Hindawi
Security and Communication Networks
Volume 2021, Article ID 6612614, 18 pages
https://doi.org/10.1155/2021/6612614

Research Article

An Efficient HPRA-Based Multiclient Verifiable Computation: Transform and Instantiation

Shuaijianni Xu [1,2,3]

[1]*School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China*
[2]*Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China*
[3]*University of Chinese Academy of Sciences, Beijing 100049, China*

Correspondence should be addressed to Shuaijianni Xu; xushjn@shanghaitech.edu.cn

Received 15 November 2020; Revised 15 December 2020; Accepted 8 January 2021; Published 17 February 2021

Academic Editor: Nanrun Zhou

Choi, Katz, Kumaresan, and Cid put forward the conception of multiclient noninteractive verifiable computation (MVC), enabling a group of clients to outsource computation of a function of f. CKKC's MVC is impractical due to their dependence on fully homomorphic encryption (FHE) and garbled circuits (GCs). In this paper, with the goal of satisfying practical requirements, a general transform is presented from the homomorphic proxy re-authenticator (HPRA) of Deler, Ramacher, and Slamanig to MVC schemes. MVC constructions in this particular study tend to be more efficient once the underlying HPRA avoids introducing FHE and GCs. By deploying the transform to DRS's HPRA scheme, a specific MVC scheme for calculating the linear combinations of vectors has been proposed. It can be understood that it is the first feasible and implementable MVC scheme so far, and the instantiation solution has a great advantage in efficiency compared with related works.

1. Introduction

The past several years have witnessed a rapid growth of attention on outsourcing computation due to the popularity of cloud computing: the on-demand availability of computer system resources, including data storage and computing power, without maintaining infrastructure by the client. The proliferation of mobile devices is also one of the reasons why outsourced computing is getting more attention.

Outsourcing computation allows relatively weak devices (phones, tablets, laptops, and PCs) to offload work (storage, image processing, and video encoding) to powerful cloud servers. However, many things can (and do) go wrong in cloud computing scenarios. One must worry about bugs, misconfigurations, operator error, natural disasters, or even malicious cloud servers. The servers may have considerable financial incentives to perform dishonestly, and the servers may offer fast but faulty computations to reduce the occupation of computational resources. In the above cases, the client needs some measures to ensure that computation was processed in the expected way and has not been tampered with. Moreover, when the client's data (e.g., function and inputs or/and outputs) are not encrypted, the servers may misuse the data. How to ensure that the correctness of the calculations and how to ensure that the clients' data are not misused are two critical security topics in the outsourcing computation area.

Gennaro et al. [1] proposed the conception of noninteractive verifiable computation (VC) for the single-client scenario. The VC scheme allows client with weak computing power to outsource the computation task of a function f on a set of inputs $x_1, x_2, \ldots, x_n$ to a server, ensuring that client can detect the malicious or malfunctioning server by verifying the results returned, but their efficiency is problematic due to the dependence on expensive cryptographic primitives, for example, fully homomorphic encryption (FHE) and garbled circuits (GCs). The initial proposal and construction of VC led to a long line of follow-up work, which provided further exploration on optimizing the efficiency of outsourced computations for restricted classes of functions.

Some researchers extended VC into different scenarios. There are, however, scenarios in which it would be meaningful

to extend this functionality to multiclient case. For example, with limited infrastructure, n resource-constrained nodes cannot directly communicate with each other while are only given access to a central server. It may be desirable for the nodes to get evaluation result of a function f over their joint inputs. At this point, only the central server is responsible for collecting the data and reporting the computation results, thereby letting the nodes to adjust the modalities.

Choi et al. [2] initiated the study of this setting, extending the single-client VC model to the multiclient noninteractive verifiable computation (MVC) model. They assumed no client-client communication involved and focused on noninteractive solutions, where each evaluation of the function f required only a single round of communication, i.e., noninteractive. In a single round of communication, n clients send the representation of joint inputs $(x_1^{(1)}, \ldots, x_n^{(1)})$ to the server, and server returns the evaluation result accompanied by a proof. MVC ensures that a malicious server can neither fool any client into accepting a wrong result nor misuse the inputs of the clients. In the scheme, they consider the case where only the first client obtains output (a more general case is dealt with by simply making the clients execute the scheme several times in parallel, with each client playing the role of the first client in one round of execution). They also presented a construction for outsourcing the evaluation of universal Boolean circuits by integrating the scheme of Gennaro et al. [1] with the proxy oblivious transfer. Since FHE and GCs are still the main technical primitives in their construction, the efficiency issue remains unresolved.

