Batch Verifiable Computation of Polynomials on Outsourced Data

Liang Feng Zhang[1(✉)] and Reihaneh Safavi-Naini[2]

[1] ShanghaiTech University, Shanghai, China
zhanglf@shanghaitech.edu.cn
[2] University of Calgary, Calgary, Canada

Abstract. Secure outsourcing of computation to cloud servers has attracted much attention in recent years. In a typical outsourcing scenario, the client stores its data on a cloud server and later asks the server to perform computations on the stored data. The verifiable computation (VC) of Gennaro, Gentry, Parno (Crypto 2010) and the homomorphic MAC (HomMAC) of Backes, Fiore, Reischuk (CCS 2013) allow the client to verify the server's computation with substantially less computational cost than performing the outsourced computation. The existing VC and HomMAC schemes that can be considered practical (do not required heavy computations such as computing fully homomorphic encryptions), are limited to compute linear and quadratic polynomials on the outsourced data. In this paper, we introduce a *batch verifiable computation* (BVC) model that can be used when the computation of the same function on multiple datasets is required, and construct two schemes for computing polynomials of high degree on the outsourced data. Our schemes allow *efficient client verification*, *efficient server computation*, and *composition* of computation results. Both schemes allow new elements to be added to each outsourced dataset. The second scheme also allows new datasets to be added. A unique feature of our schemes is that the storage required at the server for storing the authentication information, stays the same as the number of outsourced datasets is increased, and so the *server storage overhead* (the ratio of the server storage to the total size of the datasets) approaches 1. In all existing schemes this ratio is ≥ 2. Hence, our BVC can effectively halve the required server storage.

1 Introduction

Cloud computing provides an attractive solution for computationally weak clients that need to outsource data and perform large-scale computations on the outsourced data. This however raises the important security requirement of enabling the client to verify the correctness of the outsourced computation. A cloud server may return an incorrect result, accidentally or intentionally, and the ability to verify the result is a basic requirement. This requirement has motivated the research on the verifiability of outsourced computation in two directions: exploring the theoretical foundation of what computations can be

G. Pernul et al. (Eds.): ESORICS 2015, Part II, LNCS 9327, pp. 167–185, 2015.
DOI: 10.1007/978-3-319-24177-7_9

securely outsourced, and proposing secure solutions for specific problems with emphasis on practicality. Our work follows the latter direction.

Verifiable Computation. Several models have been proposed for secure outsourcing of computation. In the verifiable computation (VC) model of Gennaro, Gentry and Parno [14], the client's data defines a function and a computation is equivalent to evaluating this function that is computationally expensive. To outsource this computation, the client computes a one-time encoding of the function and stores it at the server. This enables the server to not only evaluate the function on any input, but also provide a proof that the evaluation has been done correctly. The client's verification must be substantially less time-consuming than evaluating the original function. The effort of generating the one-time encoding will be amortized over multiple evaluations of the function and so is considered acceptable.

Following [14] a number of VC schemes [2,10,11,14,21] to delegate generic functions have been proposed. These schemes are based on fully homomorphic encryption (FHE) and so with today's constructions of FHE, cannot be considered practical. Benabbas et al. [5] initiated a line of research [5,9,13,20] on *practical* VC for specific functions such as polynomials, which do not require heavy cryptographic computations such as FHE. In a VC for polynomials, the client's data consists of the coefficients of a polynomial. The client stores an encoding of the coefficients on a cloud server; this encoding allows the server to evaluates the polynomial on any requested point; the client can efficiently verify the server's computation. These schemes are secure against a malicious server which is allowed to make a polynomial (in the security parameter) number of attempts to deceive the client into accepting a wrong computation result, with each attempt being told successful or not.

Practical VC schemes however are limited to the computation of linear functions on the outsourced data (e.g., evaluating a polynomial at a point x is equivalent to computing the inner product of a vector defined by the coefficients with a vector defined by x and linear in the coefficients). This means that even simple statistical functions such as variance, cannot be computed. Also, the encoding of the function doubles the cloud storage needed by the function itself. Evaluating polynomials arise in applications such as proof of retrievability and verifiable keyword search [13], where the number of polynomial coefficients is roughly equal to the number of data elements in a file or database. In those scenarios doubling the cloud storage will result in a substantial increase of the client's expense and will become increasingly problematic as more and more data is outsourced.

Homomorphic MAC. Homomorphic MAC (HomMAC) [16] allows a client to store a *dataset* (a set of data elements) on a cloud server, and later request the computation of some specified functions, referred to as the *admissible function family*, on the dataset. The dataset may consist of employee records of an institution and a possible computation could be evaluating a function of the records. One can add elements to, or remove elements from, the dataset as needed. The encoding of the dataset consists of all data elements and a special MAC tag

for each data element. The tags allow the server to produce a MAC tag for the computation of any admissible function.

HomMACs for admissible linear functions [1] and admissible nonlinear functions [4,8,16] have been proposed. Some of these schemes require heavy cryptographic computations, such as FHE [16]. Catalano and Fiore [8] proposed an elegant HomMAC for high degree polynomials with efficient server computations (including PRF computations and polynomial evaluations over relatively small finite fields). The client verification cost however is effectively the same as performing the outsourced computation. Backes, Fiore and Reischuk [4] removed this drawback by restricting the class of admissible functions to polynomials of degree 2. They considered the computations of the same function on multiple datasets. The verification of the computations requires an expensive preprocessing which is done only once and amortized over all verifications. Restriction on the degree of the polynomials however limits their applicability. For example an important task in data analysis is to determine if a dataset is normally distributed. Two commonly used statistical measures for symmetry and flatness of a distribution relative to normal distribution, are skewness and kurtosis, which require computation of degree 3 and 4 polynomials of the data elements, respectively.

Compared to the VC model of [5,14], the security model of HomMAC is more demanding. Here the server is allowed to learn the MAC tags of arbitrary data elements of its choice and also make a polynomial (in the security parameter) number of attempts to deceive the client into accepting a wrong computation result, with each attempt being told successful or not. This stronger security property means that the HomMACs can be straightforwardly translated into VC schemes but the converse may not be true in general. In a HomMAC based VC scheme the server has to store both the data elements and the MAC tags. This usually doubles the cloud storage consumed by the data elements.

An additional desirable property of HomMACs is that they allow *composition*. That is, given multiple computation results and their MAC tags, one can perform a high level computation on these results and also generate a corresponding MAC tag for this high level computation.

Motivation. The existing VC schemes satisfy a subset of the following desirable properties: (p1) *Large admissible function family*: enabling the computation of high degree polynomials (not limited to the linear and quadratic ones) on the outsourced data; (p2) *Efficient client verification*: the client's verification is substantially less expensive than computing the delegated computation; (p3) *Efficient server computation*: the server does not need to do heavy cryptographic computations (such as FHE); (p4) *Efficient server storage*: the server stores an encoding of the client's data and the encoding consumes almost no extra storage than the data itself. (p5) *Unbounded data*: the client can freely update the outsourced data by adding new elements to every dataset and also adding new datasets. Our goal is to construct schemes that provide *all* the above properties.

1.1 Our Contributions

We introduce *batch verifiable computation (BVC),* and construct two BVC schemes that satisfy properties (p1)–(p5). Similar to Backes et al. [4], we also consider outsourcing of multiple datasets with two labels. The outsourced data m defines an $N \times s$ matrix $(m_{i,j})_{N\times s}$, where each column is called a *dataset*, and each entry $m_{i,j}$ is labeled by a pair $(i, j) \in [N] \times [s]$. However, the similarity ends here: Backes et al. allow computation of different functions on each dataset with the restriction that the polynomials are of degree at most two. Our main observation is that by batching computation of the same function on all datasets, an impressive set of properties can be achieved. In particular one can save storage at the server, and this saving will be significant when the computation on more datasets are outsourced. In BVC the client computes a tag t_i for the ith row of m for every $i \in [N]$, and stores $\mathbf{t} = (t_1, \ldots, t_N)$ as an extra column at the cloud server. A computation is specified by a program $\mathcal{P} = (f, I)$, where $f(x_1, \ldots, x_n)$ is a function and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies the subset of elements of each dataset which will be used in the computation of f. Given the program $\mathcal{P}$, the server returns s results $\rho_1 = f(m_{i_1,1}, \ldots, m_{i_n,1}), \ldots, \rho_s = f(m_{i_1,s}, \ldots, m_{i_n,s})$ and a single *batch proof* π; the client accepts the s results only if they successfully pass the client's verification. A BVC scheme is secure if no malicious cloud server can deceive the client into accepting wrong results. We consider the computation of any polynomial function (i.e., arithmetic circuit) on the outsourced data, and construct two BVC schemes with the following properties.

