell}, \tau_{\mathbf{x}_\ell}) \leftarrow \mathsf{ProbGen}(\mathrm{SK}_M, \mathbf{x}_\ell)$: An encoding and the associated verification key for $\mathbf{x}_\ell$ are generated as below:
    - for every $j \in [n]$, compute $\mu_{\ell,j} \leftarrow \mathsf{HE.Enc}_{\mathsf{pk}}(x_{\ell,j})$;
    - let $\boldsymbol{\mu}_\ell = (\mu_{\ell,1}, \ldots, \mu_{\ell,n})$; output $\sigma_{\mathbf{x}_\ell} = \boldsymbol{\mu}_\ell$ and $\tau_{\mathbf{x}_\ell} = \perp$.

    Note that $\tau_{\mathbf{x}_\ell}$ is neither computed nor used in experiment E_2.
  - (c) $\hat{\sigma}_{\mathbf{y}_\ell} \leftarrow \mathcal{A}(\mathrm{PK}_M, \{(\mathbf{x}_u, \sigma_{\mathbf{x}_u}, b_u)\}_{u=1}^{\ell-1}, \mathbf{x}_\ell, \sigma_{\mathbf{x}_\ell})$: Based on the current view, $\mathcal{A}$ provides a response $\hat{\sigma}_{\mathbf{y}_\ell}$ that consists of an encoding $\hat{\boldsymbol{y}}_\ell = (\hat{y}_{\ell,1}, \ldots, \hat{y}_{\ell,n})$ of the result and a proof $\hat{\boldsymbol{\delta}}_\ell = (\hat{\delta}_{\ell,1}, \ldots, \hat{\delta}_{\ell,n})$ in order to persuade Verify to accept and output a wrong result.

(d) $\hat{\mathbf{y}}_\ell \leftarrow \mathsf{Verify}(\mathrm{SK}_M, \tau_{\mathbf{x}_\ell}, \hat{\sigma}_{\mathbf{y}_\ell})$: The response $\hat{\sigma}_{\mathbf{y}_\ell}$ is verified as below:

– parse SK_M as $(\mathsf{dk}, \kappa, R, a)$; parse $\hat{\sigma}_{\mathbf{y}_\ell}$ as $\hat{\boldsymbol{y}}_\ell = (\hat{y}_{\ell,1}, \ldots, \hat{y}_{\ell,n}) \in \mathbb{Z}_q[X, Y]$, and

$$\hat{\boldsymbol{\delta}}_\ell = (\hat{\delta}_{\ell,1}, \ldots, \hat{\delta}_{\ell,n}) \in \mathbb{G}_T^n;$$

– for every $i \in [n]$, compute $W_i = \prod_{j=1}^n e([R(i,j)]_1, h)^{\mu_{\ell,j}(\beta,\alpha)}$; check if

$$\hat{\delta}_{\ell,i} = e(g,h)^{a\cdot\hat{y}_{\ell,i}(\beta,\alpha)} \cdot W_i; \tag{$\star$}$$

– if there is an $i \in [n]$ such that ($\star$) is not true, then output $\hat{\mathbf{y}}_\ell = \perp$; otherwise, output

$$\hat{\mathbf{y}}_\ell = (\mathsf{HE.Dec}_{\mathsf{dk}}(\hat{y}_{\ell,1}), \ldots, \mathsf{HE.Dec}_{\mathsf{dk}}(\hat{y}_{\ell,n})).$$

(e) Set $b_\ell = 1$ if $\hat{\mathbf{y}}_\ell \neq \perp$; otherwise, set $b_\ell = 0$.

- Output 1 if there exists $\ell \in [L]$ such that $b_\ell = 1$ but $\hat{\mathbf{y}}_\ell \neq M\mathbf{x}'_\ell$; otherwise, output 0.

Let $\Pr[E_0 = 1]$, $\Pr[E_1 = 1]$ and $\Pr[E_2 = 1]$ be the probabilities that $\mathcal{A}$ wins in E_0, E_1 and E_2, respectively. We need to show that $\Pr[E_0 = 1] \le \mathrm{negl}(\lambda)$. The only difference between E_1 and E_0 is that, in E_1, the algorithm $\mathsf{CFEval}^{\mathrm{on}}$ is not used. This will not change the probability that $\mathcal{A}$ wins. Therefore, $\Pr[E_1 = 1] = \Pr[E_0 = 1]$. The only difference between E_2 and E_1 is that, in E_2, the function F_k is replaced with a random function R. The security of F implies that E_1 and E_2 are computationally indistinguishable. Therefore, we have $|\Pr[E_1 = 1] - \Pr[E_2 = 1]| \le \mathrm{negl}(\lambda)$. To prove $\Pr[E_0 = 1] \le \mathrm{negl}(\lambda)$, it suffices to show that $\Pr[E_2 = 1] \le \mathrm{negl}(\lambda)$.