Reducing the use of these expensive cryptographic primitives is necessary for constructing a more practical MVC scheme. Unfortunately, if we are limited to the outsourcing of arbitrary computations on confidential data, then using these primitives is somehow unavoidable. In order to go further in efficiency, we sacrifice the generality of the outsourced functions. This paper is primarily concerned with efficient MVC schemes for specific functions, especially for calculating the linear combination of vectors.

1.1. Our Contribution. The main contributions in this paper are twofold. The first one is a proposal of general transform from the homomorphic proxy re-authenticator (HPRA) [3], a tool providing security and verifiability guarantees to the multiuser data aggregation scenarios, to the MVC schemes. The HPRA makes distinct signers authenticate input data under private keys and allows an aggregator to transform all authenticators into an MAC under the receiver's secret key. The evaluation of a function f on the data along with an aggregate authenticated message vector is computed by the aggregator; therefore, the receiver can verify whether the computation is correct or not. Compared with the MVC, there is an additional receiver in the HPRA that obtains and verifies the function output; by contrast, in the MVC, the first client not only performs verification but also provides its own function input.

A very natural idea is thus generated: what will happen if we map the parties in the HPRA into those in the MVC? Taking into account the different properties and functionalities of all the entities, the first step was to merge the receiver with the first signer in the HPRA and then to consider this merged participant as the first client in the MVC. Second, we consider each of the remaining signers play as a different client in the MVC. Third, we regard the aggregator in the HPRA as a cloud server in the MVC. Following this idea, we construct a general transform that can be applied to any HPRA scheme, resulting in an MVC with the following properties:

(i) No malicious server can generate an incorrect output passing the verification of the client, except with a nonnegligible probability.

(ii) For each client, any information on the client's input should not be leaked to the other entities.

(iii) No malicious server can learn anything about the result of the computation beyond what the description of f would leak.

(iv) There is no client-client communication, and all the involved parties do not share a secret key. As an alternative, the clients use independent private keys generated by themselves to encrypt and authenticate the data.

(v) Compared with the HPRA, no extracomputational consumption is introduced. Our MVC construction has nothing to do with FHE and GCs as long as the underlying HPRA avoids introducing these expensive cryptographic primitives.

(vi) Compared with the HPRA, no extracryptographic assumption is introduced. Our MVC construction relies only on the assumptions required in the HPRA.

The second contribution is an MVC scheme implementable by applying a particular HPRA scheme of Derler et al.'s [3] to the transform, which allows n clients to jointly outsource the linear combination of n vectors of length ℓ. It can be seen that it is the first time to implement an MVC scheme.

A model was developed by Parno et al. [4] to analyze the efficiency of the computational performance of the FHE + GC-based VC schemes [1,2]. They estimated that the client would take $\geq 10^{11}$ seconds to outsource the multiplication of two 100 by 100 matrices in the MVC by Choi et al. [2] as our MVC can also be adjusted to outsource the matrix multiplications as well. The experimental results show that, in our MVC scheme instance, calculating the multiplication of two 100 by 100 matrices takes only $\approx 10^3$ seconds at the client side. Our proposed scheme evidently has a great advantage in terms of efficiency.

1.2. Related Works. A single-client noninteractive VC model was presented by Gennaro et al. [1], and an instantiation by considering FHE and GCs as main technical tools was constructed. Parno et al. [4] and Setty et al. [5] gave general-

purpose VC protocols on the basis of the quadratic arithmetic programs (QAPs) [6]. To provide noninteractive, publicly verifiable computation and zero-knowledge proofs, many recent systems [7–10] have converged on the Pinocchio protocol [4] as a cryptographic primitive. Benabbas et al. [11] initiated a line of research about efficient protocols for specific functions. Following this line, a series of subsequent outsourcing computations systems [12–15] developed schemes with improved efficiency for restricted classes of functions. Some research studies [16,17] are dedicated to provide additional security, ensuring that the outsourced polynomial remains hidden.