Large Admissible Function Family. The first scheme admits polynomials of degree as high as any polynomial in the security parameter. The second scheme admits any constant degree polynomials. The only other known practical schemes that can compute the same class of functions is from [8] in which the client's verification is effectively as heavy as the outsourced computation.

Efficient Client Verification. In our BVC schemes the client can verify the computation results on the s datasets using a single batch proof that is computed from the tag column. In both schemes verifying the computation result of each dataset is by evaluating the batch proof (which is a polynomial) at a specific point. The batch proof in the first scheme is a univariate polynomial of bounded degree, and in the second scheme is a multivariate polynomial of bounded degree. Compared with the naive construction where the scheme in [8] is used on each dataset, the client's average verification cost in our schemes is substantially less than what is required by the original computation as long as s is large enough.

Efficient Server Computation. The server computation in our schemes consists of PRF computations and polynomial evaluations over relatively small finite fields (such as $\mathbb{Z}_p$ for $p \approx 2^{128}$ when the security parameter $\lambda = 128$). This is similar to [8] and more efficient than [4] where the server must compute a large number of exponentiations and pairings over significantly larger groups.

Efficient Server Storage. In a VC (or BVC) scheme the client stores an encoding of its data on the cloud server. We define the *storage overhead* of a VC

(or BVC) scheme as the ratio of the size of the encoding to the size of data. It is easy to see that the storage overhead is lower bounded by 1. In both schemes a tag has size equal to an element of m, resulting in a storage overhead of $1 + 1/s$ which approaches 1 as s increases. In all existing practical VC schemes [4,5,8,9,13] the storage overhead is ≥ 2.

Unbounded Data. In our BVC schemes the outsourced data m consists of s datasets, each consisting of N elements. Our schemes allow the client to add an arbitrary number of new rows and/or columns to m, and efficiently update the tag column without downloading m. While adding new rows to m is straightforward, adding new datasets to m without performing the tag computation from scratch (and so downloading the already outsourced data) is highly non-trivial. This is because in our schemes each row of m is authenticated using a single tag, and so adding a new dataset (a new data element to each row) could destroy the well-formed tag of the row, requiring the tag of the updated row to be computed from scratch. We show that our second scheme allows the client to add new datasets and efficiently update the tag column, *without downloading* m.

In summary our BVC schemes provide all the desirable properties of a VC scheme in practice, together with the unique property that the storage overhead reduces with the number of datasets. The storage efficiency however comes at a somewhat higher cost of computing the proofs by the server. In Sect. 4 we give comparisons of our schemes with [8] that supports the same functionality, when applied to the s datasets individually.

Composition. Our BVC schemes support composition. Let $m = (m_{i,j})_{N \times s}$ be the client's outsourced data, and $\mathcal{P}_1 = (f_1, I_1), \ldots, \mathcal{P}_n = (f_n, I_n)$ be n programs, where f_i is a function and $I_i \subseteq [N]$ for every $i \in [n]$. Computing the n programs on the datasets gives a matrix $\rho = (\rho_{i,j})_{n \times s}$ of computation results and n proofs $\pi_1, \ldots, \pi_n$, where $\rho_{i,j}$ is the result of computing f_i on the jth dataset and π_i is a proof of the correctness of the ith row of ρ. Our schemes allow composition in the sense that there is a polynomial time algorithm Comb that takes $(\rho, (\pi_1, \ldots, \pi_n))$ and any program $\mathcal{P} = (f(x_1, \ldots, x_n), I = [n])$ as input, and outputs $\xi_1 = f(\rho_{1,1}, \ldots, \rho_{n,1}), \ldots, \xi_s = f(\rho_{1,s}, \ldots, \rho_{n,s})$ along with a batch proof π. Moreover, the client's cost to verify $\xi_1, \ldots, \xi_s$ is substantially less than what is required by computing $\xi_1, \ldots, \xi_s$.

1.2 Overview of the Constructions

We use a novel interpretation of the technique in [8] when applied to multiple datasets to design schemes that satisfy properties (p1)–(p5). Let $m = (m_{i,j})_{N \times s}$ be a collection of s datasets that are to be outsourced. We shall authenticate the s elements in each row of m using a single authentication tag that has size equal to an entry of m. This immediately results in a storage overhead of $1 + 1/s$. The N tags are generated such that the cloud server can compute any program $\mathcal{P} = (f, I)$ on the s datasets, and also produce a single proof that verifies the correctness of *all* s computation results. The main idea is a generalization of the technique of [8] to s elements. We pick a *curve* (or a *plane*)

σ_i that passes through the s points determined by the s elements in the ith row of m and also a point determined by a pseudorandom value $F_k(i)$, where F is a pseudorandom function; the stored tag is a single field element that can be used by the server to determine σ_i; the computations of any program $\mathcal{P} = (f, I)$ on all the s outsourced datasets can be efficiently verified using the once computation of f on the pseudorandom values $\{F_k(i) : i \in I\}$.

In the first scheme, the client picks a secret key $sk = (k, a) \leftarrow \mathcal{K} \times (\mathbb{Z}_p \setminus \{0, 1, \ldots, s\})$ and determines a univariate polynomial $\sigma_i(x)$ of degree $\leq s$ that passes through the $s + 1$ points $(1, m_{i,1}), \ldots, (s, m_{i,s})$ and $(a, F_k(i))$, for every $i \in [N]$. The client takes the coefficient of x^s in $\sigma_i(x)$ as the tag t_i that authenticates all data elements in the ith row of m, i.e., $m_{i,1}, \ldots, m_{i,s}$. The client stores $pk = (m, \mathbf{t} = (t_1, \ldots, t_N))$ on the cloud server. Let $\mathcal{P} = (f, I)$ be a program where $f(x_1, \ldots, x_n)$ is a polynomial, and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies the elements of each dataset that are used in the computation of f. Given the program $\mathcal{P}$, the server returns both the s computation results $\rho_1 = f(m_{i_1,1}, \ldots, m_{i_n,1}), \ldots, \rho_s = f(m_{i_1,s}, \ldots, m_{i_n,s})$ and a proof $\pi = f(\sigma_{i_1}(x), \ldots, \sigma_{i_n}(x))$. The client accepts all s results only if $\pi(j) = \rho_j$ for every $j \in [s]$ and $\pi(a) = f(F_k(i_1), \ldots, F_k(i_n))$. In the second scheme, the client picks a secret key $sk = (k, \mathbf{a} = (a_0, a_1, \ldots, a_s)) \leftarrow \mathcal{K} \times (\mathbb{Z}_p^*)^{s+1}$ and determines an $(s+1)$-variate polynomial $\sigma_i(\mathbf{y}) = \sigma_i(y_0, y_1, \ldots, y_s) = t_i \cdot y_0 + m_{i,1} \cdot y_1 + \cdots + m_{i,s} \cdot y_s$ that passes through the $s+1$ points $(\mathbf{e}_2, m_{i,1}), \ldots, (\mathbf{e}_{s+1}, m_{i,s})$ and $(\mathbf{a}, F_k(i))$ for every $i \in [N]$, where $\mathbf{e}_j \in \mathbb{Z}_p^{s+1}$ is a 0–1 vector whose jth entry is equal to 1 and all other entries are equal to 0. The client stores $pk = (m, \mathbf{t} = (t_1, \ldots, t_N))$ on the cloud server. Given the program $\mathcal{P} = (f, I)$, the server returns both the s computation results $\rho_1, \ldots, \rho_s$ and a proof $\pi = f(\sigma_{i_1}(\mathbf{y}), \ldots, \sigma_{i_n}(\mathbf{y}))$. The client accepts all s results only if $\pi(\mathbf{e}_{j+1}) = \rho_j$ for every $j \in [s]$ and $\pi(\mathbf{a}) = f(F_k(i_1), \ldots, F_k(i_n))$.

In both schemes the server's computation consists of PRF computations and polynomial evaluations over a relatively small finite field $\mathbb{Z}_p$. In Sect. 4 we will show that the first scheme admits computation of polynomials of degree as high as any polynomial in the security parameter λ, and the second scheme admits computation of any constant-degree polynomials where the constant however can be much larger than two. In both schemes, the client's complexity of verifying all s computation results is dominated by the once computation of f on the n pseudorandom values $F_k(i_1), \ldots, F_k(i_n)$. In particular, this complexity becomes substantially less than the complexity incurred by the s outsourced computations on datasets when the number s is large enough. In both of our schemes the s datasets of size N contained in m are authenticated using a single vector $\mathbf{t}$ of N tags, where each tag is a single field element. As a consequence, the storage overheads of our schemes are both equal to $(|m| + |\mathbf{t}|)/|m| = (Ns + N)/(Ns) = 1 + 1/s$, which can be arbitrarily close to the lower bound 1 as long as s is large enough. Hence, our schemes achieve the properties (p1)–(p4).

