Verifiable Delegation of Computations with Storage-Verification Trade-off*

Liang Feng Zhang and Reihaneh Safavi-Naini

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

Abstract. Outsourcing computations has attracted much attention in recent years. An important security challenge is ensuring the correctness of the computed results. In the verifiable computation (VC) model of Gennaro, Gentry and Parno (CRYPTO 2010), a client can delegate the computation of its function to a cloud server, and efficiently verify the correctness of any computed results. In the existing VC schemes, the server must store an encoding of the function that doubles the required cloud storage, compared with storing the function itself. In this paper, we introduce a parameter that measures the trade-off between the required cloud storage and the client's verification time. We construct four (privately or publicly) VC schemes for delegating polynomials and matrices. These schemes allow the client to significantly reduce the consumed cloud storage by slightly increasing its verification time.

Keywords: verifiable computation, storage, verification, trade-off.

1 Introduction

Cloud computing allows resource-restricted clients to outsource (delegate) the storage of their data, and/or computations on the data, to cloud servers. The clients can access their data and request for computations on the outsourced data at their will. Outsourcing however, raises many security concerns such as the integrity of the stored data, and the delegated computations on the data. In this paper we are concerned with the latter.

The problem of verifiably outsourcing computation has been extensively studied in recent years, resulting in a number of models motivated by different application scenarios. In the verifiable computation (VC) model of Gennaro, Gentry and Parno [8], the client invests a one-time expensive computational effort to compute and store an encoding of its function with a cloud server, such that any evaluation of the function by the server, can be efficiently verified (using substantially less time than doing the evaluation). The one-time effort spent on encoding can be amortized over many evaluations of the function. Following [8], there has been a long list of papers on verifiable computation, both for generic functions [8,6,1,17,5] and for specific functions [2,7,16].

* This research is in part supported by Alberta Innovates Technology Futures.

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part I, LNCS 8712, pp. 112–129, 2014.

A VC scheme is called *privately verifiable*, if verification of the computed result can only be done by the client who has delegated the function; and *publicly verifiable*, if anyone with access to the computed result and possibly some public information can perform verification. To provide verifiability in a VC scheme, the encoding of the delegated function may include the delegated function itself, as well as some authentication information that will be used by the server to generate proofs. A common drawback of the existing VC schemes [8,6,1,17,5,2,7,16] is that, the encoding requires at least twice more cloud storage, compared with the delegated function. We define the *storage overhead* of a VC scheme as the ratio of the total cloud storage used by the encoding, to the cloud storage required for the delegated function itself. Less overhead is desirable because it means more efficient schemes. Under this definition, the existing VC schemes [8,6,1,17,5,2,7,16] have storage overhead ≥ 2.

1.1 Our Work

The emphasis of all existing VC schemes has been on efficient verification, and little attention has been paid to reducing the storage overhead. In practice a client may be willing to spend a bit more time on verification, if it can substantially reduce the consumed cloud storage. Note that the delegated function can be of tera byte size and so reducing the storage overhead can lead to substantial cost saving. This would be particularly attractive if the client does not need to compute the function very frequently and remains idle in between. In this paper, we investigate the trade-off between the consumed cloud storage and the client's verification time. We focus on the VC schemes of [2,7] for delegating polynomials and matrices (any matrix can define a function that takes a vector as input and outputs the vector-matrix multiplication). Both functions have important applications such as in verifiable keyword search, discrete Fourier transform, and linear transformations.

We introduce a *trade-off parameter* s, that measures the trade-off between the required cloud storage and the client's verification time. We break the delegated computation into s sub-computations, from which the result of the delegated computation can be reconstructed. In our setting, the cloud provides the s sub-computation results, along with *one* proof; the client verifies the correctness of the s results, and then computes the result of the delegated computation. The s subcomputations are the inner products of a common vector, with s vectors which are obtained from the delegated function. We authenticate the s vectors with a single tag vector, and thus obtain a saving of cloud storage (for tags).

We construct two VC schemes for delegating m-variate polynomials of degree $\leq d$ in each variable, a privately verifiable scheme, Π_1, and a publicly verifiable scheme, Π_2; and two VC schemes for delegating $m \times d$ matrices, a privately verifiable scheme, Π_3, and a publicly verifiable scheme, Π_4.

Compared with [2,7], our schemes have storage overhead $1 + 1/s < 2$. To achieve such a smaller storage overhead, the client's verification time in Π_1 and Π_2 is only slightly increased compared with [2] and [7], respectively. The scheme Π_3 is the first private VC scheme for delegating matrices. The scheme Π_4 is

more efficient than [7] not only in terms of storage overhead but also in terms of the client's verification time.

1.2 Background and Our Technique

For any finite set X, the notation "$x \leftarrow X$" means choosing an element x from X uniformly at random. Benabbas et al. [2] (Lemma 5.4) proposed a technique that allows a client to securely delegate the inner product computation of a fixed vector $\boldsymbol{f} = (f_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}} \in \mathbb{Z}_p^N$ with any vector $\boldsymbol{y} = (y_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}} \in \mathbb{Z}_p^N$, where p is a λ-bit prime and $\mathbb{I}$ is a finite ordered set of cardinality N. The client picks $\alpha \leftarrow \mathbb{Z}_p, \boldsymbol{r} = (r_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}} \leftarrow \mathbb{Z}_p^N$, and stores $f_{\boldsymbol{i}}$ and $t_{\boldsymbol{i}} = \alpha f_{\boldsymbol{i}} + r_{\boldsymbol{i}}$ with the server for every $\boldsymbol{i} \in \mathbb{I}$. In order for the client to learn the inner product $\rho = \boldsymbol{f} \cdot \boldsymbol{y}$, the server returns ρ and a proof $\pi = \boldsymbol{t} \cdot \boldsymbol{y}$. The client accepts ρ only if $\pi = \alpha\rho + \boldsymbol{r} \cdot \boldsymbol{y}$. A malicious server may try to deceive the client into accepting $\bar{\rho} \neq \rho$, using a forged proof $\bar{\pi}$ such that $\bar{\pi} = \alpha\bar{\rho} + \boldsymbol{r} \cdot \boldsymbol{y}$. A successful "try" implies (requires) the knowledge of $\alpha = (\bar{\rho} - \rho)^{-1}(\bar{\pi} - \pi)$, which is unknown to the server. Thus, the security is guaranteed. Inner product captures numerous widely used computations. For example, one can take $\mathbb{I} \subseteq \{0, 1, \ldots\}^m$, and consider $\boldsymbol{f}$ as the coefficients of a polynomial $f(\boldsymbol{x}) = \sum_{\boldsymbol{i}\in\mathbb{I}} f_{\boldsymbol{i}} \cdot \boldsymbol{x}^{\boldsymbol{i}}$, where $\boldsymbol{x}^{\boldsymbol{i}} = x_1^{i_1} \cdots x_m^{i_m}$ for $\boldsymbol{x} = (x_1, \ldots, x_m)$, and every $\boldsymbol{i} = (i_1, \ldots, i_m) \in \mathbb{I}$. Let $\boldsymbol{y} = (\boldsymbol{x}^{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}}$. Then the inner product $\boldsymbol{f} \cdot \boldsymbol{y}$ exactly represents the polynomial evaluation $f(\boldsymbol{x})$.

The technique however, has two drawbacks: (a1) The client must keep a secret key $(\alpha, \boldsymbol{r})$ which consumes more storage than $\boldsymbol{f}$; (a2) The verification requires the client to compute $\boldsymbol{r} \cdot \boldsymbol{y}$ which is as heavy as the delegated computation (i.e., $\boldsymbol{f} \cdot \boldsymbol{y}$). Benabbas et al. [2] constructed a VC scheme for delegating the polynomial $f(\boldsymbol{x})$, based on an adaption of this technique. Let $\mathbb{G}$ be a cyclic group of prime order $p \approx 2^\lambda$, generated by g. In their scheme, the client picks $\alpha \leftarrow \mathbb{Z}_p$ and a PRF $\mathsf{F}_k : \mathbb{I} \rightarrow \mathbb{G}$; it stores $f_{\boldsymbol{i}}$ and $t_{\boldsymbol{i}} = g^{\alpha f_{\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$ with the server for every $\boldsymbol{i} \in \mathbb{I}$. Given $\boldsymbol{x}$, the server returns $\rho = f(\boldsymbol{x})$ and a proof $\pi = \Pi_{\boldsymbol{i}\in\mathbb{I}}(t_{\boldsymbol{i}})^{\boldsymbol{x}^{\boldsymbol{i}}}$. The client believes that "$\rho = f(\boldsymbol{x})$" only if $\pi = g^{\alpha\rho} \cdot \tau$, where $\tau = \prod_{\boldsymbol{i}\in\mathbb{I}} \mathsf{F}_k(\boldsymbol{i})^{\boldsymbol{x}^{\boldsymbol{i}}}$. If we denote $\mathsf{F}_k(\boldsymbol{i}) = g^{r_{\boldsymbol{i}}}$ for every $\boldsymbol{i}$, then [2] actually uses their proposed technique, on the exponent of g. The client can keep (α, k) for verification and therefore avoid (a1). On the other hand, a critical observation of [2] is that one can choose F_k (*with closed-form efficiency*) such that computing τ requires substantially less time than computing ρ. This efficiency property of F_k removes (a2). The scheme of [2] has been extended to construct public VC schemes for delegating polynomials and matrices [7].

Our Technique. We propose a technique for securely delegating the inner product computations of s given vectors, $\boldsymbol{F}_1 = (F_{1,\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}}, \ldots, \boldsymbol{F}_s = (F_{s,\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}} \in \mathbb{Z}_p^N$, with any vector $\boldsymbol{y} = (y_{\boldsymbol{i}})_{\boldsymbol{i}\in\mathbb{I}} \in \mathbb{Z}_p^N$, where p is a λ-bit prime and, $\mathbb{I}$ is an ordered set of cardinality N. Note that the technique of [2] can delegate the s given vectors separately but require the server to store a separate vector of N tags for each of the s vectors. Our main observation is that the s inner product computations involve a *common* vector $\boldsymbol{y}$, and we can authenticate the s given vectors with a single vector of N tags such that the server can generate a single proof for