For every $\ell \in [L]$, let $E_{2,\ell} = 1$ be the event that $\mathbf{y}_\ell \notin \{\perp, M\mathbf{x}'_\ell\}$, i.e., the event that $\mathcal{A}$'s response

$$\hat{\sigma}_{\mathbf{y}_\ell} = ((\hat{y}_{\ell,1}, \ldots, \hat{y}_{\ell,n}), (\hat{\delta}_{\ell,1}, \ldots, \hat{\delta}_{\ell,n}))$$

for $\mathbf{x}_\ell$ suffices to persuade Verify to accept and output a wrong result for the computation of $M\mathbf{x}'_\ell$. More formally, $E_{2,\ell}$ occurs if and only if

- $\hat{\delta}_{\ell,i} = e(g,h)^{a\cdot\hat{y}_{\ell,i}(\beta,\alpha)} \cdot \prod_{j=1}^n e([R(i,j)]_1, h)^{\mu_{\ell,j}(\beta,\alpha)}$ for every $i \in [n]$,
- but $\hat{\mathbf{y}}_\ell = (\mathsf{HE.Dec}_{\mathsf{dk}}(\hat{y}_{\ell,1}), \ldots, \mathsf{HE.Dec}_{\mathsf{dk}}(\hat{y}_{\ell,n})) \neq M\mathbf{x}'_\ell$.

Then $E_2 = 1$ occurs only if there is at least one $\ell \in [L]$ such that $E_{2,\ell}$ occurs.

For $\mathbf{x}_\ell \in \mathbb{Z}_q^n$ and its encoding $\sigma_{\mathbf{x}_\ell} = \boldsymbol{\mu}_\ell = (\mu_{\ell,1}, \ldots, \mu_{\ell,n}) \in \mathbb{C}^n$, let

$$\sigma_{\mathbf{y}_\ell} = (\boldsymbol{\gamma}_\ell, \boldsymbol{\delta}_\ell) = ((\gamma_{\ell,1}, \ldots, \gamma_{\ell,n}), (\delta_{\ell,1}, \ldots, \delta_{\ell,n}))$$

be the (correct) result, and proof which could be computed by faithfully running $\mathsf{Compute}(\mathrm{PK}_M, \sigma_{\mathbf{x}_\ell})$. Then the correctness of Γ guarantees that

- $\delta_{\ell,i} = e(g,h)^{a\cdot\gamma_{\ell,i}(\beta,\alpha)} \cdot \prod_{j=1}^n e([R(i,j)]_1, h)^{\mu_{\ell,j}(\beta,\alpha)}$ for every $i \in [n]$,
- $\mathbf{y}_\ell = (\mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_{\ell,1}), \ldots, \mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_{\ell,n})) = M\mathbf{x}'_\ell$.

For every $\ell \in [L]$, let F_ℓ be the event that

- $\hat{\delta}_{\ell,i}/\delta_{\ell,i} = e(g,h)^{a\cdot(\hat{y}_{\ell,i}(\beta,\alpha)-\gamma_{\ell,i}(\beta,\alpha))}$ for every $i \in [n]$,
- but $\hat{\mathbf{y}}_\ell \neq \mathbf{y}_\ell$.

For every $\ell \in [L]$, the event $E_{2,\ell} = 1$ occurs only if the event F_ℓ occurs. For every $\ell \in [L]$ and $j \in [n]$, let $G_{\ell,j}$ be the event that

- $\hat{\delta}_{\ell,i}/\delta_{\ell,i} = e(g,h)^{a\cdot(\hat{y}_{\ell,i}(\beta,\alpha)-\gamma_{\ell,i}(\beta,\alpha))}$ for every $i \in [n]$,
- but $\mathsf{HE.Dec}_{\mathsf{dk}}(\hat{y}_{\ell,j}) \neq \mathsf{HE.Dec}_{\mathsf{dk}}(\gamma_{\ell,j})$.

Then F_ℓ occurs only if there is at least one $j \in [n]$ such that $G_{\ell,j}$ occurs. For every $\ell \in [L]$ and $j \in [n]$, let $H_{\ell,j}$ be the event that

- $\hat{\delta}_{\ell,i}/\delta_{\ell,i} = e(g,h)^{a\cdot(\hat{y}_{\ell,i}(\beta,\alpha)-\gamma_{\ell,i}(\beta,\alpha))}$ for every $i \in [n]$,
- but $\hat{y}_{\ell,j} \neq \gamma_{\ell,j}$.