Choi et al. [2] firstly formed the conception of multiclient noninteractive verifiable computation. In a nutshell, MVC is like VC with an extra feature, allowing multiple clients to jointly outsource a computation under different secret keys. They also proposed an FHE-based construction for outsourcing the computation of arbitrary Boolean circuits by integrating Gennaro et al.'s scheme [1] with proxy oblivious transfer. However, their scheme still uses FHE and GCs as main technical tools and would thus leave the efficiency problem. Moreover, if a malicious server is allowed to send malformed responses and observe the reaction of the first client, the soundness might be threatened. This problem was resolved by Gordon et al. [18], and they provide solutions against a malicious server or multiple colluding clients. However, as they used the falsifiable assumption which is not as mature as the well-known assumptions such as DLOG, CDH, and DDH, they may have potential weakness. The falsifiable assumption introduced the dependence on the circuit depth, and the efficiency is also sacrificed.

Multi-input functional encryption (MIFE) is a generalization of the functional encryption into the case of multi-input functions. MIFE has a great variety of applications related to computation over the encrypted data from multiple sources. However, the construction of MIFE assumed indistinguishability obfuscation (iO) for circuits, which introduces a strong assumption as the work of Goldwasser et al. revealed [19]. Moreover, current MIFE schemes have prohibitively large overhead. Fiore et al. [20] built a multikey homomorphic authenticator (multikey HA), allowing multiple clients to authenticate and outsource a large collection of data, together with the corresponding authenticators, to a malicious server. Backes et al. [12] added a crucial efficiency property for the verification of multikey HAs. Based on the line of multikey HAs, the HPRA introduced by Derler et al. [3] allows a group of signers to authenticate data under private keys and allows an aggregator to transform all the single authenticators into an MAC under the secret key of receiver. Following this research line of multikey HAs, Schabhüser et al. [21] presented a publicly verifiable homomorphic authenticator scheme with efficient and context hiding verification in the case of multiple clients. However, in their scheme, the result of the outsourced computation is public to all entities, thus leading to privacy breaches.

1.3. Organization of the Paper. The preliminaries, such as MVC and HPRA, are highlighted in Section 2. The HPRA to MVC transform and related security proof are presented in Section 3. A concrete construction is then provided in Section 4. The implementation of concrete instantiation and analysis of results are illustrated in Section 5. Conclusions are drawn based on this particular research in Section 6.

2. Preliminaries

To facilitate the comprehension of our work, we give some notations and review some preliminaries pertaining to our research work, namely, multiclient noninteractive verifiable computation and homomorphic proxy re-authenticator.

2.1. Notation. The Greek letter κ stands for the security parameter of schemes. A function $f(\kappa)$ is considered to be negligible in κ if $f(\kappa) = o(\kappa^{-c})$ under every constant $c > 0$, and we denote all such functions as $\mathrm{negl}(\kappa)$ and otherwise denoted as $\mathrm{non-negl}(\kappa)$. When a function can be represented as a polynomial, we use the notation $\mathrm{poly}(\cdot)$. For any $n \in \mathcal{N}$, we refer to $[n]$ as $[n] \coloneqq \{1, \ldots, n\}$. We use $\vec{a}$ to denote a vector $\vec{a} = \{a_1, a_2, \ldots, a_n\}$ and $\mathbf{a}$ to denote a sequence of vectors $\mathbf{a} = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n\}$. For vectors with subscript in their variable name, say $\vec{a}_1$, we use $\vec{a}_1[i]$ to indicate the i-th element in vector $\vec{a}_1$. Given a set S, the notation $s \xleftarrow{R} S$ remains for the process of sampling s from S uniformly.

2.2. Multiclient Verifiable Computation. In an n-party MVC introduced by Choi et al. [2], there are n clients $P_1, P_2, \ldots, P_n$ who expect to outsource the evaluation of some functions over their joint inputs to a server for several times. In the i-th evaluation, client $P_1, P_2, \ldots, P_n$ inputs are denoted as $x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}$, respectively. To ensure the data privacy, the clients encode the original inputs into $\chi_1^{(i)}, \ldots, \chi_n^{(i)}$ and send it to a server along with the encoded function Φ of function f. The server is expected to evaluate $\Phi(\chi_1^{(i)}, \ldots, \chi_n^{(i)})$ and respond with encoded output $\omega^{(i)}$. P_1 is designated to verify the correctness of $\omega^{(i)}$ with a decoding secret ξ and restore the real result $y^{(i)} = f(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ from $\omega^{(i)}$.