the s inner product computations. The details of our technique are as follows. The client picks $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_s) \in \mathbb{Z}_p^s$ and $\boldsymbol{r} = (r_{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{I}} \leftarrow \mathbb{Z}_p^N$, and computes the tag $t_{\boldsymbol{i}} = \alpha_1 F_{1,\boldsymbol{i}} + \cdots + \alpha_s F_{s,\boldsymbol{i}} + r_{\boldsymbol{i}}$ of the data blocks $(F_{1,\boldsymbol{i}}, F_{2,\boldsymbol{i}}, \ldots, F_{s,\boldsymbol{i}})$ for every $\boldsymbol{i} \in \mathbb{I}$. It then stores $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_s$, and a tag vector $\boldsymbol{t} = (t_{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{I}}$, with the server. In order for the client to learn $\rho_1 = \boldsymbol{y} \cdot \boldsymbol{F}_1, \ldots, \rho_s = \boldsymbol{y} \cdot \boldsymbol{F}_s$, the server returns $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_s)$ and a single proof $\pi = \boldsymbol{t} \cdot \boldsymbol{y}$; the client accepts $\boldsymbol{\rho}$ only if $\pi = \boldsymbol{\alpha} \cdot \boldsymbol{\rho} + \boldsymbol{r} \cdot \boldsymbol{y}$. As expected, the s vectors are authenticated using *one* tag vector $\boldsymbol{t}$ of size N. A malicious server may try to deceive the client into accepting $\bar{\boldsymbol{\rho}} \neq \boldsymbol{\rho}$ with a forged proof $\bar{\pi}$ such that $\bar{\pi} = \boldsymbol{\alpha} \cdot \bar{\boldsymbol{\rho}} + \boldsymbol{r} \cdot \boldsymbol{y}$. A successful "try" implies (requires) the knowledge of a nonzero vector $\boldsymbol{u} = (\pi - \bar{\pi}, \bar{\boldsymbol{\rho}} - \boldsymbol{\rho}) \in \mathbb{Z}_p^{N+1}$ such that $\boldsymbol{u} \cdot \boldsymbol{v} = 0$, where $\boldsymbol{v} = (1, \boldsymbol{\alpha}) \in \mathbb{Z}_p^{N+1}$. As $\boldsymbol{\alpha}$ is hidden from the server, it is intuitively hard for the malicious server to find $\boldsymbol{u}$. Our technical lemma (Lemma 1) in Section 2.2 shows that this intuition is true, even if the malicious server is allowed to make a polynomial (in λ) number of "tries", say $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_q$, and is told whether $\boldsymbol{u}_j \cdot \boldsymbol{v} = 0$ or not for every "try" $\boldsymbol{u}_j$. Our client can choose $\boldsymbol{\alpha}$ in two ways: (b1) pick $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^s$; (b2) pick $\alpha \leftarrow \mathbb{Z}_p$, and set $\boldsymbol{\alpha} = (\alpha, \alpha^2, \ldots, \alpha^s)$. In this paper we use our technique, together with the PRFs with closed-form efficiency, to delegate functions whose computations can be captured by the inner product computations of s given vectors, with a common (for all s vectors) vector. The s given vectors represent the delegated function, and the common vector is computed from an input to the delegated function. The functions we consider include polynomials and matrices.

1.3 Verifiable Delegation of Polynomials

Let $f(\boldsymbol{x}) = \sum_{\boldsymbol{i} \in \{0,1,\ldots,d\}^m} f_{\boldsymbol{i}} \cdot \boldsymbol{x}^{\boldsymbol{i}} \in \mathbb{Z}_p[\boldsymbol{x}]$ be an m-variate polynomial of degree $\leq d$ in each variable. Let s be the trade-off parameter, $n = (d+1)/s$ and $\mathbb{I} = \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, d\}^{m-1}$. Then

$$f(\boldsymbol{x}) = \sum_{\ell=1}^{s} x_1^{(\ell-1)n} \Big(\sum_{\boldsymbol{i} \in \mathbb{I}} f_{(\ell-1)n+i_1, i_2, \ldots, i_m} \cdot \boldsymbol{x}^{\boldsymbol{i}} \Big). \tag{1}$$

For every $\ell \in [s]$, let $\boldsymbol{F}_\ell = (F_{\ell,\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{I}} \in \mathbb{Z}_p^{|\mathbb{I}|}$, be a vector such that $F_{\ell,\boldsymbol{i}} = f_{(\ell-1)n+i_1, i_2, \ldots, i_m}$ for every $\boldsymbol{i} = (i_1, \ldots, i_m) \in \mathbb{I}$. Let $\boldsymbol{y} = (\boldsymbol{x}^{\boldsymbol{i}})_{\boldsymbol{i} \in \mathbb{I}}$. The equation (1) reduces the computation $f(\boldsymbol{x})$ to s inner product computations $\rho_1 = \boldsymbol{y} \cdot \boldsymbol{F}_1, \ldots, \rho_s = \boldsymbol{y} \cdot \boldsymbol{F}_s$ since $f(\boldsymbol{x}) = \sum_{\ell=1}^{s} \rho_\ell \cdot x_1^{(\ell-1)n}$. Thus, the verifiable delegation of $f(\boldsymbol{x})$ can be captured by our technique.

In this paper, we construct two schemes Π_1 and Π_2 for verifiably computing the s inner products and thus give two VC schemes for delegating f. In both schemes, the client stores $(F_{1,\boldsymbol{i}}, \ldots, F_{s,\boldsymbol{i}})$ and a corresponding tag $t_{\boldsymbol{i}}$ with the server for every $\boldsymbol{i} \in \mathbb{I}$. Given $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{Z}_p^m$, the server returns $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_s)$ and a proof $\pi = \Pi_{\boldsymbol{i} \in \mathbb{I}} (t_{\boldsymbol{i}})^{\boldsymbol{x}^{\boldsymbol{i}}}$. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p, and let $\mathsf{F}_k : \mathbb{I} \to \mathbb{G}$ be a PRF. In Π_1, the client picks $\alpha \leftarrow \mathbb{Z}_p$ and computes $t_{\boldsymbol{i}} = g^{\alpha F_{1,\boldsymbol{i}} + \cdots + \alpha^s F_{s,\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$, for every $\boldsymbol{i} \in \mathbb{I}$; it accepts $\boldsymbol{\rho}$ only if $\pi = g^{\boldsymbol{\alpha} \cdot \boldsymbol{\rho}} \cdot \tau$, where $\boldsymbol{\alpha} = (\alpha, \ldots, \alpha^s)$ and $\tau = \prod_{\boldsymbol{i} \in \mathbb{I}} \mathsf{F}_k(\boldsymbol{i})^{\boldsymbol{x}^{\boldsymbol{i}}}$. The scheme Π_2 uses a group $\mathbb{G}$

that admits a bilinear map e (see Section 2.3 for bilinear maps). The client picks $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_s) \leftarrow \mathbb{Z}_p^s$ and computes $t_{\boldsymbol{i}} = g^{\alpha_1 F_{1,\boldsymbol{i}} + \cdots + \alpha_s F_{s,\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$, for every $\boldsymbol{i} \in \mathbb{I}$. It accepts $\boldsymbol{\rho}$ only if $e(\pi, g) = e(g, g)^{\boldsymbol{\alpha} \cdot \boldsymbol{\rho}} \cdot \tau$, where $\tau = e(\prod_{\boldsymbol{i} \in \mathbb{I}} \mathsf{F}_k(\boldsymbol{i})^{\boldsymbol{x}^{\boldsymbol{i}}}, g)$. The scheme is publicly verifiable: $e(g, g)^{\alpha_1}, \ldots, e(g, g)^{\alpha_s}$ and τ are safely made public such that the verification can be done publicly. The F_k in both schemes is chosen such that computing τ requires substantially less time than computing $f(\boldsymbol{x})$. The security of Π_1 is based on the SDDH or DDH assumption for $\mathbb{G}$; the security of Π_2 is based on the DLIN assumption for $\mathbb{G}$.

1.4 Verifiable Delegation of Matrices

Let $\boldsymbol{E} = (E_{i,j})$ be an $m \times d$ matrix over $\mathbb{Z}_p$. Let s be the trade-off parameter, $n = d/s$ and $\mathbb{I} = [m] \times [n]$. We consider $\boldsymbol{E}$ as a block matrix $\boldsymbol{E} = (\boldsymbol{F}_1, \ldots, \boldsymbol{F}_s)$ where the block $\boldsymbol{F}_\ell$ consists of n columns of $\boldsymbol{E}$ numbered from $(\ell - 1)n + 1$ to ℓn for every $\ell \in [s]$. It is easy to see that $\boldsymbol{F}_\ell = (F_{\ell,\boldsymbol{i}})$ is an $m \times n$ matrix such that $F_{\ell,\boldsymbol{i}} = E_{i,(\ell-1)n+j}$ for every $\boldsymbol{i} = (i, j) \in \mathbb{I}$. Let $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{Z}_p^m$. For every $j \in [n]$, we have s inner product computations of $\boldsymbol{x}$ with the j-th columns of the block matrices $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_s$.

In this paper, we construct two schemes Π_3 and Π_4 for verifiably computing the s inner products for all $j \in [n]$ and thus give two VC schemes for delegating $\boldsymbol{E}$. In both schemes, the client stores $(F_{1,\boldsymbol{i}}, \ldots, F_{s,\boldsymbol{i}})$ and a corresponding tag $t_{\boldsymbol{i}}$ with the server for every $\boldsymbol{i} = (i, j) \in \mathbb{I}$. Given $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{Z}_p^m$, the server returns $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_d) = \boldsymbol{x} \cdot \boldsymbol{E}$ and n proofs $\{\pi_j = \Pi_{i=1}^m (t_{i,j})^{x_i}\}_{j=1}^n$, one for each set of s inner products. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order p and let $\mathsf{F}_k : \mathbb{I} \to \mathbb{G}$ be a PRF. In Π_3, the client picks $\alpha \leftarrow \mathbb{Z}_p$ and defines $t_{\boldsymbol{i}} = g^{\alpha F_{1,\boldsymbol{i}} + \alpha^2 F_{2,\boldsymbol{i}} + \cdots + \alpha^s F_{s,\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$ for every $\boldsymbol{i} = (i, j) \in \mathbb{I}$. The client accepts $\boldsymbol{\rho}$ only if $\pi_j = g^{\sum_{\ell=1}^s \rho_{j+(\ell-1)n} \cdot \alpha^\ell} \cdot \tau_j$ for every $j \in [n]$, where $\tau_j = \prod_{i=1}^m \mathsf{F}_k(i, j)^{x_i}$. The F_k is chosen such that computing $\{\tau_j\}_{j=1}^n$ requires substantially less time than computing $\boldsymbol{x} \cdot \boldsymbol{E}$. The scheme Π_4 is a public version of Π_3 using a group $\mathbb{G}$ that admits bilinear map. The security of Π_3 is based on the DDH assumption for $\mathbb{G}$; the security of Π_4 is based on the DLIN assumption for $\mathbb{G}$.

1.5 Performance Analysis and Extensions

We do performance analysis of the constructed schemes. Our analysis focuses on the trade-off between the required cloud storage, and the client's verification time. The storage overheads of our schemes are all equal to $1 + 1/s$, which is smaller than [2,7]. The clients in Π_1 and the scheme in [2], require around $((m + 2)\lambda + 4s)\lambda^2$ and $(m + 1)\lambda^3$ bit operations for each verification, respectively. Thus, Π_1 achieves significant saving of the cloud storage at the price of slightly increasing the client's verification time. The scheme Π_4 uses a bilinear map instance $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ (see Section 2.3). The client's verification time is dominated by $(m + n)$ exponentiations in $\mathbb{G}$, sn exponentiations in $\mathbb{G}_T$ and $2n$ pairing computations. Compared with Π_4, the client's verification time of [7] is dominated by $(m + sn)$ exponentiations in $\mathbb{G}$, sn exponentiations in $\mathbb{G}_T$, and $2sn$

pairing computations. Thus, our scheme Π_4 is more efficient not only in terms of storage overhead but also in terms of client's verification time. The performance analysis of Π_2 and Π_3 can be done similarly.

Our schemes for delegating $f(\boldsymbol{x})$ reduce $f(\boldsymbol{x})$ to the computations of s shorter polynomials, each taking one of the vectors $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_s$ as coefficients, and having degree $\leq n-1$ in x_1, and degree $\leq d$ in any other variable. We can *repeat this degree reduction* on $x_2, \ldots, x_m$ and thus reduce $f(\boldsymbol{x})$ to the computations of s^m shorter polynomials, each having degree $\leq n-1$, in any variable. Our technique can also be used to delegate multiple distinct functions, resulting in *batch verification*, which is more efficient than delegating the functions separately using the schemes of [2,7]. For example, when $m = 1$ we can treat the vectors $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_s$ in Π_1 as coefficients of s univariate polynomials; Π_1 allows us to efficiently verify the evaluations of the s polynomials at the same point, with a single proof from the server.