Then $G_{\ell,j}$ occurs only if $H_{\ell,j}$ occurs. For every $\ell \in [L]$ and $j \in [n]$, let $H^0_{\ell,j}$ be the event that $\hat{y}_{\ell,j} \neq \gamma_{\ell,j}$ and $\hat{y}_{\ell,j}(\beta,\alpha) = \gamma_{\ell,j}(\beta,\alpha)$. Let $H^1_{\ell,j}$ be the event that $\hat{\delta}_{\ell,i}/\delta_{\ell,i} = e(g,h)^{a\cdot(\hat{y}_{\ell,i}(\beta,\alpha)-\gamma_{\ell,i}(\beta,\alpha))}$ for every $i \in [n]$, but $\hat{y}_{\ell,j}(\beta,\alpha) \neq \gamma_{\ell,j}(\beta,\alpha)$. Then it is clear that $H_{\ell,j}$ occurs only if at least one of $H^0_{\ell,j}$ or $H^1_{\ell,j}$ occurs.

Let X_c be a random variable that denotes the first index $(\ell, j) \in [L] \times [n]$ such that $H^c_{\ell,j}$ occurs for every $c \in \{0,1\}$. In both cases, we rank the elements $(\ell, j) \in [L] \times [n]$ as

$$(1,1) < \cdots < (1,n) < (2,1) < \cdots < (2,n) < \cdots < (L,n).$$

Due to the union bound, we have that

$$\Pr[E_2 = 1] \le \sum_{\ell=1}^{L} \Pr[E_{2,\ell}] \le \sum_{\ell=1}^{L} \Pr[F_\ell] \le \sum_{\ell=1}^{L}\sum_{j=1}^{n} \Pr[G_{\ell,j}] \le \sum_{\ell=1}^{L}\sum_{j=1}^{n} \Pr[H_{\ell,j}]$$
$$\le \sum_{\ell=1}^{L}\sum_{j=1}^{n}\sum_{c=0}^{1} \Pr[H^c_{\ell,j}] \le \sum_{\ell=1}^{L}\sum_{j=1}^{n}\sum_{c=0}^{1} \sum_{(\phi,\psi)\le(\ell,j)} \Pr[X_c = (\phi,\psi)].$$

Note that $X_0 = (\phi, \psi)$ means that (ϕ, ψ) is the first index such that a collision of H_K is found by the adversary. As H is collision resistant, we must have that $\Pr[X_0 = (\phi, \psi)] \le \mathrm{negl}(\lambda)$. On the other hand, $X_1 = (\phi, \psi)$ means that (ϕ, ψ) is the first index such that an equation about a is determined by $\mathcal{A}$ and thus gives a when $\mathcal{A}$ is computationally unbounded. Note that a computationally unbounded $\mathcal{A}$ can rule out one possibility of a via any one of the inequalities of the form $\hat{\delta}_{\ell,i}/\delta_{\ell,i} \ne e(g,h)^{a\cdot(\hat{\gamma}_{\ell,i}(\beta,\alpha)-\gamma_{\ell,i}(\beta,\alpha))}$. Therefore,

$$\Pr[X_1 = (\phi,\psi)] \le \frac{1}{q-((\phi-1)n+\psi-1)}.$$

It follows that

$$\Pr[E_2 = 1] \le \sum_{\ell=1}^{L}\sum_{i=1}^{n} \sum_{(\phi,\psi)\le(\ell,i)} \left(\mathrm{negl}(\lambda) + \frac{1}{q-((\phi-1)n+\psi-1)}\right),$$

which is negligible in λ as $q \approx 2^\lambda$, $n = \mathrm{poly}(\lambda)$ and $L = \mathrm{poly}(\lambda)$. □

Efficiency: A VC scheme is outsourceable if the time to encode the input and verify the output is smaller than the time to compute the function from scratch. For every $\mathbf{x} \in \mathbb{Z}_q^n$, the time spent on $\Gamma.\mathsf{ProbGen}(\mathrm{SK}_M, \mathbf{x})$ is equal to the time of computing $\{\mathsf{HE.Enc}_{\mathsf{pk}}(x_i)\}_{i=1}^n$ plus the time of computing ω (n PRF computations and $O(n)$ field operations). For every $\sigma_{\mathbf{y}}$, the time cost of $\Gamma.\mathsf{Verify}(\mathrm{SK}_M, \tau_{\mathbf{x}}, \sigma_{\mathbf{y}})$ is dominated by $O(n)$ group operations and n executions of $\mathsf{HE.Dec}_{\mathsf{sk}}$. Note that $M \cdot \mathbf{x}'$ requires $O(n^2)$ field operations. When n is large enough, the client's cost of running $\Gamma.\mathsf{ProbGen}$ and $\Gamma.\mathsf{Verify}$ is $o(n^2)$ and substantially less than that of computing $M \cdot \mathbf{x}'$ from scratch. Hence, Γ is outsourceable.