In our schemes, a malicious cloud server may want to deceive the client into accepting some wrong results $(\bar{\rho}_1, \ldots, \bar{\rho}_s) \neq (\rho_1, \ldots, \rho_s)$ with a forged proof $\bar{\pi}$. In the first scheme, the forged proof $\bar{\pi}$, as the correct proof π, is a univariate polynomial of degree $\leq d_1 = s \cdot \deg(f)$. The malicious server succeeds only

if $(\bar{\pi}(1),\ldots,\bar{\pi}(s)) = (\bar{\rho}_1,\ldots,\bar{\rho}_s) \neq (\rho_1,\ldots,\rho_s) = (\pi(1),\ldots,\pi(s))$ and $\bar{\pi}(a) = f(F_k(i_1),\ldots,F_k(i_n)) = \pi(a)$. Let $\bar{\pi} - \pi = u_0 + u_1 x + \cdots + u_{d_1} x^{d_1}$ and $\mathbf{a} = (1, a, \ldots, a^{d_1})$. Then breaking the security of our first scheme is equivalent to finding a non-zero vector $\mathbf{u} = (u_0,\ldots,u_{d_1})$ such that the inner product $\mathbf{u}\cdot\mathbf{a} = 0$. In the second scheme, the forged proof $\bar{\pi}$, as the correct proof π, is an $(s+1)$-variate polynomial of degree $\leq d_2 = \deg(f)$. The malicious server succeeds only if $(\bar{\pi}(\mathbf{e}_2),\ldots,\bar{\pi}(\mathbf{e}_{s+1})) = (\bar{\rho}_1,\ldots,\bar{\rho}_s) \neq (\rho_1,\ldots,\rho_s) = (\pi(\mathbf{e}_2),\ldots,\pi(\mathbf{e}_{s+1}))$ and $\bar{\pi}(\mathbf{a}) = f(F_k(i_1),\ldots,F_k(i_n)) = \pi(\mathbf{a})$, where $\mathbf{a} = (a_0,\ldots,a_s)$. Let $\bar{\pi} - \pi$ have coefficient vector $\mathbf{u} \in \mathbb{Z}_p^h$ and let $\boldsymbol{\alpha} = \langle \mathbf{a}^{\mathbf{i}} : \mathrm{wt}(\mathbf{i}) \leq d_2 \rangle \in \mathbb{Z}_p^h$, where $h = \binom{s+1+d_2}{d_2}$ and $\mathbf{a}^{\mathbf{i}} = a_0^{i_0} a_1^{i_1} \cdots a_s^{i_s}$ for every $\mathbf{i} = (i_0, i_1, \ldots, i_s)$. Then breaking the security of our second scheme is equivalent to finding a non-zero vector $\mathbf{u}$ such that $\mathbf{u}\cdot\boldsymbol{\alpha} = 0$. In Sect. 2, we provide a technical lemma that shows the probability that any adversary finds such a vector $\mathbf{u}$ in both schemes is negligible in λ and thus the security proofs follow.

In both schemes, client can easily authenticate an arbitrary number of new rows using the same secret key and thus extend the size of all datasets. The second scheme also allows the number of datasets to be increased. To add a new dataset $(m_{1,s+1},\ldots,m_{N,s+1})$, the client picks $(k', a_{s+1}) \leftarrow \mathcal{K} \times \mathbb{Z}_p^*$, and sends both $(m_{1,s+1},\ldots,m_{N,s+1})$ and $(\Delta_1,\ldots,\Delta_N)$ to the cloud server, where $\Delta_i = a_0^{-1}(F_k(i) - F_{k'}(i) + a_{s+1}\cdot m_{i,s+1})$ for every $i \in [N]$. The cloud server will update m to $(m_{i,j})_{N\times(s+1)}$ and update $\mathbf{t}$ to $\mathbf{t}' = (t'_1,\ldots,t'_N)$, where $t'_i = t_i - \Delta_i$ for every $i \in [N]$. Intuitively, doing so reveals no information about $\mathbf{a}' = (a_0,\ldots,a_s,a_{s+1})$ to the cloud server. The t'_i is computed such that $\sigma'_i(y_0,\ldots,y_{s+1}) = t'_i\cdot y_0 + m_{i,1}\cdot y_1 + \cdots + m_{i,s+1}\cdot y_{s+1}$ passes through $(\mathbf{a}', F_k(i)), (\mathbf{e}_2, m_{i,1}), \ldots, (\mathbf{e}_{s+2}, m_{i,s+1})$. Thus, all the algorithms of the second scheme will work well with the new secret key $sk' = (k', \mathbf{a}')$. We show that breaking the security of this extended scheme is equivalent to finding a non-zero vector $\mathbf{u}$ such that $\mathbf{u}\cdot\boldsymbol{\alpha}' = 0$, where $\boldsymbol{\alpha}' = \langle (\mathbf{a}')^{\mathbf{i}} : \mathrm{wt}(\mathbf{i}) \leq d_2 \rangle$. We show that this cannot be done except with negligible probability. *Thus the second scheme also satisfies (p5).*

In both schemes, the composition property follows from the intrinsic structure of the constructions. Let $\mathcal{P}_1 = (f_1, I_1), \ldots, \mathcal{P}_n = (f_n, I_n)$ be n programs. In the first scheme the cloud server would compute these programs on $pk = (m, \mathbf{t})$ and then obtain a matrix $(\rho_{i,j})_{n\times s}$ of results and n proofs $(\pi_1,\ldots,\pi_n)$. Given any high level program $\mathcal{P} = (f(x_1,\ldots,x_n), I = [n])$, the cloud server would be able to compute $\mathcal{P}$ on $(\rho_{i,j})_{n\times s}$ to obtain s results $\xi_1,\ldots,\xi_s$ and also compute $\mathcal{P}$ on $(\pi_1,\ldots,\pi_n)$ to obtain a proof $\pi = f(\pi_1,\ldots,\pi_n)$.

1.3 Related Work

The problem of securely outsourcing computation has a long history. We refer the readers to [5,14] for the solutions that require strong assumptions on adversaries, and the theoretical solutions [19] that require interaction. We are only interested in the non-interactive solutions in the standard model.

Verifiable Computation. The verifiable computation of Gennaro et al. [14] gave a non-interactive solution for securely outsourcing computation in the stan-

dard model. The VC schemes of [2,11,14] can delegate any generic functions but have limited practical relevance due to their use of fully homomorphic encryption (FHE). The memory delegation [10] can delegate computations on an arbitrary portion of the outsourced data. However, the client must be stateful and suffer from the impracticality of PCP techniques. Benabbas et al. [5] initiated the study of practical (private) VC schemes for delegating specific functions such as polynomials. Parno et al. [21] initiated the study of public VC schemes. Fiore et al. [13] generalized the constructions of [5] and obtained public VC schemes for delegating polynomials and matrices. Papamanthou et al. [20] constructed a public VC scheme for delegating polynomials that allows efficient update. The storage overhead of all these schemes is ≥ 2. Furthermore, they only admit linear computations on the outsourced data. In particular, the multi-function VC [13] has similar setting as ours but only admits linear computations and has storage overhead ≥ 2.

Homomorphic MACs and Signatures. A homomorphic MAC or signature scheme [7,16] allows one to freely authenticate data and then verify computations on the authenticated data. Such schemes give VC: the client can store data elements and their MAC tags (or signatures) with a server such that the server can compute some admissible functions on an arbitrary subset of the data elements; the server provides both the answer and a MAC tag (or signature) vouching for the correctness of its answer. The storage overhead of the resulting VC scheme is ≥ 2. Catalano and Fiore [8] proposed a practical HomMAC that admits polynomials of degree as high as a polynomial in the security parameter. However, the client's verification requires as much time as the delegated computation. Backes, Fiore and Reischuk [4] proposed a HomMAC that has amortized efficient verification but only admits polynomials of degree ≤ 2.