1.6 Related Work

The problem of verifiably outsourcing computation has a long history. We refer readers to [8,2] for a more detailed treatment of solutions that use strong assumptions on the adversary, and more theoretical solutions that use interaction. Here we are interested in non-interactive solutions in the standard model.

Verifiable Computation. The verifiable computation (VC) of Gennaro et al. [8] gave a non-interactive solution for verifiably outsourcing computation in the standard model. The VC schemes of [8,6,1] can delegate a function that is represented by a boolean circuit. They stay as mainly theoretical solutions because of using fully homomorphic encryption (FHE). The memory delegation of [5] can delegate computations on an arbitrary portion of the outsourced data. However, the client must be stateful and the solution suffer from the impracticality of PCP techniques. Benabbas et al. [2] initiated the study of practical (private) VC schemes for delegating specific functions such as polynomials. Parno et al. [17] initiated the study of public VC schemes. Fiore et al. [7] extended the constructions of [2] and obtained public VC schemes for delegating polynomials and matrices. Papamanthou et al. [16] constructed a public VC scheme for delegating polynomials that allows efficient update. The storage overhead of all these schemes is ≥ 2.

Homomorphic MACs and Signatures. A homomorphic MAC or signature scheme [10,4] allows one to freely authenticate data and then verify computations on the authenticated data. Such schemes result in VC: the client stores data blocks and their MAC tags (or signatures) with a server; the server computes some admissible functions on an arbitrary subset of the data blocks; the server provides both the result and a MAC tag (or signature) vouching for the correctness of the result. The storage overhead of the resulting VC is ≥ 2.

Non-interactive Proofs. Goldwasser et al. [12] gave a non-interactive scheme for delegating NC computations. However, for any circuit of size n, the server's running time is a high degree polynomial in n, and so not practical. The SNARGs

or SNARKs [3,9], gives a non-interactive scheme for delegating computation. However, they rely on non-falsifiable assumptions [11] which are much stronger than the common assumptions (such as DDH), used in this paper.

Proofs of Retrievability. PoR [13,18] allows a client to store a file with a server, and then efficiently check the file's integrity. Homomorphic linear authenticators that are similar to our authenticators, have been used in [18] but not formally proved as an authentication system. We give a formal proof here. Other differences with [18] are as follows: firstly, the $\boldsymbol{\alpha}$ in [18] was chosen using (b1); we can use (b2) and thus have a much shorter secret key; secondly, they cannot use PRFs with closed-form efficiency while we can combine such PRFs with our technique to give VC schemes.

Organization. In Section 2 we recall the notions of VC and PRFs with closed-form efficiency; In Section 3 we present the schemes Π_1-Π_4; Section 4 contains some concluding remarks.

2 Preliminaries

Let λ be a security parameter. We denote by "$\mathsf{poly}(\lambda)$" and "$\mathsf{neg}(\lambda)$" the classes of polynomial functions and negligible functions in λ, respectively. Let $\mathcal{A}(\cdot)$ be a probabilistic polynomial time (PPT) algorithm. The notation "$y \leftarrow \mathcal{A}(x)$" means that y is the output of $\mathcal{A}$ on input x.

2.1 Verifiable Computation

A verifiable computation (VC) scheme is a tuple $\Pi = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ of four algorithms, where

- $(ek, dk, vk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$ is a key generation algorithm that takes as input the security parameter λ and a function f and outputs an evaluation key ek, a delegation key dk and a verification key vk;
- $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(dk, x)$ is a problem generation algorithm that takes as input dk and any input x in the domain of f and outputs an encoding σ of x and some auxiliary information τ for verification;
- $(\rho, \pi) \leftarrow \mathsf{Compute}(ek, \sigma)$ is a computation algorithm that takes as input ek and σ and outputs an answer ρ and a proof π; and
- $\{f(x), \perp\} \leftarrow \mathsf{Verify}(vk, \tau, \rho, \pi)$ is a verification algorithm that verifies ρ with (vk, τ, π); it outputs $f(x)$ or $\perp$ (to indicate failure).

Model. The VC model involves a client $\mathcal{C}$ and a server $\mathcal{S}$, where $\mathcal{C}$ has a function f. The client $\mathcal{C}$ picks $(ek, dk, vk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$ and gives ek to the server. In order to learn $f(x)$, the client computes $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(dk, x)$ and gives σ to the server. Given (ek, σ), the server computes and replies with $(\rho, \pi) \leftarrow \mathsf{Compute}(ek, \sigma)$. At last, the client runs $\mathsf{Verify}(vk, \tau, \rho, \pi)$ to verify ρ and recover $f(x)$. The verification of ρ requires vk, τ and π. The scheme Π is called *privately verifiable* if (vk, τ) must be kept private by $\mathcal{C}$, and *publicly verifiable* if (vk, τ) can be made public (in particular, it can be known to $\mathcal{S}$).

Correctness. This property requires that the client can always learn the correct result of the delegated computation when the server is honest. Formally, the scheme Π is *correct* if for any function f, any $(ek, dk, vk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$, any x in the domain of f, any $(\sigma, \tau) \leftarrow \mathsf{ProbGen}(dk, x)$ and any $(\rho, \pi) \leftarrow \mathsf{Compute}(ek, \sigma)$, it holds that $\mathsf{Verify}(vk, \tau, \rho, \pi) = f(x)$.

Security. This property requires that no malicious server can cause the client to compute an incorrect result of the delegated computation. Formally, the scheme Π is said to be *secure* if any PPT adversary $\mathcal{A}$ wins with probability $< \mathsf{neg}(\lambda)$ in the security game of **Fig.** 1.

- Setup. Given f, the challenger picks $(ek, dk, vk) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$. If Π is privately verifiable, it gives ek to $\mathcal{A}$ and keeps (dk, vk); if Π is publicly verifiable, it gives (ek, vk) to $\mathcal{A}$ and keeps dk.
- Queries. The adversary $\mathcal{A}$ adaptively makes a polynomial number of queries: for every $j = 1$ to $q = \mathsf{poly}(\lambda)$,
  - $\mathcal{A}$ picks x_j from the domain of f and gives it to the challenger;
  - The challenger computes $(\sigma_j, \tau_j) \leftarrow \mathsf{ProbGen}(dk, x_j)$. If Π is privately verifiable, it gives σ_j to $\mathcal{A}$; if Π is publicly verifiable, it gives (σ_j, τ_j) to $\mathcal{A}$.
  - $\mathcal{A}$ picks a response $(\bar{\rho}_j, \bar{\pi}_j)$ to the challenger;
  - The challenger gives the output of $\mathsf{Verify}(vk, \tau_j, \bar{\rho}_j, \bar{\pi}_j)$ to $\mathcal{A}$.
- Forgery. $\mathcal{A}$ picks x^* from the domain of f. The challenger computes $(\sigma^*, \tau^*) \leftarrow \mathsf{ProbGen}(dk, x^*)$. If Π is privately verifiable, the challenger gives σ^* to $\mathcal{A}$; if Π is publicly verifiable, the challenger gives (σ^*, τ^*) to $\mathcal{A}$. At last, $\mathcal{A}$ picks $(\bar{\rho}^*, \bar{\pi}^*)$.
- Output. The adversary *wins* if $\mathsf{Verify}(sk, \tau^*, \bar{\rho}^*, \bar{\pi}^*) \notin \{f(x^*), \perp\}$.

Fig. 1. Security game

Remark. In Forgery $\mathcal{A}$ behaves just like it has done in any one of the q queries. Without loss of generality, we can suppose $(x^*, \bar{\rho}^*, \bar{\pi}^*) = (x_c, \bar{\rho}_c, \bar{\pi}_c)$ for some $c \in [q]$, i.e., $\mathcal{A}$ picks one of its q queries as forgery.

2.2 A Technical Lemma

In this section we study a problem that underlies the security of our technique in Section 1.1 (and the security proofs of the privately verifiable schemes Π_1 and Π_3). Let λ be a security parameter. Let p be a λ-bit prime and let $\mathbb{F}_p$ be the finite field of p elements. Let $s > 0$. We define an equivalence relation $\sim$ over $\mathbb{F}_p^s \setminus \{0\}$ as below: two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_p^s \setminus \{0\}$ are *equivalent* if there exists $\xi \in \mathbb{F}_p \setminus \{0\}$ such that $\boldsymbol{u} = \xi \cdot \boldsymbol{v}$. Let $\Omega_{p,s} = (\mathbb{F}_p^s \setminus \{0\})/\sim$ be the set of all equivalent classes. We represent each equivalent class with a vector in that class. Without loss of generality, we agree that the representative of each class in $\Omega_{p,s}$ is chosen such that its first non-zero element is 1. For example, when $p = 3$ and $s = 2$, we have that $\Omega_{p,s} = \{(0,1), (1,0), (1,1), (1,2)\}$. For any $\boldsymbol{u}, \boldsymbol{v} \in \Omega_{p,s}$, we define $\boldsymbol{u} \odot \boldsymbol{v} = 0$ if the inner product of $\boldsymbol{u}$ and $\boldsymbol{v}$ is 0 and define $\boldsymbol{u} \odot \boldsymbol{v} = 1$ otherwise. For example, when $p = 3$ and $s = 2$, we have that $(1,1) \odot (1,2) = 0$ and $(1,1) \odot (1,1) = 1$. The following problem models the malicious server's attack in our technique.

Problem 1. Let $\mathcal{A}$ be any algorithm. Let $U, V \subseteq \Omega_{p,s+1}$ and let $q = \mathsf{poly}(\lambda)$. In this problem, a vector $\boldsymbol{v}^* \leftarrow V$ is chosen and hidden from $\mathcal{A}$; for $i = 1$ to q, $\mathcal{A}$ adaptively picks a query $\boldsymbol{u}_i \in U$ and learns $b_i = \boldsymbol{u}_i \odot \boldsymbol{v}^* \in \{0, 1\}$; $\mathcal{A}$ *wins* only if there exists an $i^* \in [q]$ such that $b_{i^*} = 0$.

Lemma 1. *Suppose that* $0 < \epsilon < 1$. *If* $|\{\boldsymbol{v} \in V : \boldsymbol{u} \odot \boldsymbol{v} = 0\}| \leq \epsilon \cdot |V|$ *for every* $\boldsymbol{u} \in U$, *then* $\mathcal{A}$ *wins in* **Problem 1** *with probability* $\leq \epsilon q$.