Implementation: We implemented the scheme Γ on a Dell Optiplex 9020 desktop with Intel Core i7-4790 Processor running at 3.6 GHz, on which we run Ubuntu 16.04.1 with 4 GB of RAM and the g++ compiler version 5.4.0. All our programs are single-threaded and built on top of GMP. We consider the multiplication between a random square matrix of n rows (columns) over a finite field of order $> 2^{256}$ and a random vector of dimension n over the same field, for $n = 100, 200, \ldots, 1000$. We record the client's time of running $\Gamma.\mathsf{ProbGen}$ and $\Gamma.\mathsf{Verify}$, and get Figure 3.

The experiment shows that, for $n = 100$, the client-side computation can be done in 0.89 seconds. If we use the scheme Γ in a natural way to delegate the multiplication of two 100×100 matrices, then the client-side computation can be done in at most 89 seconds. Parno, Howell, Gentry and Raykova [38] implemented the scheme of [18] to delegate the same computation. Their experiment shows that the client in [18] has to spend at least 10^{11} seconds on problem generation and result verification. Compared with [18], the client in our scheme is faster with an order of 9. The performance of our implementation shows that the resulting schemes of our transformation in this section is nearly practical.

5 Application

Our verifiable computation schemes have interesting applications in the design of outsourced two-party protocols such as outsourced private information retrieval (PIR). PIR [12] allows a client to retrieve any block f_i of a database $f = (f_1, f_2, \ldots, f_N)$ from a server such that $i \in [N]$ is not revealed to the server. PIR can be achieved by the client downloading f but that requires a communication cost of $O(N)$. There are PIR schemes [12, 31] in the semi-honest server model which achieve nontrivial communication cost $o(N)$. Recently, outsourced PIR [25, 34] has been suggested to offload the PIR server computation [5] to cloud. Outsourcing requires PIRs that are secure against untrusted cloud servers which may not faithfully execute the schemes.

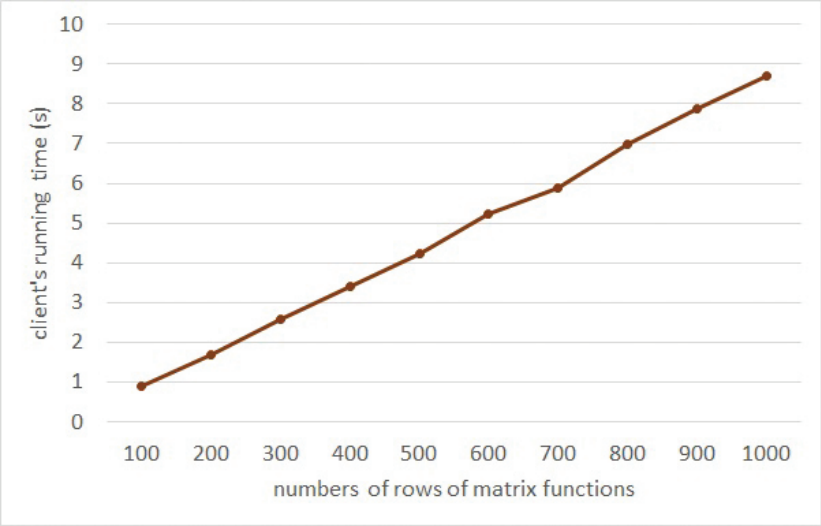

Figure 3: Performance of the matrix delegation scheme.

Solution based on $\mathcal{T}$: We model the database as a bit string $f = (f_1, f_2, \ldots, f_N) \in \{0,1\}^N$. Let $E\colon [N] \to \{0,1\}^h$ be an injection such that, for every $i \in [N]$, $\mathrm{wt}(E(i)) = d = \lfloor \frac{t-1}{k} \rfloor$, where the integers h and d are chosen such that $\binom{h}{d} \geq N$. The database (bit string) f is interpreted as an h-variate polynomial

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_h) = \sum_{j=1}^{N} f_j \cdot \prod_{\ell : E(j)_\ell = 1} x_\ell.$$

It is trivial to see that $f(E(j)) = f_j$ for every $j \in [N]$. Suppose a number of clients are to retrieve s bits of the database, say $f_{i_1}, \ldots, f_{i_s}$, where $i_1, \ldots, i_s \in [N]$. The encodings $\mathbf{a}_1 = E(i_1), \ldots, \mathbf{a}_s = E(i_s)$ form a set of s function inputs. Our scheme $\mathcal{T}(\Sigma)$ from Section 3.3 allows the clients to produce a noisy encoding of $\vec{\mathbf{a}} = (\mathbf{a}_1, \ldots, \mathbf{a}_s)$ and delegate the computations of $\{f_{i_j}\}_{j=1}^s = \{f(\mathbf{a}_j)\}_{j=1}^s$ to a PIR server such that $\vec{\mathbf{a}}$ is kept private and any incorrect responses from the server will be detected. The amortized communication cost for each of the s retrievals is dominated by $O(t)$ vectors from $\mathbb{F}^h$, which gives a nontrivial outsourced PIR.