Non-interactive Proofs. Goldwasser et al. [18] gave a non-interactive scheme for delegating NC computations. However, for any circuit of size n, the server's running time may be a high degree polynomial of n and thus not practical. The SNARGs/SNARKs of [3,6,15] give non-interactive schemes for delegating computations. However, they must rely on the non-falsifiable assumptions [17] which are not standard and much stronger than the common assumptions such as the existence of secure PRFs we use in this paper.

1.4 Organization

In Sect. 2 we provide a formal definition of batch verifiable computation and its security; we also develop a lemma which will be used in our security proofs; In Sect. 3 we present our BVC schemes; In Sect. 4, we give a detailed analysis of the proposed schemes and compare them with the solutions based on [4,8]; we also discuss extra properties of our schemes such as composition; Sect. 5 contains some concluding remarks.

2 Preliminaries

Let λ be a security parameter. We say that a function $q(\lambda)$ is a polynomial function of λ, denoted as $q(\lambda) = \mathsf{poly}(\lambda)$, if there is a real number $c > 0$ such that $q(\lambda) = O(\lambda^c)$; we say that a function $\epsilon(\lambda)$ is a negligible function of λ, denoted as $\epsilon(\lambda) = \mathsf{neg}(\lambda)$, if $\epsilon(\lambda) = o(\lambda^{-c})$ for any real number $c > 0$. Let $\mathcal{A}(\cdot)$ be a probabilistic polynomial time (PPT) algorithm. The symbol "$y \leftarrow \mathcal{A}(x)$" means that y is the output distribution of running algorithm $\mathcal{A}$ on the input x. We denote by $\mathbf{u} = \langle u_x : x \in X \rangle$ any vector whose entries are labeled by elements of the finite set X.

2.1 Batch Verifiable Computation on Outsourced Data

In this section we formally define the notion of batch verifiable computation on outsourced data. In our model, the client has a set of data elements and stores them on the cloud server. The set is organized as a matrix $m = (m_{i,j})_{N \times s}$, where each element $m_{i,j}$ is labeled with a pair $(i, j) \in [N] \times [s]$. Each column of m is called a *dataset*. Let $\mathcal{F}$ be any admissible function family. The client is interested in delegating the computation of some function $f(x_1, \ldots, x_n) \in \mathcal{F}$ on the n elements labeled by $I = \{i_1, \ldots, i_n\} \subseteq [N]$, of every dataset. In other words, the client is interested in learning $\rho_1 = f(m_{i_1,1}, \ldots, m_{i_n,1}), \rho_2 = f(m_{i_1,2}, \ldots, m_{i_n,2}), \ldots, \rho_s = f(m_{i_1,s}, \ldots, m_{i_n,s})$. We say that such a batch of computations is defined by a *program* $\mathcal{P} = (f, I) \in \mathcal{F} \times 2^{[N]}$.

Definition 1 (Batch Verifiable Computation). *A BVC scheme for $\mathcal{F}$ is a tuple $\Pi = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ of four polynomial-time algorithms, where*

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, m)$ *is a key generation algorithm that takes as input the security parameter λ and a set $m = (m_{i,j})_{N \times s}$ of data elements and outputs a secret key sk and a public key pk;*
- $vk \leftarrow \mathsf{ProbGen}(sk, \mathcal{P})$ *is a problem generation algorithm that takes as input sk, a program $\mathcal{P} = (f, I) \in \mathcal{F} \times 2^{[N]}$ and outputs a verification key vk;*
- $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \mathcal{P})$ *is a computation algorithm that takes as input pk and a program $\mathcal{P} = (f, I) \in \mathcal{F} \times 2^{[N]}$ and outputs an answer $\rho = (\rho_1, \ldots, \rho_s)$ and a proof π; and*
- $\{0, 1\} \leftarrow \mathsf{Verify}(sk, vk, (\rho, \pi))$ *is a verification algorithm that verifies ρ with (sk, vk, π); it outputs 1 (to indicate acceptance) or 0 (to indicate rejection).*

In our BVC model, the client generates $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, m)$ and gives pk to the server. To compute some program $\mathcal{P} = (f, I)$ on the outsourced data, the client generates $vk \leftarrow \mathsf{ProbGen}(sk, \mathcal{P})$ and gives $\mathcal{P}$ to the server. Given $(pk, \mathcal{P})$, the server computes and replies with $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \mathcal{P})$. At last, the client accepts ρ only if $\mathsf{Verify}(sk, vk, (\rho, \pi)) = 1$.

Correctness. This property requires that the client always accepts the results computed by an honest server (using the algorithm $\mathsf{Compute}$). Formally, the

- SETUP. Given m, the challenger computes $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, m)$ and gives pk to $\mathcal{A}$;
- QUERIES. The adversary $\mathcal{A}$ adaptively makes a polynomial number of queries:
  For every $\ell = 1$ to $q = \mathsf{poly}(\lambda)$,
  - The adversary $\mathcal{A}$ picks a program $\mathcal{P}_\ell$ and gives it to the challenger;
  - The challenger computes $vk_\ell \leftarrow \mathsf{ProbGen}(sk, \mathcal{P}_\ell)$;
  - The adversary $\mathcal{A}$ constructs a response $(\bar{\rho}_\ell, \bar{\pi}_\ell)$ to the challenger;
  - The challenger gives the output $b_\ell = \mathsf{Verify}(sk, vk, (\bar{\rho}_\ell, \bar{\pi}_\ell))$ to $\mathcal{A}$.
- FORGERY. The adversary $\mathcal{A}$ picks a program $\mathcal{P}^*$ and gives it to the challenger. The challenger computes $vk^* \leftarrow \mathsf{ProbGen}(sk, \mathcal{P}^*)$. At last, $\mathcal{A}$ constructs a response $(\bar{\rho}^*, \bar{\pi}^*)$ to the challenger.
- OUTPUT. The challenger computes $(\rho^*, \pi^*) \leftarrow \mathsf{Compute}(pk, \mathcal{P}^*)$. The adversary wins the security game if $\mathsf{Verify}(sk, vk, (\bar{\rho}^*, \bar{\pi}^*)) = 1$ but $\bar{\rho}^* \neq \rho^*$.

Fig. 1. Security game

scheme Π is *correct* if for any data $m = (m_{i,j})$, any $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, m)$, any program $\mathcal{P}$, any $vk \leftarrow \mathsf{ProbGen}(sk, \mathcal{P})$ and any $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \mathcal{P})$, it holds that $\mathsf{Verify}(sk, vk, (\rho, \pi)) = 1$.

Security. This property requires that no malicious server can deceive the client into accepting any incorrect results. 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.

REMARKS: (1) In the FORGERY phase the adversary $\mathcal{A}$ behaves just like it has done in any one of the q queries. Without loss of generality, we can suppose $(\mathcal{P}^*, \bar{\rho}^*, \bar{\pi}^*) = (\mathcal{P}_{\ell^*}, \bar{\rho}_{\ell^*}, \bar{\pi}_{\ell^*})$ for some $\ell^* \in [q]$, i.e., $\mathcal{A}$ picks one of its q queries as the final forgery. (2) In the literature, many VC schemes such as [2,11,14] are not immune to the "rejection problem": if the malicious server knows whether the client has accepted or rejected its answer, then the algorithm KeyGen (requiring heavy computation effort) must be run again to refresh both sk and pk; otherwise, the VC scheme becomes no longer secure. In our security definition, the adversary $\mathcal{A}$ is allowed to make a polynomial number of queries and learns whether some adaptively chosen answers in each query will be accepted by the client. Therefore, the BVC schemes secure under our definition will be immune to the "rejection problem". (3) Our definition of BVC is different from the VC [5] in the sense that we neither consider the outsourced data as a function nor consider the client's input to $\mathsf{ProbGen}$ as an input from that function's domain. In our definition, the client's input to $\mathsf{ProbGen}$ is a program $\mathcal{P} = (f, I) \in \mathcal{F} \times 2^{[N]}$ that specifies the computations of an admissible function f on the portion labeled by I of every dataset. Clearly our definition captures more general scenarios than [5]. In particular, the VC model of [5] can be captured by our BVC as below. Let $m(x)$ be the client's function which will be delegated to the cloud server (e.g.,

$m(x)$ may be a polynomial $m_1 + m_2 x + \cdots + m_N x^{N-1}$ in [5]); from our point of view, the coefficients $(m_1, \ldots, m_N)$ of the polynomial $m(x)$ is a dataset; and furthermore, any input α to the polynomial $m(x)$ specifies a program $\mathcal{P} = (f_\alpha, [N])$, where $f_\alpha(m_1, \ldots, m_N) = m(\alpha)$. Therefore, the polynomial evaluations considered in [5] can be captured by some specific linear computations in our BVC model. (4) In our BVC, the client's verification requires the secret key sk. Thus, our BVC schemes are *privately verifiable*. (5) A critical efficiency measure of the BVC scheme in Definition 1 is to what extent the client's verification requires less computing time (resources) than the delegated computations. The client's verification in [5,9,13,14,20,21] is efficient in the sense that it requires substantially less time than performing the delegated computation. In our BVC, the client performs verification by generating a verification key $vk \leftarrow \mathsf{ProbGen}(sk, \mathcal{P})$ and then running the verification algorithm $\mathsf{Verify}(sk, vk, (\rho, \pi))$. The client's verification time is equal to the total time required for running both algorithms. Let $t_{\mathcal{P}}$ be the time required for computing the program $\mathcal{P}$ on the outsourced data. We say that a BVC scheme is *outsourceable* if the client's verification time is of the order $o(t_{\mathcal{P}})$. In this paper, we shall construct BVC schemes that are outsourceable.