Proof. For every $i \in [q]$, let $\mathbf{S}_i$ be the event that $b_i = 0$ and let $\neg\mathbf{S}_i$ be the event that $b_i = 1$. We denote $\mathbf{F}_i = \neg\mathbf{S}_i \wedge \cdots \wedge \neg\mathbf{S}_1$ for every $i \in [q]$. Clearly, the probabilities of $\mathbf{S}_i, \neg\mathbf{S}_i$ and $\mathbf{F}_i$ are all taken over the uniform choice of $\boldsymbol{v}^*$ and the adversarial choices of $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_i$ by $\mathcal{A}$. The probability that $\mathcal{A}$ wins in **Problem 1** is bounded by

$$\epsilon^* = \Pr[\mathbf{S}_1] + \sum_{i=2}^{q} \Pr[\mathbf{S}_i \wedge \mathbf{F}_{i-1}]. \tag{2}$$

Note that $|\{\boldsymbol{v} \in V : \boldsymbol{u}_1 \odot \boldsymbol{v} = 0\}| \leq \epsilon|V|$ for any $\boldsymbol{u}_1 \in U$, we must have that $\Pr[\mathbf{S}_1] = \Pr[\boldsymbol{u}_1 \odot \boldsymbol{v}^* = 0] = \frac{|\{\boldsymbol{v} \in V : \boldsymbol{u}_1 \odot \boldsymbol{v} = 0\}|}{|V|} \leq \epsilon$. If $\mathbf{F}_1$ occurs, then $\mathcal{A}$ learns that $\boldsymbol{v}^* \notin \{\boldsymbol{v} \in V : \boldsymbol{u}_1 \odot \boldsymbol{v} = 0\}$, which allows $\mathcal{A}$ to rule out at most $\epsilon \cdot |V|$ possibilities of $\boldsymbol{v}^*$. Conditioned on $\mathbf{F}_1$, the $\boldsymbol{v}^*$ will be uniformly distributed over the set $V_1 = \{\boldsymbol{v} \in V : \boldsymbol{u}_1 \odot \boldsymbol{v} = 1\}$. Note that $|\{\boldsymbol{v} \in V_1 : \boldsymbol{u}_2 \odot \boldsymbol{v} = 0\}| \leq |\{\boldsymbol{v} \in V : \boldsymbol{u}_2 \odot \boldsymbol{v} = 0\}| \leq \epsilon|V|$ for any $\boldsymbol{u}_2 \in U$. Thus, $\Pr[\mathbf{S}_2 \wedge \mathbf{F}_1] = \Pr[\boldsymbol{u}_2 \odot \boldsymbol{v}^* = 0|\mathbf{F}_1] \cdot \Pr[\mathbf{F}_1] = \frac{|\{\boldsymbol{v} \in V_1 : \boldsymbol{u}_2 \odot \boldsymbol{v} = 0\}|}{|V_1|} \frac{|V_1|}{|V|} \leq \epsilon$, where the second equality follows from the fact that $\boldsymbol{v}^*$ is uniformly distributed over V_1 (conditioned on $\mathbf{F}_1$). In general, for any $i \in [q]$ we can bound the probability $\epsilon_i = \Pr[\mathbf{S}_i \wedge \mathbf{F}_{i-1}]$ that $\boldsymbol{u}_i$ is the first query such that $\boldsymbol{u}_i \odot \boldsymbol{v}^* = 0$. Let $V_{i-1} = \{\boldsymbol{v} \in V : \boldsymbol{u}_1 \odot \boldsymbol{v} = 1, \cdots, \boldsymbol{u}_{i-1} \odot \boldsymbol{v} = 1\}$. Conditioned on $\mathbf{F}_{i-1}$, $\boldsymbol{v}^*$ must be uniformly distributed over V_{i-1}. Note that $|\{\boldsymbol{v} \in V_{i-1} : \boldsymbol{u}_i \odot \boldsymbol{v} = 0\}| \leq \epsilon|V|$ for any $\boldsymbol{u}_i \in U$. We have

$$\begin{aligned}\epsilon_i &= \Pr[\mathbf{S}_i|\mathbf{F}_{i-1}] \Pr[\mathbf{F}_{i-1}] = \Pr[\boldsymbol{u}_i \odot \boldsymbol{v}^* = 0|\mathbf{F}_{i-1}] \prod_{h=2}^{i-1} \Pr[\neg\mathbf{S}_h|\mathbf{F}_{h-1}] \Pr[\mathbf{F}_1] \\ &= \frac{|\{\boldsymbol{v} \in V_{i-1} : \boldsymbol{u}_i \odot \boldsymbol{v} = 0\}|}{|V_{i-1}|} \frac{|V_{i-1}|}{|V_{i-2}|} \cdots \frac{|V_2|}{|V_1|} \frac{|V_1|}{|V|} \leq \epsilon,\end{aligned}$$

where $\Pr[\neg\mathbf{S}_h|\mathbf{F}_{h-1}]$ is the probability that the uniform random variable $\boldsymbol{v}^*$ (over V_{h-1}) falls into $V_h \subseteq V_{h-1}$ for every $h \in [i-1]$. Hence, we have $\epsilon^* \leq \epsilon q$ as each summand on the right hand side of (2) is $\leq \epsilon$. □

Example 1. Let $V_{\text{lin}} = \{(1, \boldsymbol{w}) : \boldsymbol{w} \in \mathbb{F}_p^s\} \subseteq \Omega_{p,s+1}$ and $U \subseteq \Omega_{p,s+1}$. It is easy to see that $|V_{\text{lin}}| = p^s$. For any $\boldsymbol{u} \in U$, there are $\leq N = (p^s - 1)/(p-1)$ elements $\boldsymbol{v} \in V_{\text{lin}}$ such that $\boldsymbol{u} \odot \boldsymbol{v} = 0$. Thus, $\epsilon \leq N/|V_{\text{lin}}| < 1/(p-1)$.

Example 2. Let $V_{\text{pol}} = \{(1, \alpha, \ldots, \alpha^s) : \alpha \in \mathbb{F}_p\} \subseteq \Omega_{p,s+1}$ and $U \subseteq \Omega_{p,s+1}$. Let $\boldsymbol{u} = (u_0, u_1, \ldots, u_s) \in U$ and $\boldsymbol{v} = (1, \alpha, \ldots, \alpha^s) \in V_{\text{pol}}$. Then $\boldsymbol{u} \odot \boldsymbol{v} = 0$ only if α is a root of $u_0 + u_1 x + \cdots + u_s x^s$. For any $\boldsymbol{u} \in U$, there are $\leq s$ elements $\boldsymbol{v} \in V_{\text{pol}}$

such that $\boldsymbol{u} \odot \boldsymbol{v} = 0$ because a univariate polynomial of degree s has $\leq s$ roots in $\mathbb{F}_p$. Thus, $\epsilon \leq s/p$ in this case.

The two examples provide us with two ways of choosing the $\boldsymbol{\alpha}$ in our technique: (b1) pick $\boldsymbol{\alpha} \leftarrow \mathbb{F}_p^s$; and (b2) pick $\alpha \leftarrow \mathbb{F}_p$ and define $\boldsymbol{\alpha} = (\alpha, \alpha^2, \ldots, \alpha^s)$. We use (b2) in Π_1 and Π_3 such that a short secret key suffices to do verification.

2.3 Cryptographic Assumptions

We assume a *group scheme* $\mathcal{G}(1^\lambda, \ell)$ that takes as input the security parameter λ and an integer $\ell \in \{1, 2\}$ and outputs a *random group instance* Λ. When $\ell = 1$, Λ is a triple $(p, \mathbb{G}, g)$, where $\mathbb{G} = \langle g \rangle$ is a group of prime order $p \approx 2^\lambda$; when $\ell = 2$, Λ is a quintuple $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $\mathbb{G} = \langle g \rangle$ and $\mathbb{G}_T$ are groups of prime order $p \approx 2^\lambda$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently computable non-degenerated bilinear map. In Section 2.4 we present five PRFs. The security of each PRF is based on one of the following assumptions for $\mathbb{G}$: the *d-strong decision Diffie-Hellman* assumption (d-SDDH); the *decision Diffie-Hellman* assumption (DDH); and the *decision linear* assumption (DLIN). We refer the readers to [2,7] for their definitions. There is also a much weaker assumption for $\mathbb{G}$: the hardness of *computational Diffie-Hellman* (CDH) problem. In CDH, one is given (g, g^α, h) and must output h^α, where $\alpha \leftarrow \mathbb{Z}_p$ and $h \leftarrow \mathbb{G}$. The hardness of CDH says that no PPT algorithm can output h^α except with probability $< \mathsf{neg}(\lambda)$.

2.4 PRFs with Closed-Form Efficiency

In this section, we review the notion of PRFs with closed-form efficiency and present several such PRFs for our VC schemes. A PRF is a pair $\Sigma = (\mathsf{Kg}, \mathsf{F})$ of algorithms. The key generation algorithm $\mathsf{Kg}(1^\lambda, \mathsf{params})$ takes as input a security parameter λ and some additional parameters params and outputs a secret key k and a public parameter pp, where pp specifies the domain $\mathbb{I}$ and range $\mathbb{Y}$ of F_k. Given any $\boldsymbol{i} \in \mathbb{I}$, $\mathsf{F}_k(\boldsymbol{i})$ outputs a value $y \in \mathbb{Y}$. Σ is called *pseudorandom* if for any PPT algorithm $\mathcal{A}$, we have $|\Pr[\mathcal{A}^{\mathsf{F}_k(\cdot)}(1^\lambda, \mathsf{pp}) = 1] - \Pr[\mathcal{A}^{\Phi(\cdot)}(1^\lambda, \mathsf{pp}) = 1]| < \mathsf{neg}(\lambda)$, where the probabilities are taken over $(\mathsf{pp}, k) \leftarrow \mathsf{Kg}(1^\lambda, \mathsf{params})$, the choice of a random function $\Phi : \mathbb{I} \to \mathbb{Y}$ and $\mathcal{A}$'s random coins. Let $C(\boldsymbol{y}, \boldsymbol{x})$ be any computation that takes $\boldsymbol{y} = \{y_{\boldsymbol{i}}\}_{\boldsymbol{i} \in \mathbb{I}} \in \mathbb{Y}^{|\mathbb{I}|}$ and some $\boldsymbol{x}$ as input. Suppose that computing $C(\boldsymbol{y}, \boldsymbol{x})$ for general $(\boldsymbol{y}, \boldsymbol{x})$ requires time t. We say that Σ has *closed-form efficiency for* C if there is an algorithm $\Sigma.\mathsf{CFE}$ such that $\Sigma.\mathsf{CFE}(k, \mathbb{I}, \boldsymbol{x}) = C(\{\mathsf{F}_k(\boldsymbol{i})\}_{\boldsymbol{i} \in \mathbb{I}}, \boldsymbol{x})$ but only requires time $o(t)$.

Construction 1. Let $m > 0$ and $d + 1 = sn$. Our first PRF $\Sigma_1 = (\mathsf{Kg}, \mathsf{F})$ is tailored from the $\mathcal{PRF}_{2,d}$ in [2]. The algorithm $\mathsf{Kg}(1^\lambda, (m, s, n))$ picks $\Lambda = (p, \mathbb{G}, g) \leftarrow \mathcal{G}(1^\lambda, 1)$, $k = (k_0, k_1, \ldots, k_m) \leftarrow \mathbb{Z}_p^{m+1}$ and outputs k and $\mathsf{pp} = (\Lambda, m, s, n)$. The domain of F_k is $\mathbb{I} = \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, d\}^{m-1}$; the range of F_k is $\mathbb{G}$. For any $\boldsymbol{i} = (i_1, \ldots, i_m) \in \mathbb{I}$, $\mathsf{F}_k(\boldsymbol{i}) = g^{k_0 k_1^{i_1} \cdots k_m^{i_m}}$. In fact, Σ_1 is the restriction of $\mathcal{PRF}_{2,d}$ on $\mathbb{I}$. Theorem 1 of [2] shows that Σ_1 is a PRF under the d-SDDH assumption. For $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{Z}_p^m$ and $\boldsymbol{y} = \{y_{\boldsymbol{i}}\}_{\boldsymbol{i} \in \mathbb{I}} \in \mathbb{G}^{|\mathbb{I}|}$, let $C'(\boldsymbol{y}, \boldsymbol{x}) = \prod_{\boldsymbol{i} \in \mathbb{I}} (y_{\boldsymbol{i}})^{\boldsymbol{x}^{\boldsymbol{i}}}$. Then we have $C'(\{\mathsf{F}_k(\boldsymbol{i})\}_{\boldsymbol{i} \in \mathbb{I}}, \boldsymbol{x}) = g^\xi$, where

$\xi = k_0(1-(k_1x_1)^n)/(1-k_1x_1)\cdot\prod_{j=2}^{m}(1-(k_jx_j)^{d+1})/(1-k_jx_j)$. Without k, computing $C'(\{\mathsf{F}_k(\boldsymbol{i})\}_{\boldsymbol{i}\in\mathbb{I}},\boldsymbol{x})$ requires $O(|\mathbb{I}|)$ operations. Given k, the $\Sigma_1.\mathsf{CFE}$ can compute ξ and then g^{ξ} using $O(m)=o(|\mathbb{I}|)$ operations. Thus, Σ_1 has closed-form efficiency for C'.