For the convenience of follow-up research, we made some reasonable modifications to the MVC model: (1) we replace the security parameter 1^κ with public parameter set pp containing the information of 1^κ and the public outsourcing function f. (2) Let P_1 runs KeyGen_1 algorithm first and let pk_1 be one of the inputs of algorithm KeyGen_j, for $j = 2, \ldots, n$. As MVC has assumed the existence of a public-key infrastructure (PKI), which makes all the public keys of clients be accessible to all other entities. This change makes other client P_j to wait for client P_1 to finish running KeyGen_1 but does not affect security. (3) As EnFunc is run by client P_1, taking P_1's private key sk_1 as an extrainput of EnFunc is also reasonable.

Definition 1. (MVC). An n-party MVC scheme $\Pi = (\{\mathrm{KeyGen}_j\}_{j=1}^n, \mathrm{EnFunc}, \{\mathrm{EnInput}_j\}_{j=1}^n, \mathrm{Compute}, \mathrm{Verify})$ for a function family $\mathcal{F}$ consists of $2n + 3$ algorithms as follows and in Figure 1:

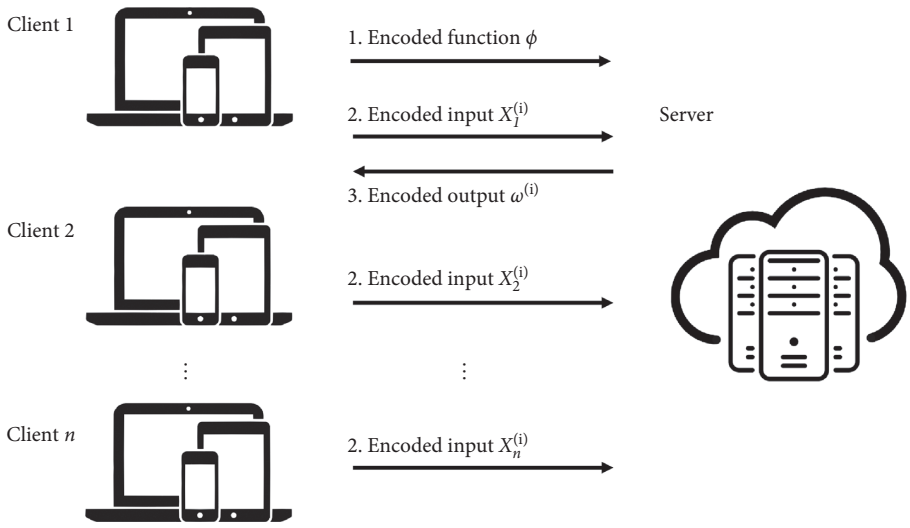

Figure 1: Multiclient verifiable computation model.

(i) $(pk_1, sk_1) \leftarrow \text{KeyGen}_1(pp)$. Client P_1 will execute this algorithm on public parameters pp to produce a public key pk_1 and a private key sk_1.

(ii) $(pk_j, sk_j) \leftarrow \text{KeyGen}_j(pp, pk_1)$. For $j = 2, \ldots, n$, client P_j will execute this algorithm to produce a public key pk_j and a private key sk_j.

(iii) $(\phi, \xi) \leftarrow \text{EnFunc}(\overrightarrow{pk}, sk_1, f)$. Client P_1 will execute this algorithm with $\overrightarrow{pk} = (pk_1, pk_2, \ldots, pk_n)$ and sk_1 to encode any $f \in \mathcal{F}$ to an encoded function ϕ and send ϕ to the server. Then, Client P_1 will produce a decoding secret ξ and keep it private.