2.2 A Lemma

In this section we present a lemma (Lemma 1) that underlies the security proofs of our BVC schemes. Let λ be a security parameter. Let p be a λ-bit prime and let $\mathbb{Z}_p$ be the finite field of p elements. Let $h \geq 0$ be an integer. We define an equivalence relation $\sim$ over $\mathbb{Z}_p^{h+1} \setminus \{\mathbf{0}\}$ as below: two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_p^{h+1} \setminus \{\mathbf{0}\}$ are said to be *equivalent* if there exists $\xi \in \mathbb{Z}_p \setminus \{0\}$ such that $\mathbf{u} = \xi \cdot \mathbf{v}$. Let $\Omega_{p,h} = (\mathbb{Z}_p^{h+1} \setminus \{\mathbf{0}\})/\sim$ be the set of all equivalence classes. We represent each equivalence class with a vector in that class. Without loss of generality, we agree that the representative of each class in $\Omega_{p,h}$ is chosen such that its first non-zero element is 1. For any $\mathbf{u}, \mathbf{v} \in \Omega_{p,h}$, we define $\mathbf{u} \odot \mathbf{v} = 0$ if the inner product of $\mathbf{u}$ and $\mathbf{v}$ is equal to 0 modulo p and define $\mathbf{u} \odot \mathbf{v} = 1$ otherwise. The following game models the malicious server's attack in our BVC schemes.

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

In Appendix A, we show the following technical lemma:

Lemma 1. *Let p be a prime and let $d, h, s > 0$ be integers.*

(1) Let $A \subseteq \mathbb{Z}_p$ be a non-empty subset of $\mathbb{Z}_p$. Let $\mathcal{V}_{\mathrm{up}} = \{(1, a, \ldots, a^h) : a \in A\}$. Then any adversary $\mathcal{A}$ wins in $\mathbf{Game}_{\mathcal{V}_{\mathrm{up}}}$ with probability $\leq hq/|A|$.

(2) Let $\mathcal{V}_{\mathrm{mp}} = \{\langle \mathbf{a}^{\mathbf{i}} : \mathrm{wt}(\mathbf{i}) \leq d \rangle : \mathbf{a} \in A^{s+1}\}$, where $h = \binom{s+1+d}{d} - 1$. Then any adversary $\mathcal{A}$ wins in $\mathbf{Game}_{\mathcal{V}_{\mathrm{mp}}}$ with probability $\leq dq/|A|$.

3 Constructions

In this section we propose two BVC schemes for delegating polynomial computations on outsourced data. Our schemes use curves and planes to authenticate the outsourced data, respectively.

3.1 The First Construction

Let p be a λ-bit prime and let $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$ be a PRF with key space $\mathcal{K}$, domain $\{0,1\}^*$ and range $\mathbb{Z}_p$. Let $s > 0$ be an integer. Let $m = (m_{i,j}) \in \mathbb{Z}_p^{N \times s}$ be a matrix that models the client's data. We consider $1, 2, \ldots, s$ as elements of $\mathbb{Z}_p$. Below is our first construction Π_1.

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, m)$: Pick $k \leftarrow \mathcal{K}$ and $a \leftarrow \mathbb{Z}_p \setminus \{0, 1, 2, \ldots, s\}$. For every $i \in [N]$, compute the coefficients of a polynomial $\sigma_i(x) = \sigma_{i,1} + \sigma_{i,2} \cdot x + \cdots + \sigma_{i,s} \cdot x^{s-1} + t_i \cdot x^s$ such that $\sigma_i(j) = m_{i,j}$ for every $j \in [s]$ and $\sigma_i(a) = F_k(i)$. This can be done by solving the following equation system

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^s \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & s & s^2 & \cdots & s^s \\ 1 & a & a^2 & \cdots & a^s \end{pmatrix} \begin{pmatrix} \sigma_{i,1} \\ \sigma_{i,2} \\ \vdots \\ \sigma_{i,s} \\ t_i \end{pmatrix} = \begin{pmatrix} m_{i,1} \\ m_{i,2} \\ \vdots \\ m_{i,s} \\ F_k(i) \end{pmatrix} \tag{1}$$

  for every $i \in [N]$. The algorithm outputs $pk = (m, \mathbf{t})$ and $sk = (k, a)$, where $\mathbf{t} = (t_1, \ldots, t_N)$.
- $vk \leftarrow \mathsf{ProbGen}(sk, \mathcal{P})$: Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial of degree d over $\mathbb{Z}_p$ and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies the data elements on which f should be computed. This algorithm computes and outputs a verification key $vk = f(F_k(i_1), \ldots, F_k(i_n))$.
- $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \mathcal{P})$: Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial of degree d over $\mathbb{Z}_p$ and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies the data elements on which f should be computed. This algorithm computes $\rho_j = f(m_{i_1,j}, \ldots, m_{i_n,j})$ for every $j \in [s]$. It solves the following equation system

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{s-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & s & s^2 & \cdots & s^{s-1} \end{pmatrix} \begin{pmatrix} \sigma_{i,1} \\ \sigma_{i,2} \\ \vdots \\ \sigma_{i,s} \end{pmatrix} = \begin{pmatrix} m_{i,1} - t_i \\ m_{i,2} - 2^s t_i \\ \vdots \\ m_{i,s} - s^s t_i \end{pmatrix} \tag{2}$$

  to determine s coefficients $\sigma_{i,1}, \ldots, \sigma_{i,s}$ for every $i \in I$. Let $\sigma_i(x) = \sigma_{i,1} + \sigma_{i,2} \cdot x + \cdots + \sigma_{i,s} \cdot x^{s-1} + t_i \cdot x^s$. This algorithm outputs $\rho = (\rho_1, \ldots, \rho_s)$ and $\pi = f(\sigma_{i_1}(x), \ldots, \sigma_{i_n}(x))$.
- $\{0,1\} \leftarrow \mathsf{Verify}(sk, vk, (\rho, \pi))$: This algorithm accepts ρ and outputs 1 only if $\pi(a) = vk$ and $\pi(j) = \rho_j$ for every $j \in [s]$.

It is easy to see Π_1 is correct. In the full version we show that no PPT adversary can win in the standard security game (Fig. 1) for Π_1 except with negligible probability. So we have

Theorem 1. *If F is a secure PRF, then Π_1 is a secure BVC scheme.*

3.2 The Second Construction

Let p be a λ-bit prime and let $F : \mathcal{K} \times \{0,1\}^* \to \mathbb{Z}_p$ be a PRF with key space $\mathcal{K}$, domain $\{0,1\}^*$ and range $\mathbb{Z}_p$. Let $s > 0$ be an integer. Let $m = (m_{i,j}) \in \mathbb{Z}_p^{N \times s}$ be a matrix that models the client's data. We consider $1, 2, \ldots, s$ as elements of $\mathbb{Z}_p$. Below is our second construction Π_2.