Construction 2. Let $m>0$ and $d+1=2^a=sn=s\cdot 2^b$. Below is our instantiation $\Sigma_2=(\mathsf{Kg},\mathsf{F})$ of the Naor-Reingold PRF (Section 4.1, [15]).

$\mathsf{Kg}(1^\lambda,(m,a,b))$: Picks $\Lambda=(p,\mathbb{G},g)\leftarrow\mathcal{G}(1^\lambda,1)$. Picks $k_0\leftarrow\mathbb{Z}_p$, $k_{1,w}\leftarrow\mathbb{Z}_p$ for every $w\in[b]$ and $k_{u,v}\leftarrow\mathbb{Z}_p$ for every $(u,v)\in\{2,\ldots,m\}\times[a]$. Outputs $k=\{k_0\}\cup\{k_{1,w}:w\in[b]\}\cup\{k_{u,v}:(u,v)\in\{2,\ldots,m\}\times[a]\}$ and $\mathsf{pp}=(\Lambda,m,a,b)$. The domain of F_k is $\mathbb{I}=\{0,1,\ldots,n-1\}\times\{0,1,\ldots,d\}^{m-1}$. The range of F_k is $\mathbb{G}$.

$\mathsf{F}_k(\cdot)$: Given $\boldsymbol{i}=(i_1,\ldots,i_m)\in\mathbb{I}$, computes the binary representations of $i_1,\ldots,i_m$, say $i_1=(i_{1,1},\ldots,i_{1,b})$ and $i_u=(i_{u,1},\ldots,i_{u,a})$ for every $2\le u\le m$. It outputs $\mathsf{F}_k(\boldsymbol{i})=g^{\xi_{\boldsymbol{i}}}$, where $\xi_{\boldsymbol{i}}=k_0\cdot\prod_{w=1}^{b}k_{1,w}^{i_{1,w}}\cdot\prod_{u=2}^{m}\prod_{v=1}^{a}k_{u,v}^{i_{u,v}}$.

As a Naor-Reingold PRF defined over $\{0,1\}^{b+(m-1)a}$, Σ_2 is pseudorandom under DDH. Note that $C'(\{\mathsf{F}_k(\boldsymbol{i})\}_{\boldsymbol{i}\in\mathbb{I}},\boldsymbol{x})=\prod_{\boldsymbol{i}\in\mathbb{I}}\mathsf{F}_k(\boldsymbol{i})^{\boldsymbol{x}^{\boldsymbol{i}}}=g^{\xi}$, where

$$\xi=\sum_{\boldsymbol{i}\in\mathbb{I}}\xi_{\boldsymbol{i}}\cdot\boldsymbol{x}^{\boldsymbol{i}}=\sum_{i_1=0}^{2^b-1}\sum_{i_2=0}^{2^a-1}\cdots\sum_{i_m=0}^{2^a-1}k_0\cdot\prod_{w=1}^{b}k_{1,w}^{i_{1,w}}x_1^{i_{1,w}\cdot 2^{w-1}}\cdot\prod_{u=2}^{m}\prod_{v=1}^{a}k_{u,v}^{i_{u,v}}\cdot x_u^{i_{u,v}\cdot 2^{v-1}}$$
$$=k_0\cdot\prod_{w=1}^{b}\left(1+k_{1,w}x_1^{2^{w-1}}\right)\cdot\prod_{u=2}^{m}\prod_{v=1}^{a}\left(1+k_{u,v}x_u^{2^{v-1}}\right).$$

Computing $C'(\{\mathsf{F}_k(\boldsymbol{i})\}_{\boldsymbol{i}\in\mathbb{I}},\boldsymbol{x})$ without k requires $O(|\mathbb{I}|)$ operations. Given k, the $\Sigma_2.\mathsf{CFE}$ can compute ξ and then g^{ξ} using $O(ma)=o(|\mathbb{I}|)$ operations. Thus, Σ_2 has the closed form efficiency for C'.

Construction 3. Let $m>0$ and $d+1=2^a=sn=s\cdot 2^b$. Below is an instantiation $\Sigma_3=(\mathsf{Kg},\mathsf{F})$ of the Lewko-Waters PRF (Section 3.1, [14]).

$\mathsf{Kg}(1^\lambda,(m,a,b))$: Picks $\Lambda=(p,\mathbb{G},\mathbb{G}_T,g)\leftarrow\mathcal{G}(1^\lambda,2)$, $k_0,l_0\leftarrow\mathbb{Z}_p$, a 2×2 matrix $K_{1,w}\leftarrow\mathbb{Z}_p^{2\times 2}$ for every $w\in[b]$, and a 2×2 matrix $K_{u,v}\leftarrow\mathbb{Z}_p^{2\times 2}$ for every $(u,v)\in\{2,\ldots,m\}\times[a]$. Outputs $k=\{k_0,l_0\}\cup\{K_{1,w}:w\in[b]\}\cup\{K_{u,v}:(u,v)\in\{2,\ldots,m\}\times[a]\}$ and $\mathsf{pp}=(\Lambda,m,a,b)$. The domain of F_k is $\mathbb{I}=\{0,1,\ldots,n-1\}\times\{0,1,\ldots,d\}^{m-1}$; the range of F_k is $\mathbb{G}$.

$\mathsf{F}_k(\cdot)$: Given $\boldsymbol{i}=(i_1,\ldots,i_m)\in\mathbb{I}$, computes the binary representations of $i_1,\ldots,i_m$, say $i_1=(i_{1,1},\ldots,i_{1,b})$ and $i_u=(i_{u,1},\ldots,i_{u,a})$ for every $2\le u\le m$. It outputs $\mathsf{F}_k(\boldsymbol{i})=g^{\xi_{\boldsymbol{i}}}$, where $(\xi_{\boldsymbol{i}},\eta_{\boldsymbol{i}})=(k_0,l_0)\prod_{w=1}^{b}K_{1,w}^{i_{1,w}}\prod_{u=2}^{m}\prod_{v=1}^{a}K_{u,v}^{i_{u,v}}$.

As a Lewko-Waters PRF defined over $\{0,1\}^{b+(m-1)a}$, Σ_3 is pseudorandom under DLIN. Note that $C'(\{\mathsf{F}_k(\boldsymbol{i})\}_{\boldsymbol{i}\in\mathbb{I}},\boldsymbol{x})=g^{\xi}$ with $\xi=\sum_{\boldsymbol{i}\in\mathbb{I}}\xi_{\boldsymbol{i}}\cdot\boldsymbol{x}^{\boldsymbol{i}}$. As in construction 2, we can similarly show that Σ_3 has the closed form efficiency for C'.

Construction 4. Let $m,n>0$. Below is a DDH based PRF $\Sigma_4=(\mathsf{Kg},\mathsf{F})$.

$\mathsf{Kg}(1^\lambda,(m,n))$: Picks $\Lambda=(p,\mathbb{G},g)\leftarrow\mathcal{G}(1^\lambda,1)$, $u_i\leftarrow\mathbb{G}$ for every $i\in[m]$, $k_j\leftarrow\mathbb{Z}_p$ for every $j\in[n]$; outputs $k=\{u_i\}_{i=1}^{m}\cup\{k_j\}_{j=1}^{n}$ and $\mathsf{pp}=(\Lambda,m,n)$. The domain of F_k is $\mathbb{I}=[m]\times[n]$; the range of F_k is $\mathbb{G}$.

$\mathsf{F}_k(\cdot)$: Given $(i,j)\in[m]\times[n]$, it outputs $\mathsf{F}_k(i,j)=u_i^{k_j}$.

In the full version, we show Σ_4 is a DDH-based PRF. Let $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{Z}_p^m$ and $\boldsymbol{y} = \{y_{i,j}\} \in \mathbb{G}^{m \times n}$. Let $C''(\boldsymbol{y}, \boldsymbol{x}) = \{\prod_{i=1}^m (y_{i,j})^{x_i}\}_{j=1}^n$. Then $\prod_{i=1}^m \mathsf{F}_k(i,j)^{x_i} = \prod_{i=1}^m (u_i^{k_j})^{x_i} = (\prod_{i=1}^m u_i^{x_i})^{k_j}$ for every $j \in [n]$. Computing $C''(\{\mathsf{F}_k(i,j)\}, \boldsymbol{x})$ without k requires $O(mn)$ operations. Given k, the $\Sigma_4.\mathsf{CFE}$ can compute $U = \prod_{i=1}^m u_i^{x_i}$ and then U^{k_j} for all $j \in [n]$ using $O(m+n)$ operations. Thus, Σ_4 has closed-form efficiency for C''.

Construction 5. Fiore et al. (Section 3.1.3, [7]) constructed a PRF $\Sigma_5 = (\mathsf{Kg}, \mathsf{F})$. The $\mathsf{Kg}(1^\lambda, (m,n))$ picks $\Lambda = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda, 2)$, $u_i, v_i \leftarrow \mathbb{G}$ for every $i \in [m]$, $k_j, l_j \leftarrow \mathbb{Z}_p$ for every $j \in [n]$ and outputs $k = \{(u_i, v_i) : i \in [m]\} \cup \{(k_j, l_j) : j \in [n]\}$ and $\mathsf{pp} = (\Lambda, m, n)$. The domain and range of F_k are $\mathbb{I} = [m] \times [n]$ and $\mathbb{G}$, respectively. For any $(i,j) \in \mathbb{I}$, $\mathsf{F}_k(i,j) = u_i^{k_j} v_i^{l_j}$. They showed that Σ_5 is a DLIN-based PRF and has closed-form efficiency for C''.

3 Our Schemes

3.1 Verifiable Delegation of Polynomials

In this section, we present two VC schemes Π_1 and Π_2 for delegating the polynomial f in Section 1.3. We use all notations from there. Furthermore, we suppose that $d+1 = 2^a = sn = s \cdot 2^b$ for some integers $a, b > 0$. Equation (1) reduces the computation of $f(\boldsymbol{x})$ to the s inner products $\rho_1 = \boldsymbol{y} \cdot \boldsymbol{F}_1, \ldots, \rho_s = \boldsymbol{y} \cdot \boldsymbol{F}_s$. In Π_1 and Π_2 the server must return $\rho_1, \ldots, \rho_s$ and a proof such that the client can verify and then compute $f(\boldsymbol{x})$.

A Privately Verifiable Scheme: Fig. 2 shows our private VC scheme Π_1 for delegating $f(\boldsymbol{x})$. The PRF Σ is Σ_1 or Σ_2. The params is equal to (m, s, n) when $\Sigma = \Sigma_1$ and equal to (m, a, b) when $\Sigma = \Sigma_2$. The τ in Π_1 is computed using $\Sigma.\mathsf{CFE}$. It is easy to see that Π_1 is correct.

$\mathsf{KeyGen}(1^\lambda, f)$: picks $(\mathsf{pp}, k) \leftarrow \Sigma.\mathsf{Kg}(1^\lambda, \mathsf{params})$, where $\mathsf{pp} = (\Lambda, \mathsf{params})$ and $\Lambda = (p, \mathbb{G}, g)$; picks $\alpha \leftarrow \mathbb{Z}_p$; computes $t_{\boldsymbol{i}} = g^{\alpha F_{1,\boldsymbol{i}} + \alpha^2 F_{2,\boldsymbol{i}} + \cdots + \alpha^s F_{s,\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$ for every $\boldsymbol{i} \in \mathbb{I}$; then outputs $ek = (f, \{t_{\boldsymbol{i}}\}_{\boldsymbol{i} \in \mathbb{I}})$, $dk = k$ and $vk = \alpha$.