(iv) $(\chi_1^{(i)}, \tau^{(i)}) \leftarrow \text{EnInput}_1(i, \overrightarrow{pk}, sk_1, \xi, x_1^{(i)})$. When out sourcing the i-th computation to the server, P_1 will execute this algorithm to encode its input $x_1^{(i)}$ to an encoded input $\chi_1^{(i)}$ and send $\chi_1^{(i)}$ to the server. Then, Client P_1 will produce a decoding secret $\tau^{(i)}$ and keep it private.

(v) $\chi_j^{(i)} \leftarrow \text{EnInput}_j(i, \overrightarrow{pk}, sk_j, x_j^{(i)})$. When outsourcing the i-th computation to the server, each client P_j $(j \neq 1)$ will execute the algorithm to encode its input $x_j^{(i)}$ to an encoded input $\chi_j^{(i)}$ and send $\chi_j^{(i)}$ to the server. We denote $\overrightarrow{\chi}^{(i)} = (\chi_1^{(i)}, \ldots, \chi_n^{(i)})$.

(vi) $\omega^{(i)} \leftarrow \text{Compute}(i, \overrightarrow{pk}, \phi, \overrightarrow{\chi}^{(i)})$. The server will execute the algorithm to obtain an encoded output $\omega^{(i)}$.

(vii) $y^{(i)} \cup \{\perp\} \leftarrow \text{Verify}(i, \xi, \tau^{(i)}, \omega^{(i)})$. Client P_1 will implement this algorithm to return either an evaluation result $y^{(i)} = f(x_1^{(i)}, \ldots x_n^{(i)})$ or a symbol $\perp$ informing that the server returned an incorrect result.

Required by [2,18], an MVC scheme should be correct, sound, and input private. An MVC satisfies the property of correctness if all the involving algorithms are honestly executed; an honest server will always produce output corresponding to the evaluation of f on those inputs and will always pass the verification.

An MVC scheme satisfies the property of soundness if no malicious server can fool clients into obtaining a wrong evaluation on given inputs, even if the server is given access to an oracle, which can generate arbitrary valid input encodings:

(i) **Oracle** $\mathcal{IN}(x_1, \ldots, x_n)$

(ii) $i := i + 1$;

(iii) record $(x_1^{(i)}, \ldots, x_n^{(i)}) := (x_1, \ldots, x_n)$;

(iv) $(\chi_1^{(i)}, \tau^{(i)}) \leftarrow \text{EnInput}_1(i, \overrightarrow{pk}, sk_1, \xi, x_1^{(i)})$;

(v) for $j = 2, \ldots, n$: $\chi_j^{(i)} \leftarrow \text{EnInput}_j(i, \overrightarrow{pk}, sk_j, x_j^{(i)})$;

(vi) output $(\chi_1^{(i)}, \ldots, \chi_n^{(i)})$.

Definition 2. (soundness). For scheme Π, consider an experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{sound}}[\Pi, f, \kappa, n]$ with a malicious server $\mathcal{A}$: for $j = 1, \ldots, n$, the public/private key pairs (pk_j, sk_j) are produced; an encoded function ϕ and a decoding secret ξ are produced. $\mathcal{A}$ is given inputs $\overrightarrow{pk}$ and ϕ and access to Oracle $\mathcal{IN}$ and returns a forge ω^*. The challenger obtains y^* by executing Verify; if $y^* \notin \{\perp, f(x_1^{(i)}, \ldots, x_n^{(i)})\}$, the output of $\mathbf{Exp}_{\mathcal{A}}^{\text{sound}}[\Pi, f, \kappa, n]$ is defined to be 1 and 0 otherwise. Scheme Π satisfies the property of soundness if for all $n = \text{poly}(\kappa)$, all functions $f \in \mathcal{F}$, and all probabilistic polynomial-time adversary (PPT) adversary $\mathcal{A}$; there is a negligible function $\text{negl}(\cdot)$ such that $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{sound}}[\Pi, f, \kappa, n] = 1] \leq \text{negl}(\kappa)$.

An MVC scheme satisfies the property of input privacy if no information about the inputs is leaked to all the other entities including both server and other clients. While the clients except the first one apparently had no opportunity to learn any information about the others' input data, the input privacy of the MVC scheme includes two properties: privacy against the first client and privacy against the server.