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, m)$: Pick $k \leftarrow \mathcal{K}$ and $a_0, a_1, \ldots, a_s \leftarrow \mathbb{Z}_p^*$; for every $i \in [N]$, compute

$$t_i = a_0^{-1}(F_k(i) - a_1 \cdot m_{i,1} - \cdots - a_s \cdot m_{i,s}). \tag{3}$$

  This algorithm outputs $pk = (m, \mathbf{t})$ and $sk = (k, \mathbf{a})$, where $\mathbf{t} = (t_1, \ldots, t_N)$ and $\mathbf{a} = (a_0, a_1, \ldots, a_s)$.
- $vk \leftarrow \mathsf{ProbGen}(sk, \mathcal{P})$: Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial of degree d over $\mathbb{Z}_p$ and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies the data elements on which f should be computed. This algorithm computes and outputs a verification key $vk = f(F_k(i_1), \ldots, F_k(i_n))$.
- $(\rho, \pi) \leftarrow \mathsf{Compute}(pk, \mathcal{P})$: Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial of degree d over $\mathbb{Z}_p$ and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies the data elements on which f should be computed. This algorithm computes $\rho_j = f(m_{i_1,j}, \ldots, m_{i_n,j})$ for every $j \in [s]$. Let $\sigma_i(\mathbf{y}) = t_i \cdot y_0 + m_{i,1} \cdot y_1 + \cdots + m_{i,s} \cdot y_s$ for every $i \in I$, where $\mathbf{y} = (y_0, y_1, \ldots, y_s)$. This algorithm outputs s results $\rho = (\rho_1, \ldots, \rho_s)$ and a proof $\pi = f(\sigma_{i_1}(\mathbf{y}), \ldots, \sigma_{i_n}(\mathbf{y}))$.
- $\mathsf{Verify}(sk, vk, (\rho, \pi))$: This algorithm accepts ρ and outputs 1 only if $\pi(\mathbf{a}) = vk$ and $\pi(\mathbf{e}_{j+1}) = \rho_j$ for every $j \in [s]$, where $\mathbf{e}_{j+1} \in \mathbb{Z}_p^{s+1}$ is a 0–1 vector whose $j+1$st component is 1 and all other components are 0.

It is easy to see Π_2 is correct. In the full version we show that no PPT adversary can win the standard security game (Fig. 1) for Π_2 except with negligible probability. So we have

Theorem 2. *If F is a secure PRF, then Π_2 is a secure BVC scheme.*

4 Analysis

In this section we analyze our BVC schemes and compare them with several (naive) solutions based on the existing works [4,8].

Admissible Function Family. In both of our schemes the integer s is allowed to be $O(\lambda)$ to capture the scenario that a large enough number of datasets are

outsourced. In Π_1 the cloud server's computation consists of computing f on s points, solving n equation systems of the form (2) and also computing a proof $\pi = f(\sigma_{i_1}(x), \ldots, \sigma_{i_n}(x))$. On one hand, the first two computations are light for the powerful server. On the other hand, computing the proof π involves some symbolic computation and seems heavy. However, π is a univariate polynomial of degree $\leq sd$. So π can be interpolated given $D = sd+1$ evaluations of π, which requires the computations of f on $O(D) = O(ds)$ points. This work is acceptable for the cloud server even if $d = \mathsf{poly}(\lambda)$. Therefore, Π_1 allows the computation of polynomials of degree d as high as a polynomial in the security parameter. In Π_2 the cloud server's computation consists of computing f on s points and also computing a proof $\pi = f(\sigma_{i_1}(\mathbf{y}), \ldots, \sigma_{i_n}(\mathbf{y}))$. On one hand, the first computation is light for the powerful cloud server. On the other hand, computing the proof π involves some symbolic computation. Note that $f(x_1, \ldots, x_n)$ is of degree d and each of the $(s+1)$-variate polynomials $\sigma_{i_1}(\mathbf{y}), \ldots, \sigma_{i_n}(\mathbf{y})$ is of degree 1. The cost required by computing π is roughly equal to that required by computing f on $(s+1)^d$ points. Furthermore, the server needs to send a representation of π that consists of $\binom{s+1+d}{d}$ field elements. If we allow $s = O(\lambda)$, then degree d must be restricted to $O(1)$ such that the server's computation and communication are not too costly. So Π_2 allows the computation of any $O(1)$-degree polynomials. This admissible function family of $O(1)$-degree polynomials can be significantly larger than the admissible function family of quadratic polynomials in [4].

Efficient Client Verification. Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial function and $I = \{i_1, \ldots, i_n\} \subseteq [N]$. Let (ρ, π) be the results and proof generated by $\mathsf{Compute}$. The verification complexity is measured by the time complexity of running two algorithms: $\mathsf{ProbGen}(sk, \mathcal{P})$ and $\mathsf{Verify}(sk, vk, (\rho, \pi))$. In our schemes, the time complexity of running Verify is independent of n. As we always consider large enough n, the verification complexity in both of our schemes will be dominated by the time complexity of running $\mathsf{ProbGen}(sk, \mathcal{P})$. This is the complexity of computing f on n pseudorandom values $F_k(i_1), \ldots, F_k(i_n)$ once. Note that this computation requires roughly $1/s$ times as much time as that required by the s delegated computations of f on the outsourced data. Whenever s is large enough, the client's verification per each dataset uses substantially less time than computing f on each dataset. Hence, our schemes are outsourceable.

Efficient Server Computation. In our schemes, the cloud server's computation only involves PRF computations and polynomial evaluations over the finite field $\mathbb{Z}_p$. Note that we never need any number-theoretic assumptions. As a result, the size of the finite field $\mathbb{Z}_p$ can be chosen as small as $p \approx 2^{128}$ when the security parameter $\lambda = 128$. In particular, the PRF F in our both constructions can be chosen as some heuristic PRFs such as AES block ciphers in practical implementations. In Sect. 4.3 we shall see that our server's computation is significantly more efficient than [4].

Efficient Server Storage. The storage overheads of our schemes are equal to $|pk|/|m|$, where $|pk|$ and $|m|$ denote the numbers of field elements contained

in pk and m respectively. Recall that the number $|pk|/|m|$ is always ≥ 1 and our objective is making it as close to 1 as possible. It is trivial to see that $|pk|/|m| = (|m| + |\mathbf{t}|)/|m| = (Ns + N)/Ns = 1 + 1/s$ in our schemes. Therefore, the storage overheads of our schemes can be made arbitrarily close to 1 as long as s is large enough.

Extending the Size of Datasets. In our schemes the client's outsourced data is a collection $m = (m_{i,j})_{N\times s}$ of s datasets, each containing N elements. In practice, the client may add new data elements to the outsourced datasets. Let $\Pi = \Pi_1$ or Π_2. Let (pk, sk) be any public key and secret key generated by $\Pi.\mathsf{KeyGen}(1^\lambda, m)$. Note that pk takes the form $(m, \mathbf{t} = (t_1, \ldots, t_N))$, where t_i is a tag authenticating the elements $(m_{i,1}, \ldots, m_{i,s})$ for every $i \in [N]$. In particular, the tag t_i is computed using (1) when $\Pi = \Pi_1$ and using (3) when $\Pi = \Pi_2$, respectively. Let $N' = N + 1$. To add s new elements $(m_{N',1}, \ldots, m_{N',s})$ to the s datasets, the client can simply compute a tag $t_{N'}$ authenticating these elements and instruct the cloud server to change $pk = (m, \mathbf{t})$ to $pk' = (m', \mathbf{t}')$, where $m' = (m_{i,j})_{N'\times s}$ and $\mathbf{t}' = (t_1, \ldots, t_{N'})$. In particular, when $\Pi = \Pi_1$, the tag $t_{N'}$ will computed by solving the equation system (1) for $i = N'$; and when $\Pi = \Pi_2$, the tag $t_{N'}$ will be computed using the equation (3) for $i = N'$. Extending the size of all datasets in this way will never compromise the security of the underlying schemes.

Extending the Number of Datasets in Π_2. In practice, the client may also want to extend the number of datasets. Let $s' = s + 1$. We consider the scenario of the client updating m to $m' = (m_{i,j})_{N\times s'}$, where $(m_{1,s'}, \ldots, m_{N,s'})$ is a new dataset. The general case for adding more than one new datasets can be done by adding one after the other. In a naive way of updating m to m', the client may simply download $pk = (m, \mathbf{t})$, verify the integrity of m and then run our schemes on m'. However, this method will be quite inefficient when the size of m is large. Here we show how the client in Π_2 can authenticate m' without downloading m.