$\mathsf{ProbGen}(dk, \boldsymbol{x})$: given $\boldsymbol{x} \in \mathbb{Z}_p^m$, outputs $\sigma = \boldsymbol{x}$ and $\tau = \prod_{\boldsymbol{i} \in \mathbb{I}} \mathsf{F}_k(\boldsymbol{i})^{\boldsymbol{x}^{\boldsymbol{i}}}$.

$\mathsf{Compute}(ek, \sigma)$: computes $\rho_\ell = \sum_{\boldsymbol{i} \in \mathbb{I}} F_{\ell,\boldsymbol{i}} \cdot \boldsymbol{x}^{\boldsymbol{i}}$ for every $\ell \in [s]$ and $\pi = \prod_{\boldsymbol{i} \in \mathbb{I}} (t_{\boldsymbol{i}})^{\boldsymbol{x}^{\boldsymbol{i}}}$; then outputs $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_s)$ and π.

$\mathsf{Verify}(vk, \tau, \boldsymbol{\rho}, \pi)$: verifies if $\pi = g^{\rho_1 \cdot \alpha + \rho_2 \cdot \alpha^2 + \cdots + \rho_s \cdot \alpha^s} \cdot \tau$. Outputs $y = \sum_{\ell=1}^s \rho_\ell \cdot x_1^{(\ell-1)n}$ if the equality holds; otherwise, outputs $\perp$.

Fig. 2. The scheme Π_1

Theorem 1. *Π_1 is secure under the d-SDDH assumption for $\mathbb{G}$ when $\Sigma = \Sigma_1$ and secure under the DDH assumption for $\mathbb{G}$ when $\Sigma = \Sigma_2$.*

Proof. Let $\mathbf{G}_0$ be the standard security game for Π_1 (Fig. 1). Let $\mathbf{G}_1$ be a security game which makes no difference with $\mathbf{G}_0$ except that the function F_k is replaced with a random function $\Phi : \mathbb{I} \to \mathbb{G}$. Let $\mathcal{A}$ be any PPT adversary. Let ϵ_i be the probability that $\mathcal{A}$ wins in $\mathbf{G}_i$ for every $i \in \{0, 1\}$. We need to show $\epsilon_0 < \mathsf{neg}(\lambda)$. Firstly, we have $|\epsilon_0 - \epsilon_1| < \mathsf{neg}(\lambda)$ because otherwise one can use $\mathcal{A}$ to break the security of Σ which however is secure under the respective assumptions. Thus, it suffices to show $\epsilon_1 < \mathsf{neg}(\lambda)$. We show $\epsilon_1 < \mathsf{neg}(\lambda)$ even if $\mathcal{A}$ is computationally unbounded.

Consider $\mathbf{G}_1$. We use the notations f, m, d, s, n, a, b and $\mathbb{I}$ from the beginning of this section. Let $ek = (f, \{t_{\boldsymbol{i}}\}_{\boldsymbol{i}\in\mathbb{I}})$, $dk = k$ and $vk = \alpha$ be the keys generated by $\mathsf{KeyGen}(1^\lambda, f)$. Note that the function F_k is replaced with Φ and therefore $t_{\boldsymbol{i}} = g^{\alpha F_{1,\boldsymbol{i}} + \alpha^2 F_{2,\boldsymbol{i}} + \cdots + \alpha^s F_{s,\boldsymbol{i}}} \cdot \Phi(\boldsymbol{i})$ for every $\boldsymbol{i} \in \mathbb{I}$. The adversary $\mathcal{A}$ is given ek. For any choice of $\alpha \in \mathbb{Z}_p$, there is a unique choice of $\Phi(\boldsymbol{i}) \in \mathbb{G}$ for every $\boldsymbol{i} \in \mathbb{I}$ such that $t_{\boldsymbol{i}}$ is consistent with $\mathcal{A}$'s view. As Φ is truly random, $\mathcal{A}$ learns no information about α from ek even if it is computationally unbounded (such that computing discrete logarithms is easy). Thus, from $\mathcal{A}$'s view, $\boldsymbol{\alpha} = (1, \alpha, \ldots, \alpha^s)$ is uniformly chosen from V_{pol}. Given ek, the adversary $\mathcal{A}$ adaptively makes a polynomial number of queries $\{\boldsymbol{x}_j\}_{j=1}^q$ to $\mathsf{ProbGen}(dk, \cdot)$ and $\{(\bar{\boldsymbol{\rho}}_j, \bar{\pi}_j)\}_{j=1}^q$ to $\mathsf{Verify}(vk, \tau_j, \cdot, \cdot)$:

for $j = 1$ to q, the challenger and $\mathcal{A}$ proceeds as below.
- $\mathcal{A}$ gives an input $\boldsymbol{x}_j = (x_{j,1}, \ldots, x_{j,m}) \in \mathbb{Z}_p^m$ to the challenger;
- the challenger gives $\sigma_j = \boldsymbol{x}_j$ to $\mathcal{A}$ and keeps $\tau_j = \prod_{\boldsymbol{i}\in\mathbb{I}} \Phi(\boldsymbol{i})^{\boldsymbol{x}_j^{\boldsymbol{i}}}$;
- $\mathcal{A}$ gives $\bar{\boldsymbol{\rho}}_j = (\bar{\rho}_{j,1}, \ldots, \bar{\rho}_{j,s}) \in \mathbb{Z}_p^s$ and $\bar{\pi}_j \in \mathbb{G}$ to the challenger;
- the challenger gives the output of $\mathsf{Verify}(vk, \tau_j, \bar{\boldsymbol{\rho}}_j, \bar{\pi}_j)$ to $\mathcal{A}$.

After the queries, $\mathcal{A}$ needs to produce a forgery. As remarked in Section 2.1, we can suppose that $\mathcal{A}$'s forgery is $(\boldsymbol{x}_c, \bar{\boldsymbol{\rho}}_c, \bar{\pi}_c)$ for some $c \in [q]$. Let $(\boldsymbol{\rho}_c, \pi_c) \leftarrow \mathsf{Compute}(ek, \sigma_c)$ be the response that could be computed by an honest server, where $\boldsymbol{\rho}_c = (\rho_{c,1}, \ldots, \rho_{c,s}) \in \mathbb{Z}_p^s$ and $\pi_c \in \mathbb{G}$. Due to the correctness of Π_1, we must have that $\pi_c = g^{\sum_{\ell=1}^s \rho_{c,\ell} \cdot \alpha^\ell} \cdot \tau_c$. The adversary $\mathcal{A}$ wins in $\mathbf{G}_1$ only if $\bar{\boldsymbol{\rho}}_c \neq \boldsymbol{\rho}_c$ and $\bar{\pi}_c = g^{\sum_{\ell=1}^s \bar{\rho}_{c,\ell} \cdot \alpha^\ell} \cdot \tau_c$. It follows that $\mathcal{A}$ wins only if $\bar{\boldsymbol{\rho}}_c \neq \boldsymbol{\rho}_c$ and

$$\bar{\pi}_c / \pi_c = g^{\sum_{\ell=1}^s (\bar{\rho}_{c,\ell} - \rho_{c,\ell}) \cdot \alpha^\ell}. \tag{3}$$

Suppose that $\bar{\pi}_c / \pi_c = g^{\beta_c}$. Then (3) holds only if $\boldsymbol{u}_c = (-\beta_c, \bar{\rho}_{c,1} - \rho_{c,1}, \ldots, \bar{\rho}_{c,s} - \rho_{c,s}) \in \mathbb{Z}_p^{s+1}$ is a nonzero vector such that $\boldsymbol{u}_c \cdot \boldsymbol{\alpha} = 0$. Recall that $\boldsymbol{\alpha} \leftarrow V_{\mathrm{pol}}$. Without loss of generality, we can suppose that the first nonzero component of $\boldsymbol{u}_c$ is 1 such that $\boldsymbol{u}_c \in \Omega_{p,s+1}$. This does not matter because if the first nonzero component of $\boldsymbol{u}_c$ is $\gamma \neq 0$ then $\gamma^{-1} \cdot \boldsymbol{u}_c$ will belong to $\Omega_{p,s+1}$ and $\mathcal{A}$ could have made the query $\gamma^{-1} \cdot \boldsymbol{u}_c$ instead of $\boldsymbol{u}_c$ with the same consequence (i.e., success or failure). In general, for every $j \in [q]$ and the j-th queries $\boldsymbol{x}_j$ and $(\bar{\boldsymbol{\rho}}_j, \bar{\pi}_j)$, we can follow the analysis for $j = c$ and learn a vector $\boldsymbol{u}_j \in \Omega_{p,s+1}$. The j-th queries cause $\mathcal{A}$ to win only if $\boldsymbol{u}_j \cdot \boldsymbol{\alpha} = 0$. Thus, the query part of $\mathbf{G}_1$ turns out to be **Problem 1** with $U \subseteq \Omega_{p,s+1}$ and $V = V_{\mathrm{pol}}$. Lemma 1 and Example 2 show that $\mathcal{A}$ wins with probability $\leq sq/p$, which is negligible, i.e., $\epsilon_1 < \mathsf{neg}(\lambda)$. □

A Publicly Verifiable Scheme: Fig. 3 shows our public VC scheme Π_2 for delegating $f(\boldsymbol{x})$. The τ in Π_2 is computed using $\Sigma_3.\mathsf{CFE}$. It is easy to see that Π_2 is correct.

$\mathsf{KeyGen}(1^\lambda, f)$: picks $(\mathsf{pp}, k) \leftarrow \Sigma_3.\mathsf{Kg}(1^\lambda, (m, a, b))$, where $\mathsf{pp} = (\Lambda, m, a, b)$ and $\Lambda = (p, \mathbb{G}, \mathbb{G}_T, e, g)$; picks $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_s) \leftarrow \mathbb{Z}_p^s$; computes $t_{\boldsymbol{i}} = g^{\alpha_1 F_{1,\boldsymbol{i}} + \cdots + \alpha_s F_{s,\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$ for every $\boldsymbol{i} \in \mathbb{I}$; then outputs $ek = (f, \{t_{\boldsymbol{i}}\}_{\boldsymbol{i}\in\mathbb{I}})$, $dk = k$ and $vk = (h_1, \ldots, h_s) = (e(g,g)^{\alpha_1}, \ldots, e(g,g)^{\alpha_s})$;

$\mathsf{ProbGen}(dk, \boldsymbol{x})$: given $\boldsymbol{x} \in \mathbb{Z}_p^m$, outputs $\sigma = \boldsymbol{x}$ and $\tau = e\big(\prod_{\boldsymbol{i}\in\mathbb{I}} \mathsf{F}_k(\boldsymbol{i})^{\boldsymbol{x}^{\boldsymbol{i}}}, g\big)$.

$\mathsf{Compute}(ek, \sigma)$: computes $\rho_\ell = \sum_{\boldsymbol{i}\in\mathbb{I}} F_{\ell,\boldsymbol{i}} \cdot \boldsymbol{x}^{\boldsymbol{i}}$ for every $\ell \in [s]$ and $\pi = \prod_{\boldsymbol{i}\in\mathbb{I}} (t_{\boldsymbol{i}})^{\boldsymbol{x}^{\boldsymbol{i}}}$; then outputs $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_s)$ and π.