Definition 3. (privacy against the first client). The scheme Π achieves the privacy against the first client if for any $\overrightarrow{x}_0 = (x_1, x_2, \ldots, x_n)$, $\overrightarrow{x}_1 = (x_1, x_2', \ldots, x_n')$ with

$f(\vec{x}_0) = f(\vec{x}_1)$, the view of P_1 executing Π if all clients holding $\vec{x}_0$ cannot be distinguished from the view of P_1 when executing with all clients holding $\vec{x}_1$.

Privacy against the server requires that the server should not be able to distinguish the encoded inputs from two distinct inputs, even if the malicious server gains access to the Oracle $\mathcal{IN}$.

Definition 4. (privacy against the server). Consider an experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, b)$ with an adversarial server $\mathcal{A}$: for $j = 1, \ldots, n$, the public/private key pairs (pk_j, sk_j) are generated, so are an encoded function ϕ and a decoding secret ξ. The adversary $\mathcal{A}$ is given inputs pk and ϕ and access to Oracle $\mathcal{IN}$ and outputs two series of jointly inputs $(x_1^0, \ldots, x_n^0)$, $(x_1^1, \ldots, x_n^1)$. The challenge ciphertext $(\chi_1^b, \ldots, \chi_n^b)$ is computed and given to $\mathcal{A}$. $\mathcal{A}$ continues to have oracle access to $\mathcal{IN}$ and outputs a guess b' of b. The advantage of $\mathcal{A}$ in the experiment above is defined as

$$\text{Adv}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n) = \left|\Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, 0) = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, 1) = 1\right]\right|. \tag{1}$$

The MVC scheme Π is private against the server if for any $n = \text{poly}(\kappa)$, any function $f \in \mathcal{F}$, and any PPT adversary $\mathcal{A}$, there is a negligible function $\text{negl}(\cdot)$ such that $\text{Adv}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n) \leq \text{negl}(\kappa)$.

2.3. Homomorphic Proxy Re-Authenticator (HPRA). The HPRA scheme introduced by Derler et al. [3] consists of nine algorithms: Gen, *S*Gen, *V*Gen, Sign, Verify, SRGen, VRGen, Agg, and *A*Verify. And there are three types of parties: a set of signers, an aggregator, and a receiver. In a nutshell, HPRA allows n signers to authenticate data items $\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n$ with signatures under their own distinct keys and allows the aggregator to convert their signatures to one under the receiver's key. The aggregator outputs an aggregate authenticated message vector Λ consists of an evaluation result of function f on the inputs, and a signature corresponds to the result.

Definition 5. (HPRA). A homomorphic proxy re-authenticator scheme $\Sigma = (\text{Gen}, \text{SGen}, \text{VGen}, \text{Sign}, \text{Verify}, \text{SRGen}, \text{VRGen}, \text{Agg}, \text{AVerify})$ is composed of nine polynomial-time algorithms as follows and demonstrated in Figure 2.

(i) $pp \leftarrow \text{Gen}(1^\kappa, \ell)$: given a security parameter 1^κ and a constant ℓ, the algorithm generates public parameter set pp, which defines a message space $\mathcal{M}^\ell$, a function family $\mathcal{F} = \{f | f: (\mathcal{M}^\ell)^n \longrightarrow \mathcal{M}^\ell\}$, and a tag space.

(ii) $(\text{id}, \text{sk}, \text{pk}) \leftarrow \text{SGen}(pp)$: each signer P_i will execute this algorithm on public parameter set pp to output a signer key, including an identifier id, a private key sk, and a public key pk.

(iii) $(\text{mk}, \text{aux}) \leftarrow \text{VGen}(pp)$: the receiver will execute this algorithm on public parameters pp to obtain an MAC key mk and an auxiliary information aux.

(iv) $\sigma \leftarrow \text{Sign}(\text{sk}, \vec{m}, \mu)$: each signer will execute this algorithm to sign its input $\vec{m}$ as a signature σ, which will be sent to the aggregator. For all the signers $P_1, \ldots, P_n$, we denote their signatures as $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$.

(v) $b \leftarrow \text{Verify}(\text{pk}, \vec{m}, \mu, \sigma)$: any entity with $\vec{m}$ can verify the validation of a signature σ with the algorithm and outputs a bit $b \leftarrow \{0, 1\}$.