Let $F : \mathcal{K}\times\{0,1\}^* \to \mathbb{Z}_p$ be the PRF and let $sk = (k, \mathbf{a}) \leftarrow \mathcal{K}\times(\mathbb{Z}_p^*)^{s+1}$ be the secret key used to outsource $m = (m_{i,j})_{N\times s}$ in Π_2. Let $pk = (m, \mathbf{t})$, where $t_i = a_0^{-1}(F_k(i) - a_1 \cdot m_{i,1} - \cdots - a_s \cdot m_{i,s})$ for every $i \in [N]$. Let $(m_{1,s+1}, \ldots, m_{N,s+1})$ be a new dataset. To authenticate $m' = (m_{i,j})_{N\times s'}$, the client picks $(k', a_{s+1}) \leftarrow \mathcal{K} \times \mathbb{Z}_p^*$, updates sk to $sk' = (k', \mathbf{a}' = (a_0, \ldots, a_s, a_{s+1}))$ and instructs the server to change pk to $pk' = (m', \mathbf{t}' = (t'_1, \ldots, t'_N))$, where $t'_i = a_0^{-1}(F_{k'}(i) - a_1 \cdot m_{i,1} - \cdots - a_{s+1} \cdot m_{i,s+1}) = t_i - a_0^{-1} \cdot (F_k(i) - F_{k'}(i) + a_{s+1} \cdot m_{i,s+1})$. To do so, the client only needs to send the new dataset $(m_{1,s+1}, \ldots, m_{N,s+1})$ together with $\Delta_i = a_0^{-1}(F_k(i) - F_{k'}(i) + a_{s+1} \cdot m_{i,s+1}), 1 \leq i \leq N$, to the cloud server such that the server can update t_i to t'_i by computing $t'_i = t_i - \Delta_i$ for every $i \in [N]$. All the other algorithms will be changed as below to work with (sk', pk'):

- $vk \leftarrow \mathsf{ProbGen}(sk', \mathcal{P})$: Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial of degree d over $\mathbb{Z}_p$ and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies on which elements of each dataset f should be computed. This algorithm computes and outputs a verification key $vk = f(F_{k'}(i_1), \ldots, F_{k'}(i_n))$.

- $(\rho, \pi) \leftarrow \mathsf{Compute}(pk', \mathcal{P})$: Let $\mathcal{P} = (f, I)$ be a program, where $f(x_1, \ldots, x_n)$ is a polynomial of degree d over $\mathbb{Z}_p$ and $I = \{i_1, \ldots, i_n\} \subseteq [N]$ specifies on which elements of each dataset f should be computed. This algorithm computes $\rho_j = f(m_{i_1,j}, \ldots, m_{i_n,j})$ for every $j \in [s+1]$. Let $\sigma_i(\mathbf{y}) = t'_i \cdot y_0 + m_{i,1} \cdot y_1 + \cdots + m_{i,s} \cdot y_s + m_{i,s+1} \cdot y_{s+1}$ for every $i \in I$, where $\mathbf{y} = (y_0, y_1, \ldots, y_s, y_{s+1})$. This algorithm outputs $\rho = (\rho_1, \ldots, \rho_{s+1})$ and a proof $\pi = f(\sigma_{i_1}(\mathbf{y}), \ldots, \sigma_{i_n}(\mathbf{y}))$.
- $\mathsf{Verify}(sk', vk, (\rho, \pi))$: This algorithm accepts ρ and outputs 1 only if $\pi(\mathbf{a}') = vk$ and $\pi(\mathbf{e}_{j+1}) = \rho_j$ for every $j \in [s+1]$.

We say that these modifications resulting in an extended scheme Π'_2. It is trivial to verify the correctness of Π'_2. In the full version we show that no PPT adversary can win a slight modification of the standard security game (Fig. 1) for Π'_2 except with negligible probability, where the modification means that the adversary is allowed to know two tag vectors $\mathbf{t}$ and $\mathbf{t}'$ instead of one.

Theorem 3. *If F is a secure PRF, then Π'_2 is a secure BVC scheme.*

Composition. We now show that our BVC schemes allow composition and the composed computations can be efficiently verified as well. Let $\Pi = \Pi_1$ or Π_2. Let $m = (m_{i,j})_{N \times s} \in \mathbb{Z}_p^{N \times s}$ be a collection of s datasets. Let pk and sk be any public key and secret key generated by $\Pi.\mathsf{KeyGen}(1^\lambda, m)$. Let $\mathcal{P}_1 = (f_1, I_1), \ldots, \mathcal{P}_n = (f_n, I_n)$ be n programs, where $f_i \in \mathcal{F}$ and $I_i \subseteq [N]$. Let $vk_i = f_i(\langle F_k(j) : j \in I_i \rangle)$ be generated by $\Pi.\mathsf{ProbGen}(sk, \mathcal{P}_i)$ for every $i \in [n]$. Let $((\rho_{i,1}, \ldots, \rho_{i,s}), \pi_i) \leftarrow \Pi.\mathsf{Compute}(pk, \mathcal{P}_i)$ be the results and proof generated by the computing algorithm. We can consider $\rho = (\rho_{i,\ell})_{n \times s}$ as a collection of s new datasets and consider $(\rho, \{\pi_i\}_{i=1}^n)$ as an encoding of ρ. Let $\mathcal{P} = (f(x_1, \ldots, x_n), I = [n])$ be a program that defines a computation on ρ.

If $\Pi = \Pi_1$, we have that $sk = (k, a) \in \mathcal{K} \times (\mathbb{Z}_p \setminus \{0, 1, \ldots, s\})$ and $pk = (m, \mathbf{t})$. Due to the correctness of Π_1, we have that $\mathsf{Verify}(sk, vk_i, \{\rho_{i,\ell}\}_{\ell \in [s]}, \pi_i) = 1$ for every $i \in [n]$, that is, $\pi_i(1) = \rho_{i,1}, \pi_i(2) = \rho_{i,2}, \ldots, \pi_i(s) = \rho_{i,s}$ and $\pi_i(a) = vk_i$. Below is the combing algorithm:

- $((\xi_1, \ldots, \xi_s), \pi) \leftarrow \mathsf{Comb}(f, (\rho_{i,\ell})_{n \times s}, \{\pi_i\}_{i \in [n]})$: computes $\xi_\ell = f(\rho_{1,\ell}, \ldots, \rho_{n,\ell})$ for every $\ell \in [s]$ and $\pi = f(\pi_1(x), \ldots, \pi_n(x))$. Outputs $\xi_1, \ldots, \xi_s$ and π.

If $\Pi = \Pi_2$, we have that $sk = (k, \mathbf{a}) \in \mathcal{K} \times (\mathbb{Z}_p^*)^{s+1}$ and $pk = (m, \mathbf{t})$. Due to the correctness of Π_2, we have $\mathsf{Verify}(sk, vk_i, \{\rho_{i,\ell}\}_{\ell \in [s]}, \pi_i) = 1$ for every $i \in [n]$, that is, $\pi_i(\mathbf{e}_2) = \rho_{i,1}, \pi_i(\mathbf{e}_3) = \rho_{i,2}, \ldots, \pi_i(\mathbf{e}_{s+1}) = \rho_{i,s}$ and $\pi_i(\mathbf{a}) = vk_i$. Below is the combing algorithm:

- $((\xi_1, \ldots, \xi_s), \pi) \leftarrow \mathsf{Comb}(f, (\rho_{i,\ell})_{n \times s}, \{\pi_i\}_{i \in [n]})$: computes $\xi_\ell = f(\rho_{1,\ell}, \ldots, \rho_{n,\ell})$ for every $\ell \in [s]$ and $\pi = f(\pi_1(\mathbf{y}), \ldots, \pi_n(\mathbf{y}))$. Outputs $\xi_1, \ldots, \xi_s$ and π.

5 Concluding Remarks

We introduced a model for batch verifiable computation and constructed two BVC schemes with attractive properties. Extending the first scheme to support

efficient outsourcing of new datasets, expanding the admissible function family of the second scheme, and constructing publicly verifiable batch computation schemes are interesting open problems that follow from this work.

Acknowledgement. Liang Feng Zhang's research is currently supported by ShanghaiTech University's start-up funding (No. F-0203-15-001). This work was done when the author was a postdoctoral fellow at the University of Calgary. Reihaneh Safavi-Naini's research is supported in part by Alberta Innovates Technology Futures, in the Province of Alberta, Canada.

A Proof of Lemma 1

Our proof of Lemma 1 begins with the following lemma from [22].