Solution based on Γ: We model the database as an $n \times n$ matrix $M = (M_{i,j})$. Suppose that the client is interested in $M_{i,j}$. Then it suffices for the client to retrieve the i-th row of M, i.e., $(M_{i,1}, \ldots, M_{i,n})$. This retrieval can be captured by $M \cdot \mathbf{x}'$ with $\mathbf{x} = \mathbf{e}_i = (0, \ldots, 1_i, \ldots, 0)$, the vector whose i-th component is 1 and all other components are 0. The scheme Γ allows the client to delegate the computation of $M \cdot \mathbf{x}'$ with $\mathbf{x}$ being kept private and then verify the server's response efficiently. In particular, the client and the server only need to communicate $O(n) = o(n^2)$ HE ciphertexts, which gives a nontrivial outsourced PIR.

Extension: By applying $\mathcal{T}$ to the publicly delegatable and verifiable schemes of [37], one would obtain schemes that are publicly delegatable and verifiable as well. These schemes would allow one to store f on a cloud server, and later, any client can freely retrieve a block of f on its own. Our schemes can be also used in other two-party protocols, such as oblivious polynomial evaluation and oblivious transfer [36], in order to obtain schemes against malicious parties.

6 Conclusion

In this paper, we proposed a transformation that adds privacy to a number of existing verifiable outsourcing schemes for the function family of multivariate polynomials over finite fields. The transformation is based on a noisy encoding of inputs and gives the first nearly practical verifiable computation scheme that has input (and output) privacy and does not limit the degree of the delegated polynomials. We also gave a verifiable

computation scheme for delegation of matrix-vector multiplication which has very efficient verification. We show an application of our schemes to the outsourcing of PIR. Applications of the schemes to other problems such as oblivious polynomial evaluation, and oblivious transfer, are interesting directions for future work.

A Security proof for the transformation $\mathcal{T}$

Let $f(x_1, \ldots, x_h)$ be any h-variate polynomial, and let k be a security parameter. Consider the security definition, Definition 2. Let $\mathcal{A}$ be any p.p.t. adversary attacking Γ, and let $\epsilon = \Pr[\mathsf{Exp}^{\mathsf{Ver}}_{\Gamma,\mathcal{A}}(f, k) = 1]$. We need to show that ϵ is a negligible function of k. This can be done by constructing a p.p.t adversary $\mathcal{B}$ that executes $\mathcal{A}$ as a subroutine and attacks Σ successfully at least with the same probability, i.e., $\Pr[\mathsf{Exp}^{\mathsf{Ver}}_{\Sigma,\mathcal{B}}(f, k) = 1] \geq \epsilon$. Given (f, k), the adversary $\mathcal{B}$ simply works as below:

- first of all, $\mathcal{B}$'s challenger computes $(\mathrm{pk}_f, \mathrm{sk}_f) \leftarrow \Sigma.\mathsf{KeyGen}(1^k, f)$ and then gives pk_f to $\mathcal{B}$;
- the adversary $\mathcal{B}$ invokes $\mathcal{A}$ with $\mathrm{PK}_f = \mathrm{pk}_f$;
- for $i = 1$ to $q = q(k)$, the adversaries $\mathcal{B}$, $\mathcal{A}$ and the challenger of $\mathcal{B}$ proceed as below:
  - based on its current view, i.e., $(\mathrm{PK}_f, \{\vec{\mathbf{a}}_j, \sigma_{\vec{\mathbf{a}}_j}, b_j\}_{j=1}^{i-1})$, the adversary $\mathcal{A}$ produces a new set
  $$\vec{\mathbf{a}}_i = (\mathbf{a}_{i,1}, \ldots, \mathbf{a}_{i,s}) \in (\mathbb{F}^h)^s$$
  of points and gives the set to $\mathcal{B}$;
  - the adversary $\mathcal{B}$ computes $(\mathsf{pk}_{\vec{\mathbf{a}}_i}, \mathsf{rk}_{\vec{\mathbf{a}}_i}) \leftarrow \mathsf{NEnc}(1^k, \vec{\mathbf{a}}_i)$ and parses $\mathsf{pk}_{\vec{\mathbf{a}}_i}$ as $\{\mathbf{c}_{i,j}\}_{j=1}^m$;
  - for $j = 1$ to m: the adversary $\mathcal{B}$ gives $\mathbf{c}_{i,j}$ to its challenger; the challenger computes
  $$(\sigma_{\mathbf{c}_{i,j}}, \tau_{\mathbf{c}_{i,j}}) \leftarrow \Sigma.\mathsf{ProbGen}(\mathrm{sk}_f, \mathbf{c}_{i,j})$$
  and gives $\sigma_{\mathbf{c}_{i,j}}$ to $\mathcal{B}$;
  - the adversary $\mathcal{B}$ gives $\sigma_{\vec{\mathbf{a}}_i} = \{\sigma_{\mathbf{c}_{i,j}}\}_{j=1}^m$ to the adversary $\mathcal{A}$;
  - based on its current view, i.e., $(\mathrm{PK}_f, \{\vec{\mathbf{a}}_j, \sigma_{\vec{\mathbf{a}}_j}, b_j\}_{j=1}^{i-1}, \vec{\mathbf{a}}_i, \sigma_{\vec{\mathbf{a}}_i})$, the adversary $\mathcal{A}$ produces $\{\hat{\sigma}_{f(\mathbf{c}_{i,j})}\}_{j=1}^m$ and gives it to $\mathcal{B}$;
  - for $j = 1$ to m: $\mathcal{B}$ gives $\hat{\sigma}_{f(\mathbf{c}_{i,j})}$ to its challenger; the challenger gives $\hat{v}_{i,j} \leftarrow \Sigma.\mathsf{Verify}(\mathrm{sk}_f, \tau_{\mathbf{c}_{i,j}}, \hat{\sigma}_{f(\mathbf{c}_{i,j})})$ to $\mathcal{B}$;
  - if there exists $j \in [m]$ such that $\hat{v}_{i,j} = \perp$, then $\mathcal{B}$ gives the bit $b_i = 0$ to $\mathcal{A}$; otherwise, $\mathcal{B}$ gives the bit $b_i = 1$ to $\mathcal{A}$.

Adversary $\mathcal{B}$ makes qm verification queries, i.e., $\{\hat{\sigma}_{f(\mathbf{c}_{i,j})}\}_{i\in[q],j\in[m]}$, to its challenger. The event $\mathsf{Exp}^{\mathsf{Ver}}_{\Sigma,\mathcal{B}}(f, k) = 1$ occurs if and only if there exist $i \in [q]$ and $j \in [m]$ such that $\hat{v}_{i,j} \notin \{f(\mathbf{c}_{i,j}), \perp\}$. The latter event occurs if there exists $i \in [q]$ such that (1) $b_i = 1$ and (2) there is an $\ell \in [s]$ such that $Q_{i,\ell}(0) \neq f(\mathbf{a}_{i,\ell})$, where $Q_{i,\ell}(z)$ is the polynomial interpolated from $\{\hat{v}_{i,j}\}_{j\in T_{i,\ell}}$ ($(T_{i,0}, T_{i,1}, \ldots, T_{i,s}) = \mathsf{rk}_{\vec{\mathbf{a}}_i}$ form a partition of $[m]$). We denote by $\mathbf{E}$ the last event. What $\mathcal{A}$ observes in the experiment above is exactly identical to what it should observe in the standard security experiment of Definition 2, i.e., $\mathsf{Exp}^{\mathsf{Ver}}_{\Gamma,\mathcal{A}}(f, k)$. Therefore,

$$\epsilon = \Pr[\mathsf{Exp}^{\mathsf{Ver}}_{\Gamma,\mathcal{A}}(f, k) = 1] = \Pr[\mathbf{E}] \leq \Pr[\mathsf{Exp}^{\mathsf{Ver}}_{\Sigma,\mathcal{B}}(f, k) = 1].$$

Due to the security of Σ under Definition 2, the function ϵ must be negligible in k, the security parameter.

Funding: This work was supported by National Natural Science Foundation of China (No. 61602304).

References

[1] P. Ananth, N. Chandran, V. Goyal, B. Kanukurthi and R. Ostrovsky, Achieving privacy in verifiable computation with multiple servers—without FHE and without pre-processing, in: *Public-key Cryptography—PKC 2014*, Lecture Notes in Comput. Sci. 8383, Springer, Heidelberg (2014), 149–166.

[2] B. Applebaum, Y. Ishai and E. Kushilevitz, From secrecy to soundness: Efficient verification via secure computation, in: *Automata, Languages and Programming—ICALP 2010*, Lecture Notes in Comput. Sci. 6198, Springer, Heidelberg (2010), 152–163.