$\mathsf{Verify}(vk, \tau, \boldsymbol{\rho}, \pi)$: verifies if $e(\pi, g) = \prod_{\ell=1}^s h_\ell^{\rho_\ell} \cdot \tau$. outputs $y = \sum_{\ell=1}^s \rho_\ell \cdot x_1^{(\ell-1)n}$ if the equality holds; otherwise, outputs $\perp$.

Fig. 3. The scheme Π_2

Theorem 2. *Π_2 is secure under the DLIN assumption for $\mathbb{G}$.*

Proof. Let $\mathbf{G}_0$ be the standard security game for Π_2 (see Fig. 1). Let $\mathbf{G}_1$ be a security game which makes no difference with $\mathbf{G}_0$ except that the function F_k is replaced with a random function $\Phi : \mathbb{I} \to \mathbb{G}$. Let $\mathcal{A}$ be any PPT adversary. Let ϵ_i be the probability that $\mathcal{A}$ wins in $\mathbf{G}_i$ for every $i \in \{0, 1\}$. As in Theorem 1, it suffices to show that $\epsilon_1 < \mathsf{neg}(\lambda)$.

Consider $\mathbf{G}_1$. We use the notations f, m, d, s, n, a, b and $\mathbb{I}$ from the beginning of Section 3.1. Suppose that ϵ_1 is non-negligible. We show that there is a challenger $\mathcal{B}$ that can simulate $\mathcal{A}$ to solve the CDH problem, which however should be hard under DLIN. Given a CDH problem (g, g^α, h), $\mathcal{B}$ must output h^α. Fig. 4 shows how $\mathcal{B}$ plays with the adversary $\mathcal{A}$ in $\mathbf{G}_1$. Let $(\boldsymbol{\rho}_c, \pi_c) \leftarrow \mathsf{Compute}(ek, \sigma_c)$ be the response that could be computed by an honest server,

Setup. The challenger $\mathcal{B}$ picks $t_{\boldsymbol{i}} \leftarrow \mathbb{G}$ for every $\boldsymbol{i} \in \mathbb{I}$; picks $r \leftarrow [s]$ and computes $h_r = e(g^\alpha, h)$; picks $\alpha_\ell \leftarrow \mathbb{Z}_p$ and computes $h_\ell = e(g,g)^{\alpha_\ell}$ for every $\ell \in [s] \setminus \{r\}$; then it defines $ek = (f, \{t_{\boldsymbol{i}}\}_{\boldsymbol{i}\in\mathbb{I}}), dk = \perp, vk = (h_1, \ldots, h_s)$; $\mathcal{B}$ gives (ek, vk) to $\mathcal{A}$.

Queries. The adversary $\mathcal{A}$ adaptively makes a polynomial number of queries $\{\boldsymbol{x}_j\}_{j=1}^q$ to $\mathsf{ProbGen}(dk, \cdot)$ and $\{(\bar{\rho}_j, \bar{\pi}_j)\}_{j=1}^q$ to $\mathsf{Verify}(vk, \tau_j, \cdot, \cdot)$:

for $j = 1$ to q, $\mathcal{B}$ and $\mathcal{A}$ proceed as below

1. $\mathcal{A}$ picks $\boldsymbol{x}_j = (x_{j,1}, \ldots, x_{j,m}) \in \mathbb{Z}_p^m$ and give it to $\mathcal{B}$;
2. $\mathcal{B}$ gives $\sigma_j = \boldsymbol{x}_j$ and $\tau_j = e\big(\prod_{\boldsymbol{i}\in\mathbb{I}} (t_{\boldsymbol{i}})^{\boldsymbol{x}_j^{\boldsymbol{i}}}, g\big) / \prod_{\ell=1}^s h_\ell^{\sum_{\boldsymbol{i}\in\mathbb{I}} F_{\ell,\boldsymbol{i}} \cdot \boldsymbol{x}_j^{\boldsymbol{i}}}$ to $\mathcal{A}$;
3. $\mathcal{A}$ picks $\bar{\boldsymbol{\rho}}_j = (\bar{\rho}_{j,1}, \ldots, \bar{\rho}_{j,s}) \in \mathbb{Z}_p^s$ and $\bar{\pi}_j \in \mathbb{G}$ and gives them to $\mathcal{B}$;
4. $\mathcal{B}$ gives the output of $\mathsf{Verify}(vk, \tau_j, \bar{\boldsymbol{\rho}}_j, \bar{\pi}_j)$ to $\mathcal{A}$.

Forgery. As remarked in Section 2.1, $\mathcal{A}$ outputs $(\boldsymbol{x}_c, \bar{\boldsymbol{\rho}}_c, \bar{\pi}_c)$ as its forgery ($c \in [q]$).

Fig. 4. $\mathcal{B}$'s simulation in the game $\mathbf{G}_1$

where $\boldsymbol{\rho}_c = (\rho_{c,1}, \ldots, \rho_{c,s}) \in \mathbb{Z}_p^s$ and $\pi_c \in \mathbb{G}$. Due to the correctness of Π_2, we have that $e(\pi_c, g) = \prod_{\ell=1}^{s} h_\ell^{\rho_{c,\ell}} \cdot \tau_c$. The adversary $\mathcal{A}$ wins the game $\mathbf{G}_1$ only if $\bar{\boldsymbol{\rho}}_c \neq \boldsymbol{\rho}_c$ and $e(\bar{\pi}_c, g) = \prod_{\ell=1}^{s} h_\ell^{\bar{\rho}_{c,\ell}} \cdot \tau_c$. It follows that $\mathcal{A}$ wins only if $\bar{\boldsymbol{\rho}}_c \neq \boldsymbol{\rho}_c$ and

$$e(\bar{\pi}_c/\pi_c, g) = \prod_{\ell=1}^{s} h_\ell^{\bar{\rho}_{c,\ell}-\rho_{c,\ell}} \tag{4}$$

It is not hard to see that the (ek, vk) and $\{(\sigma_j, \tau_j)\}_{j=1}^{q}$ generated by $\mathcal{B}$ strictly follow the respective distributions in $\mathbf{G}_1$, although they are not obtained by directly running the algorithms KeyGen and $\mathsf{ProbGen}$. Due to our assumption, $\mathcal{A}$ should win with probability ϵ_1, i.e., the probability that $\bar{\boldsymbol{\rho}}_c \neq \boldsymbol{\rho}_c$ and (4) holds is ϵ_1. As $\bar{\boldsymbol{\rho}}_c \neq \boldsymbol{\rho}_c$, there is a nonempty set $R \subseteq [s]$ such that $\bar{\rho}_{c,r^*} \neq \rho_{c,r^*}$ for any $r^* \in R$. The r in Fig. 4 was uniformly chosen and independent of everything else. Therefore, the probability that r falls into R is $\geq |R|/s \geq 1/s$. The challenger $\mathcal{B}$ as a CDH-solver outputs $\perp$ (to indicate failure) if $r \notin R$. Otherwise, (4) implies that $e(\bar{\pi}_c \cdot \pi_c^{-1}, g) \cdot \prod_{\ell \in [s]\setminus\{r\}} h_\ell^{-(\bar{\rho}_{c,\ell}-\rho_{c,\ell})} = h_r^{\bar{\rho}_{c,r}-\rho_{c,r}} = e(g^\alpha, h)^{\bar{\rho}_{c,r}-\rho_{c,r}} = e(h^{\alpha(\bar{\rho}_{c,r}-\rho_{c,r})}, g)$. It follows that the challenger $\mathcal{B}$ can compute $h^\alpha = (\bar{\pi}_c \cdot \pi_c^{-1} \cdot \prod_{\ell \in [s]\setminus\{r\}} g^{-\alpha_\ell(\bar{\rho}_{c,\ell}-\rho_{c,\ell})})^{\frac{1}{\bar{\rho}_{c,r}-\rho_{c,r}}}$. The probability that $\mathcal{B}$ learns h^α is exactly equal to the probability that $\mathcal{A}$ wins in $\mathbf{G}_1$ and $r \in R$, which is $\geq \epsilon_1/s$ and thus non-negligible. This contradicts the hardness of CDH and thus the DLIN assumption. Hence, ϵ_1 must be negligible.

3.2 Verifiable Delegation of Matrices

In this section, we present two VC schemes Π_3 and Π_4 for delegating the matrix $\boldsymbol{E}$ in Section 1.4. We use all notations from there. Recall that $\boldsymbol{x} \cdot \boldsymbol{E}$ can be reduced to n sets of inner product computations. In Π_3 and Π_4 the server must return $\boldsymbol{x} \cdot \boldsymbol{E}$ and n proofs, one for each set of s inner product computations.

A Privately Verifiable Scheme: Fig. 5 shows our private VC scheme Π_3 for delegating $\boldsymbol{E}$. The τ in Π_3 is computed using $\Sigma_4.\mathsf{CFE}$. It is easy to see that Π_3 is correct. Due to lack of space, we show that Π_3 is secure under the DDH assumption for $\mathbb{G}$ in the full version.

$\mathsf{KeyGen}(1^\lambda, \boldsymbol{E})$: picks $(\mathsf{pp}, k) \leftarrow \Sigma_4.\mathsf{Kg}(1^\lambda, (m, n))$, where $\mathsf{pp} = (\Lambda, m, n)$ and $\Lambda = (p, \mathbb{G}, g)$; picks $\alpha \leftarrow \mathbb{Z}_p$; computes $t_{i,j} = t_{\boldsymbol{i}} = g^{\alpha F_{1,\boldsymbol{i}} + \alpha^2 F_{2,\boldsymbol{i}} + \cdots + \alpha^s F_{s,\boldsymbol{i}}} \cdot \mathsf{F}_k(\boldsymbol{i})$ for every $\boldsymbol{i} = (i, j) \in [m] \times [n]$; then outputs $ek = (\boldsymbol{E}, \{t_{\boldsymbol{i}}\})$, $dk = k$ and $vk = \alpha$.

$\mathsf{ProbGen}(dk, \boldsymbol{x})$: given $\boldsymbol{x} \in \mathbb{Z}_p^m$, computes $\tau_j = \prod_{i=1}^{m} \mathsf{F}_k(i, j)^{x_i}$ for every $j \in [n]$; then outputs $\sigma = \boldsymbol{x}$ and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$.

$\mathsf{Compute}(ek, \sigma)$: computes $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_d) = \boldsymbol{x} \cdot \boldsymbol{E}$ and $\pi_j = \prod_{i=1}^{m} (t_{i,j})^{x_i}$ for every $j \in [n]$; then outputs $\boldsymbol{\rho}$ and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$.

$\mathsf{Verify}(vk, \boldsymbol{\tau}, \boldsymbol{\rho}, \boldsymbol{\pi})$: verifies if $\pi_j = g^{\sum_{\ell=1}^{s} \rho_{j+(\ell-1)n} \cdot \alpha^\ell} \cdot \tau_j$ for every $j \in [n]$; if all equalities hold, outputs $\boldsymbol{\rho}$; otherwise, outputs $\perp$.

Fig. 5. The scheme Π_3

A Publicly Verifiable Scheme: Fig. 6 shows our public VC scheme Π_4 for delegating $\boldsymbol{E}$. The τ is in Π_4 is computed using Σ_5.CFE. It is easy to see that Π_4 is correct. Due to lack of space, we show that Π_4 is secure under the DLIN assumption for $\mathbb{G}$ in the full version.