(vi) $\text{rk}_i \leftarrow \text{SRGen}(\text{sk}_i, \text{aux})$: each signer will generate a re-encryption key rk_i with the algorithm.

(vii) $\text{ak}_i \leftarrow \text{VRGen}(\text{pk}_i, \text{mk}, \text{rk}_i)$: with this algorithm, an aggregation key ak_i can be generated by the receiver, which will be sent to the aggregator.

(viii) $\Lambda \leftarrow \text{Agg}(\vec{\text{ak}}, \vec{\sigma}, \mu, f)$: the aggregator will generate the aggregate authenticated message vector Λ by the algorithm. Let $\vec{\text{ak}} = (\text{ak}_1, \ldots, \text{ak}_n)$.

(ix) $(\vec{m}, \mu)/(\perp, \perp) \leftarrow A\text{Verify}(\text{mk}, \Lambda, \text{ID}, f)$: the receiver will generate a pair $(\vec{m}, \mu)$ with the algorithm and otherwise output $(\perp, \perp)$ showing that the aggregator tries to cheat.

Required by [3], an HPRA scheme is required to be correct, input private, signer unforgeable, and aggregator unforgeable.

The correctness of an HPRA should meet the requirement that if all the involving algorithms are honestly executed, the aggregate authenticated message vector Λ will always pass the verification and extract the real result of evaluation $\vec{m} = f(\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n)$.

The input privacy of HPRA should meet the requirement that an aggregate authenticated message vector Λ should not leak any more information of the signers' data $\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n$ compared with what can be directly speculated from f and the real result of evaluation $\vec{m}$.

Definition 6. (input privacy). The HPRA scheme Σ for $\mathcal{F}$ is input private if for any $\kappa \in \mathbb{N}$, any $f \in \mathcal{F}$, all tags μ, and all $\mathbf{m}_1 = (\vec{m}_{11}, \ldots, \vec{m}_{n1}) \in (\mathcal{M}^\ell)^n$ and $\mathbf{m}_2 = (\vec{m}_{12}, \ldots, \vec{m}_{n2}) \in (\mathcal{M}^\ell)^n$ with $f(\vec{m}_{11}, \ldots, \vec{m}_{n1}) = f(\vec{m}_{12}, \ldots, \vec{m}_{n2})$, all $pp \leftarrow \text{Gen}(1^\kappa, \ell)$, all $(\text{mk}, \text{aux}) \leftarrow \text{VGen}(pp)$, for $i = 1, \ldots, n$, $(sk_i, pk_i) \leftarrow \text{SGen}(pp)$, $\text{rk}_i \leftarrow \text{SRGen}(sk_i, \text{aux})$, and $\text{ak}_i \leftarrow \text{VRGen}(\text{pk}_i, \text{mk}, \text{rk}_i)$. We denote QUOTE $\vec{\text{sk}} = (\text{sk}_1, \ldots \text{sk}_n)$ $\vec{\text{sk}} = (\text{sk}_1, \ldots \text{sk}_n)$. The following distributions are identical:

$$\left\{\text{Agg}\left(\vec{\text{ak}}, \text{Sign}\left(\vec{\text{sk}}, \mathbf{m}_1, \mu\right), \mu, f\right)\right\}, \quad \left\{\text{Agg}\left(\vec{\text{ak}}, \text{Sign}\left(\vec{\text{sk}}, \mathbf{m}_2, \mu\right), \mu, f\right)\right\}. \tag{2}$$

The signer unforgeability of an HPRA requires that if the aggregator always remains honest, no coalition of dishonest signers can produce a valid Λ with respect to the function $f \in \mathcal{F}$ such that Λ is outside of the range of f evaluated on arbitrary combinations of the actually signed vectors. The aggregator unforgeability is the natural counterpart of signer