[3] L. Babai, Trading group theory for randomness, in: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing—STOC'85*, ACM, New York (1985), 421–429.
[4] M. Barbosa and P. Farshim, Delegatable homomorphic encryption with applications to secure outsourcing of computation, in: *Topics in Cryptology—CT-RSA 2012*, Lecture Notes in Comput. Sci. 7178, Springer, Heidelberg (2012), 296–312.
[5] A. Beimel, Y. Ishai and T. Malkin, Reducing the servers computation in private information retrieval: PIR with preprocessing, in: *Advances in Cryptology—CRYPTO 2000*, Lecture Notes in Comput. Sci. 1880, Springer, Berlin (2000), 55–73.
[6] S. Benabbas, R. Gennaro and Y. Vahlis, Verifiable delegation of computation over large datasets, in: *Advances in Cryptology—CRYPTO 2011*, Lecture Notes in Comput. Sci. 6841, Springer, Heidelberg, (2011) 111–131.
[7] D. Bleichenbacher, A. Kiayias and M. Yung, Decoding of interleaved Reed Solomon codes over noisy data, in: *Automata, Languages and Programming*, Lecture Notes in Comput. Sci. 2719, Springer, Berlin (2003), 97–108.
[8] Z. Brakerski and V. Vaikuntanathan, Fully homomorphic encryption from ring-LWE and security for key dependent messages, in: *Advances in Cryptology—CRYPTO 2011*, Lecture Notes in Comput. Sci. 6841, Springer, Heidelberg (2011), 505–524.
[9] R. Canetti, B. Riva and G. Rothblum, Practical delegation of computation using multiple servers, in: *Proceedings of the 18th ACM Conference on Computer and Communications Security–CCS'11*, ACM, New York (2011), 445–454.
[10] D. Catalano, D. Fiore, R. Gennaro and K. Vamvourellis, Algebraic (trapdoor) one-way functions: Constructions and applications, in: *Theory of Cryptography—TCC 2013*, Lecture Notes in Comput. Sci. 7785, Springer, Heidelberg (2013), 680–699.
[11] S. G. Choi, J. Katz, R. Kumaresan and C. Cid, Multi-client non-interactive verifiable computation, in: *Theory of Cryptography—TCC 2013*, Lecture Notes in Comput. Sci. 7785, Springer, Heidelberg (2013), 499–518.
[12] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, Private information retrieval, in: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science—FOCS'95*, IEEE Press, Piscataway (1995), 41–50.
[13] K.-M. Chung, Y. Kalai and S. Vadhan, Improved delegation of computation using fully homomorphic encryption, in: *Advances in Cryptology—CRYPTO 2010*, Lecture Notes in Comput. Sci. 6223, Springer, Berlin (2010), 483–501.
[14] K.-M. Chung, Y. T. Kalai, F.-H. Liu and R. Raz, Memory delegation, in: *Advances in Cryptology—CRYPTO 2011*, Lecture Notes in Comput. Sci. 6841, Springer, Heidelberg (2011), 151–168.
[15] D. Coppersmith and M. Sudan, Reconstructing curves in three (and higher) dimensional space from noisy data, in: *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, ACM, New York (2003), 136–142.
[16] D. Fiore and R. Gennaro, Publicly verifiable delegation of large polynomials and matrix computations, with applications, in: *Proceedings of the 2012 ACM Conference on Computer and Communications Security—CCS 2012*, ACM, New York (2012), 501–512.
[17] D. Fiore, R. Gennaro and V. Pastro, Efficiently verifiable computation on encrypted data, in: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security—CCS'14*, ACM, New York (2014), 844–855.
[18] R. Gennaro, C. Gentry and B. Parno, Non-interactive verifiable computing: Outsourcing computation to untrusted workers, in: *Advances in Cryptology—CRYPTO 2010*, Lecture Notes in Comput. Sci. 6223, Springer, Berlin (2010), 465–482.
[19] O. Goldreich, R. Rubinfeld and M. Sudan, Learning polynomials with queries: The highly noisy case, in: *36th Annual Symposium on Foundations of Computer Science*, IEEE Press, Los Alamitos (1995), 294–303.
[20] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan and N. Zeldovich, How to run Turing machines on encrypted data, in: *Advances in Cryptology—CRYPTO 2013. Part II*, Lecture Notes in Comput. Sci. 8043, Springer, Heidelberg (2013), 536–553.
[21] S. Goldwasser, Y. T. Kalai and G. N. Rothblum, Delegating computation: interactive proofs for muggles, in: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing—STOC'08*, ACM, New York (2008), 113–122.
[22] S. Goldwasser, S. Micali and C. Rackoff, The knowledge complexity of interactive proof systems, *SIAM J. Comput.* **18** (1989), no. 1, 186–208.
[23] V. Guruswami and A. Rudra, Explicit capacity-achieving list-decodable codes, in: *Proceedings of the 38th Annual ACM symposium on Theory of Computing—STOC'06*, IEEE Press, Piscataway (2006), 1–10.
[24] V. Guruswami and M. Sudan, Improved decoding of Reed–Solomon and algebraic-geometry codes, *IEEE Trans. Inform. Theory* **45** (1999), no. 6, 1757–1767.
[25] Y. Huang and L. Goldberg, Outsourced Private Information Retrieval, in: *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society—WPES '13*, IEEE Press, Piscataway (2013), 119–130.
[26] Y. Ishai, E. Kushilevitz, R. Ostrovsky and A. Sahai, Cryptography from anonymity, in: *Proceedings of the 47th Annual Symposium on Foundations of Computer Science—FOCS'06*, IEEE Press, Piscataway (2006), 239–248.
[27] C. Joo and A. Yun, Homomorphic authenticated encryption secure against chosen-ciphertext attack, in: *Advances in Cryptology—ASIACRYPT 2014. Part II*, Lecture Notes in Comput. Sci. 8874, Springer, Heidelberg (2014), 173–192.
[28] A. Kiayias and M. Yung, Cryptographic hardness based on the decoding of Reed–Solomon codes, *IEEE Trans. Inform. Theory* **54** (2008), no. 6, 2752–2769.
[29] J. Kilian, A note on efficient zero-knowledge proofs and arguments, in: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing—STOC'92*, ACM, New York (1992), 723–732.
[30] J. Kilian, Improved efficient arguments, in: *Advances in Cryptology—CRYPT0'95*, Lecture Notes in Comput. Sci. 963, Springer, Heidelberg (1995), 311–324.