Lemma 2 (Zhang and Safavi-Naini [22]). *If there is a number* $0 < \epsilon < 1$ *such that* $|\{\mathbf{v} \in \mathcal{V} : \mathbf{u} \odot \mathbf{v} = 0\}| \leq \epsilon \cdot |\mathcal{V}|$ *for every* $\mathbf{u} \in \Omega_{p,h}$*, then* $\mathcal{A}$ *wins in the* $\mathbf{Game}_{\mathcal{V}}$ *with probability* $\leq \epsilon q$*.*

Example 1. Let $A \subseteq \mathbb{Z}_p$ be a non-empty subset of $\mathbb{Z}_p$. Let $\mathcal{V}_{\text{up}} = \{(1, a, \ldots, a^h) : a \in A\} \subseteq \Omega_{p,h}$. For any $\mathbf{u} = (u_0, u_1, \ldots, u_h) \in \Omega_{p,h}$ and $\mathbf{v} = (1, a, \ldots, a^h) \in \mathcal{V}_{\text{up}}$, $\mathbf{u} \odot \mathbf{v} = 0$ if and only if a is a root of the polynomial $u_0 + u_1 x + \cdots + u_h x^h$. Note that any non-zero univariate polynomial of degree $\leq h$ has $\leq h$ roots in $\mathbb{Z}_p$ (and thus has $\leq h$ roots in A). For any $\mathbf{u} \in \Omega_{p,h}$, there are $\leq h$ elements $\mathbf{v} \in \mathcal{V}_{\text{up}}$ such that $\mathbf{u} \odot \mathbf{v} = 0$. It follows that $\epsilon \triangleq \max_{\mathbf{u}} \frac{|\{\mathbf{v} \in \mathcal{V}_{\text{up}} : \mathbf{u} \odot \mathbf{v} = 0\}|}{|\mathcal{V}_{\text{up}}|} \leq \frac{h}{|A|}$.

Let $s > 0$ be an integer. Let $\mathbb{Z}_p[\mathbf{y}]$ be the ring of polynomials in $\mathbf{y} = (y_0, y_1, \ldots, y_s)$ with coefficients from $\mathbb{Z}_p$. For any vector $\mathbf{i} = (i_0, i_1, \ldots, i_s)$ of non-negative integers, we denote $\mathbf{y}^{\mathbf{i}} = y_0^{i_0} y_1^{i_1} \cdots y_s^{i_s}$. We define the weight of $\mathbf{i}$ to be $\text{wt}(\mathbf{i}) = i_0 + i_1 + \cdots + i_s$. Then $\mathbf{y}^{\mathbf{i}}$ is a monomial of total degree $\text{wt}(\mathbf{i})$.

Definition 2 (Hasse Derivative). *For any polynomial* $P(\mathbf{y}) \in \mathbb{Z}_p[\mathbf{y}]$ *and any vector* $\mathbf{i} = (i_0, i_1, \ldots, i_s)$ *of non-negative integers, the* $\mathbf{i}$*-th* Hasse Derivative *of* $P(\mathbf{y})$*, denoted as* $P^{(\mathbf{i})}(\mathbf{y})$*, is the coefficient of* $\mathbf{w}^{\mathbf{i}}$ *in the polynomial* $P(\mathbf{y} + \mathbf{w}) \in \mathbb{Z}_p[\mathbf{y}, \mathbf{w}]$*, where* $\mathbf{w} = (w_0, w_1, \ldots, w_s)$*.*

Definition 3 (Multiplicity). *For any polynomial* $P(\mathbf{y}) \in \mathbb{Z}_p[\mathbf{y}]$ *and any point* $\mathbf{a} \in \mathbb{Z}_p^{s+1}$*, the* multiplicity *of* P *at* $\mathbf{a}$*, denoted as* $\text{mult}(P, \mathbf{a})$*, is the largest integer* M *such that for any non-negative integer vector* $\mathbf{i} = (i_0, i_1, \ldots, i_s)$ *with* $\text{wt}(\mathbf{i}) < M$*, we have* $P^{(\mathbf{i})}(\mathbf{a}) = 0$ *(if* M *may be taken arbitrarily large, then we set* $\text{mult}(P, \mathbf{a}) = \infty$*).*

It is trivial to see that $\text{mult}(P, \mathbf{a}) \geq 0$ for any polynomial $P(\mathbf{y})$ and any point $\mathbf{a}$. Furthermore, $P(\mathbf{a}) = 0$ if and only if $\text{mult}(P, \mathbf{a}) \geq 1$. The following lemma is from [12] and shows an interesting property of multiplicity.

Lemma 3. *Let* $P(\mathbf{y}) \in \mathbb{Z}_p[\mathbf{y}]$ *be any non-zero polynomial of total degree at most* d*. Then for any finite set* $A \subseteq \mathbb{Z}_p$*, it holds that* $\sum_{\mathbf{a} \in A^{s+1}} \text{mult}(P, \mathbf{a}) \leq d \cdot |A|^s$.

Let $\mathcal{N}_A(P)$ be the number of roots of $P(\mathbf{y})$ in the set A^{s+1}. Recall that any root $\mathbf{a} \in \mathbb{Z}_p^{s+1}$ of $P(\mathbf{y})$ must satisfies the property that $\text{mult}(P, \mathbf{a}) \geq 1$. Then $\mathcal{N}_A(P) \leq \sum_{\mathbf{a} \in A^{s+1}} \text{mult}(P, \mathbf{a})$. Lemma 3 in particular implies that $\mathcal{N}_A(P) \leq d \cdot |A|^s$ whenever $P(\mathbf{y})$ has total degree at most d. As a generalization of Example 1, we have the following Example related to multivariate polynomials.

Example 2. Let $\mathcal{V}_{\text{mp}} = \{\langle \mathbf{a}^{\mathbf{i}} : \text{wt}(\mathbf{i}) \leq d \rangle : \mathbf{a} \in A^{s+1}\} \subseteq \Omega_{p,h}$, where $h = \binom{s+1+d}{d} - 1$. For any two vectors $\mathbf{u} = \langle u_{\mathbf{i}} : \text{wt}(\mathbf{i}) \leq d \rangle \in \Omega_{p,h}$ and $\mathbf{v} = \langle \mathbf{a}^{\mathbf{i}} : \text{wt}(\mathbf{i}) \leq d \rangle \in \mathcal{V}_{\text{mp}}$, $\mathbf{u} \odot \mathbf{v} = 0$ if and only if $\mathbf{a}$ is a root of the s-variate polynomial $P(\mathbf{y}) = \sum_{\text{wt}(\mathbf{i}) \leq d} u_{\mathbf{i}} \cdot \mathbf{y}^{\mathbf{i}}$. Note that $|\{\mathbf{v} \in \mathcal{V}_{\text{mp}} : \mathbf{u} \odot \mathbf{v} = 0\}| = \mathcal{N}_A(P) \leq d \cdot |A|^s$ and $|\mathcal{V}_{\text{mp}}| = |A|^{s+1}$. Thus, $\epsilon \triangleq \max_{\mathbf{u}} \frac{|\{\mathbf{v} \in \mathcal{V}_{\text{mp}} : \mathbf{u} \odot \mathbf{v} = 0\}|}{|\mathcal{V}_{\text{mp}}|} \leq \frac{d}{|A|}$.

Lemma 2 together with Examples 1 and 2 gives us the technical Lemma 1.

References

1. Agrawal, S., Boneh, D.: Homomorphic MACs: MAC-based integrity for network coding. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 292–305. Springer, Heidelberg (2009)
2. 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)
3. Backes, M., Barbosa, M., Fiore, D., Reischuk, R.M.: ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In: 2015 IEEE Symposium on Security and Privacy (2012)
4. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: 2013 ACM Conference on Computer and Communication Security. ACM Press, November 2013
5. 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)
6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS 2012: Proceedings of the 3rd Symposium on Innovations in Theoretical Computer Science (2012)
7. 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)
8. Catalano, D., Fiore, D.: Practical homomorphic MACs for arithmetic circuits. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 336–352. Springer, Heidelberg (2013)
9. Catalano, D., Fiore, D., Gennaro, R., Vamvourellis, K.: Algebraic (trapdoor) one-way functions and their applications. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 680–699. Springer, Heidelberg (2013)
10. Chung, K.-M., Kalai, Y.T., Liu, F.-H., Raz, R.: Memory delegation. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 151–168. Springer, Heidelberg (2011)

11. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
12. Dvir, Z., Kopparty, S., Saraf, S., Sudan, M.: Extensions to the method of multiplicities, with applications to kakeya sets and mergers. In: FOCS 2009, pp. 181–190 (2009)
13. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: 2012 ACM Conference on Computer and Communication Security. ACM Press, October 2012
14. 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)
15. 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)
16. 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)
17. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 99–108. ACM Press, June 2011
18. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 113–122. ACM Press, May 2008
19. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)
20. 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)
21. 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)
22. Zhang, L.F., Safavi-Naini, R.: Verifiable delegation of computations with storage-verification trade-off. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014, Part I. LNCS, vol. 8712, pp. 112–129. Springer, Heidelberg (2014)