$\mathsf{KeyGen}(1^\lambda, \boldsymbol{E})$: picks $(\mathsf{pp}, k) \leftarrow \Sigma_5.\mathsf{Kg}(1^\lambda, (m, n))$, where $\mathsf{pp} = (\Lambda, m, n)$ and $\Lambda = (p, \mathbb{G}, \mathbb{G}_T, e, g)$; picks $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_s) \leftarrow \mathbb{Z}_p^s$; computes $t_{\boldsymbol{i}} = t_{i,j} = g^{\alpha_1 F_{1,\boldsymbol{i}} + \cdots + \alpha_s F_{s,\boldsymbol{i}}}$. $\mathsf{F}_k(\boldsymbol{i})$ for every $\boldsymbol{i} = (i, j) \in [m] \times [n]$; then outputs $ek = (\boldsymbol{E}, \{t_{\boldsymbol{i}}\})$, $dk = k$ and $vk = (h_1, \ldots, h_s) = (e(g,g)^{\alpha_1}, \ldots, e(g,g)^{\alpha_s})$;

$\mathsf{ProbGen}(dk, \boldsymbol{x})$: given $\boldsymbol{x} \in \mathbb{Z}_p^m$, computes $\tau_j = e\big(\prod_{i=1}^m \mathsf{F}_k(i,j)^{x_i}, g\big)$ for every $j \in [n]$; then outputs $\sigma = \boldsymbol{x}$ and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$.

$\mathsf{Compute}(ek, \sigma)$: computes $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_d) = \boldsymbol{x} \cdot \boldsymbol{E}$ and $\pi_j = \prod_{i=1}^m (t_{i,j})^{x_i}$ for every $j \in [n]$; then outputs $\boldsymbol{\rho}$ and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$.

$\mathsf{Verify}(vk, \boldsymbol{\tau}, \boldsymbol{\rho}, \boldsymbol{\pi})$: verifies if $e(\pi_j, g) = \prod_{\ell=1}^s h_\ell^{\rho_{j+(\ell-1)n}} \cdot \tau_j$ for every $j \in [n]$; if all equalities hold, outputs $\boldsymbol{\rho}$; otherwise, outputs $\perp$.

Fig. 6. The scheme Π_4

3.3 Performance Analysis and Extensions

In this section we analyze our schemes. We take Π_1 and Π_4 as example. The analysis of Π_2 and Π_3 can be done similarly.

Analysis of Π_1. STORAGE: In Π_1, the client stores $ek = (f, \{t_{\boldsymbol{i}}\}_{\boldsymbol{i} \in \mathbb{I}})$ with the server, where f can be represented by $(d+1)^m$ elements of $\mathbb{Z}_p$ and $t_{\boldsymbol{i}}$ belongs to a group $\mathbb{G}$ of order p for every $\boldsymbol{i} \in \mathbb{I}$. The storage overhead of Π_1 is $((d+1)^m + |\mathbb{I}|)/(d+1)^m = 1 + 1/s$. Let $\lambda = 1024, |p| = \lambda, m = 1, d+1 = 2^{30}, s = \lambda$ and $n = 2^{20}$. Let the $\mathbb{G}$ in Π_1 be an order p subgroup of $\mathbb{Z}_{p'}^*$, where p' is a prime. To delegate f, the client stores $|\mathbb{I}| = 2^{20}$ tags in $\mathbb{G}$ with the server. Thus, to delegate $(d+1)^m \times |p|/2^3\mathrm{B} = 128\mathrm{GB}$ data, Π_1 requires $|\mathbb{I}| \cdot \lambda/2^3\mathrm{B} = 128\mathrm{MB}$ cloud storage for tags. This is only 1/1024 times the 128GB tags used by [2]. VERIFICATION: Let $\mathsf{E}_p, \mathsf{M}_p$ and A_p be the number of bit operations required by each exponentiation, multiplication, and addition in $\mathbb{Z}_p$, respectively. Let $\mathsf{E}_\mathbb{G}$ and $\mathsf{M}_\mathbb{G}$ be the number of bit operations required by each exponentiation and multiplication in $\mathbb{G}$, respectively. Let $\mathsf{C}_\mathbb{G}$ be the number of bit operations required for comparing two elements of $\mathbb{G}$. In Π_1.ProbGen, the client requires $m\mathsf{E}_p + 3m\mathsf{M}_p + 2m\mathsf{A}_p$ bit operations to compute the τ using Σ_1.CFE. In Π_1.Verify, the client requires $(2s-1)\mathsf{M}_p + (s-1)\mathsf{A}_p$ bit operations to compute $\eta = \rho_1\alpha + \cdots + \rho_s\alpha^s$, $\mathsf{E}_\mathbb{G}$ bit operations to compute g^η, $\mathsf{M}_\mathbb{G}$ bit operations to compute $g^\eta \cdot \tau$ and then $\mathsf{C}_\mathbb{G}$ bit operations to compare π with $g^\eta \cdot \tau$; it also requires E_p bit operations to compute x_1^n, $(s-2)\mathsf{M}_p$ bit operations to compute $x_1^{2n}, ..., x_1^{(s-1)n}$, and then $s\mathsf{M}_p + (s-1)\mathsf{A}_p$ bit operations to compute $f(\boldsymbol{x})$. Thus, the client's verification totally requires $(m+1)\mathsf{E}_p + (3m+4s-3)\mathsf{M}_p + (2m+2s-2)\mathsf{A}_p + \mathsf{E}_\mathbb{G} + \mathsf{M}_\mathbb{G} + \mathsf{C}_\mathbb{G}$ bit operations. The scheme of [2] requires $m\mathsf{E}_p + (3m+1)\mathsf{M}_p + 2m\mathsf{A}_p + \mathsf{E}_\mathbb{G} + \mathsf{M}_\mathbb{G} + \mathsf{C}_\mathbb{G}$ bit operations to do verification. Note that $\mathsf{E}_p, \mathsf{E}_\mathbb{G} \approx \lambda^3, \mathsf{M}_p, \mathsf{M}_\mathbb{G} \approx \lambda^2$, and $\mathsf{A}_p, \mathsf{C}_\mathbb{G} \approx \lambda$. The client in Π_1 requires $\approx ((m+2)\lambda + 4s)\lambda^2$ bit operations and

the client in [2] requires $\approx (m+1)\lambda^3$ bit operations. Therefore, our client is roughly $\delta = 1 + \frac{\lambda+4s}{(m+1)\lambda}$ times slower than the client of [2]. When $m = 1$ and $s = \lambda$, we have that $\delta = 3.5$. Our parameter s provides a meaningful trade-off between the size of tags and the client's verification time. The larger the s is, the smaller the storage overhead is and the slower the client of Π_1 is. The δ shows that our client can significantly reduce the consumption of cloud storage by slightly increasing the verification time.

Analysis of Π_4. Our scheme Π_4 uses a random bilinear map instance $(p, \mathbb{G}, \mathbb{G}_T, e, g)$. In $\Pi_4.\mathsf{ProbGen}$, the client requires $m+n$ exponentiations in $\mathbb{G}$ and n pairing computations to compute $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ using $\Sigma_5.\mathsf{CFE}$. In $\Pi_4.\mathsf{Verify}$, the client requires s exponentiations in $\mathbb{G}_T$, s multiplications in $\mathbb{G}_T$, one pairing computation and one comparision of the elements of $\mathbb{G}_T$ to check the equality $e(\pi_j, g) = \prod_{\ell=1}^{s} h_\ell^{\rho_{j+(\ell-1)n}} \cdot \tau_j$ for every $j \in [n]$. Thus, the client's verification time is dominated by $m+n$ exponentiations in $\mathbb{G}$, sn exponentiations in $\mathbb{G}_T$, and $2n$ pairing computations. The scheme of [7] is a special case of Π_4 with $s = 1$. In their scheme, the client requires $m + sn$ exponentiations in $\mathbb{G}$ and sn pairing computations to compute $d = sn$ elements $\tau_1, \ldots, \tau_d$ for future verification. The client also requires one exponentiation in $\mathbb{G}_T$, one multiplication in $\mathbb{G}_T$, one pairing computation and one comparision of the elements of $\mathbb{G}_T$ to check an equality for each of the sn components of $\boldsymbol{x} \cdot \boldsymbol{E}$. Therefore, the verification time of their client is dominated by $(m+sn)$ exponentiations in $\mathbb{G}$, sn exponentiations in $\mathbb{G}_T$, and $2sn$ pairing computations. Hence, our publicly verifiable scheme Π_4 for delegating matrices is much more efficient than [7] not only in terms of storage overhead ($1 + 1/s$ vs. 2) but also in terms of the client's verification time.

Extensions. The coefficients of f are considered as s vectors in Π_1 and the computation of $f(\boldsymbol{x})$ is reduced to computing inner products with them. Each inner product is an evaluation of an m-variate polynomial of degree $\leq n-1$ in x_1 and degree $\leq d$ in any other variables. We can *repeatedly apply* our technique on these shorter polynomials to further reduce the degrees of $x_2, \ldots, x_m$ such that the computation of $f(\boldsymbol{x})$ is reduced to evaluating s^m different m-variate polynomials of degree $\leq n-1$ in each variable. This is particularly useful when $d = O(1)$ but $m = O(\log \lambda)$. Our schemes also provide *batch verifications* of multiple functions. For example, if we set $m = 1$, then Π_1 allows the client to verify the evaluations of s univariate polynomials of degree $\leq n-1$ using substantially less time ($\approx 7\lambda^3$ bit operations when $s = \lambda$) than delegating the s polynomials separately using [2] (which requires $\approx 2s\lambda^3$ bit operations for verification).

4 Conclusions

In this paper, we construct VC schemes for delegating polynomials and matrices that provide trade-offs between the consumed cloud storage and the client's verification time. As [2,7], our polynomial f must have special form. For example, in Π_1 we require that $d+1 = 2^a = sn = s2^b$. This is necessary to use PRFs with closed-form efficiency. For a general polynomial, one may add redundant terms to meet the requirement. It is interesting to extend our results to more general functions such as the m-variate polynomials of total degree $\leq d$.

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: From Secrecy to Soundness: Efficient Verification via Secure Computation. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 152–163. Springer, Heidelberg (2010)
2. 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)
3. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. In: ITCS, pp. 326–349 (2012)
4. Boneh, D., Freeman, D.M.: Homomorphic Signatures for Polynomial Functions. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 149–168. Springer, Heidelberg (2011)
5. Chung, K.-M., Kalai, Y.T., Liu, F.-H., Raz, R.: Memory Delegation. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 151–168. Springer, Heidelberg (2011)
6. Chung, K.-M., Kalai, Y., Vadhan, S.P.: Improved Delegation of Computation Using Fully Homomorphic Encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
7. Fiore, D., Gennaro, R.: Publicly Verifiable Delegation of Large Polynomials and Matrix Computations, with Applications. In: CCS, pp. 501–512 (2012)
8. 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)
9. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013)
10. Gennaro, R., Wichs, D.: Fully Homomorphic Message Authenticators. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 301–320. Springer, Heidelberg (2013)
11. Gentry, C., Wichs, D.: Separating Succinct Non-Interactive Arguments from All Falsfiable Assumptions. In: STOC, pp. 99–108 (2011)
12. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating Computation: Interactive Proofs for Muggles. In: STOC, pp. 113–122 (2008)
13. Juels, A., Kaliski, B.: PORs: Proofs of Retrievability for Large Files. In: CCS, pp. 584–597 (2007)
14. Lewko, A.B., Waters, B.: Efficient Pseudorandom Functions from the Decisional Linear Assumption and Weaker Variants. In: CCS, pp. 112–120 (2009)
15. Naor, M., Reingold, O.: Number-Theoretic Constructions of Efficient Pseudo-Random Functions. J. ACM 51(2), 231–262 (2004)
16. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of Correct Computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 222–242. Springer, Heidelberg (2013)
17. 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)
18. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)