[31] E. Kushilevitz and R. Ostrovsky, Replication is not needed: Single database, computationally-private information retrieval, in: *Proceedings 38th Annual Symposium on Foundations of Computer Science—FOCS'97*, IEEE Press, Piscataway (1997), 364–373.
[32] B. Libert, T. Peters, M. Joye and M. Yung, Linearly homomorphic structure-preserving signatures and their applications, in: *Advances in Cryptology—CRYPTO 2013. Part II*, Lecture Notes in Comput. Sci. 8043, Springer, Heidelberg (2013), 289–307.
[33] F. J. MacWilliams and N. J. A. Sloane, *The theory of Error-correcting Codes. I*, North-Holland, Amsterdam, 1977.
[34] T. Mayberry, E.-O. Blass and A. H. Chan, PIRMAP: Efficient private information retrieval for mapreduce, in: *Financial Cryptography and Data Security—FC'13*, Lecture Notes in Comput. Sci. 7859, Springer, Heidelberg (2013), 371–385.
[35] S. Micali, CS proofs, in: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science—FOCS '94*, IEEE Press, Piscataway (1994), 436–453.
[36] M. Naor and B. Pinkas, Oblivious polynomial evaluation, *SIAM J. Comput.* **35** (2006), no. 5, 1254–1281.
[37] C. Papamanthou, E. Shi and R. Tamassia, Signatures of correct computation, in: *Theory of Cryptography—TCC 2013*, Lecture Notes in Comput. Sci. 7785, Springer, Heidelberg (2013), 222–242.
[38] B. Parno, J. Howell, C. Gentry and M. Raykova, Pinocchio: Nearly practical verifiable computation, in: *IEEE Symposium on Security and Privacy*, IEEE Press, Piscataway (2013), 238–25.
[39] B. Parno, M. Raykova and V. Vaikuntanathan, How to delegate and verify in public: Verifiable computation from attribute-based encryption, in: *Theory of Cryptography—TCC 2012*, Lecture Notes in Comput. Sci. 7194, Springer, Heidelberg (2012), 422–439.
[40] F. Parvaresh and A. Vardy, Correcting errors beyond the guruswami-sudan radius in polynomial time, in: *Proceedings of the 46th Annual Symposium on Foundations of Computer Science—FOCS'05*, IEEE Press, Piscataway (2005), 285–294.
[41] M. Sudan, Decoding of Reed Solomon codes beyond the error-correction bound, *J. Complexity* **13** (1997), no. 1, 180–193.
[42] T. Tassa, A. Jarrous and Y. Ben-Ya'akov, Oblivious evaluation of multivariate polynomials, *J. Math. Cryptol.* **7** (2013), no. 1, 1–29.
[43] D. Woodruff and S. Yekhanin, A geometric approach to information-theoretic private information retrieval, *SIAM J. Comput.* **37** (2007), no. 4, 1046–1056.