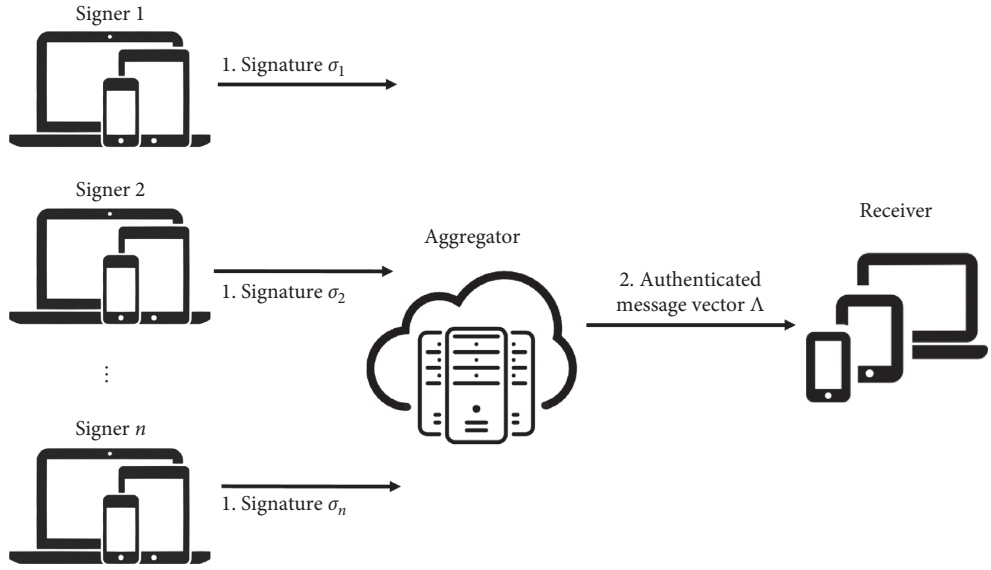

Figure 2: Homomorphic proxy re-authenticator model.

unforgeability; if the signers always remain honest, malicious aggregator cannot output a valid aggregate authenticated message vector with respect to the function f, such that the aggregate authenticated message vector is out of the range of f evaluated on the virtually signed vectors.

Let T represent "Signer" or "Aggregator." In both definitions, the adversary gains access to a set $\mathcal{O}_T$ of oracles, where $\mathcal{O}_T \coloneqq \{\mathrm{SG}, \mathrm{SKey}, \mathrm{SR}, \mathrm{VR}, A\}$ for $T =$ "Signer" and $\mathcal{O}_T \coloneqq \{\mathrm{SG}, \mathrm{Sig}, \mathrm{SR}, \mathrm{VR}, \mathrm{VRKey}\}$ for $T =$ "Aggregator." The oracles maintain some sets S, AK, RK, and SIG which are initially empty and work as follows, let $i = 1, \ldots, n$ represents the index of client P_i, and we do not consider the corruption between the signers:

(i) $\mathrm{SG}(i)$: works as SGen, sets $S[i] \leftarrow (\mathrm{id}, \mathrm{sk}, \mathrm{pk})$ and returns $(\mathrm{id}, \mathrm{pk})$.

(ii) $\mathrm{Skey}(i)$: returns $S[i]$.

(iii) $\mathrm{Sig}(\{1, \ldots, n\}, \mathbf{m})$: works as Sign, sets $\mathrm{SIG}[\mu] \leftarrow \mathrm{SIG}[\mu] \cup \{\vec{m}_i, S[i]\}$ for $i = 1, \ldots, n$, returns $\vec{\sigma} = (\sigma_1 \ldots \sigma_n)$ and μ. Let $\mathbf{m} = (\vec{m}_1, \ldots, \vec{m}_n)$.

(iv) $\mathrm{SR}(i)$: works as SRGen, returns $\mathrm{RK}[i] = \mathrm{rk}_i$.

(v) $\mathrm{VR}(i)$: works as VRGen but without returning anything, sets $\mathrm{AK}[i] = \mathrm{ak}_i$.

(vi) $\mathrm{VRKey}(i)$: returns $\mathrm{AK}[i]$.

(vii) $A(\vec{\sigma}, \{1, \ldots, n\}, \mu, f)$: works as Agg, returns Λ.

Definition 7. (**T**-unforgeability). For the HPRA scheme Σ, consider an experiment $\mathbf{Exp}_{\mathscr{A}}^{T-\mathrm{unforge}}(\Sigma, \kappa, n, l)$ with regard to a PPT adversary $\mathscr{A}$: Public parameter pp is generated by running $\mathrm{Gen}(1^\kappa, \ell)$; the MAC key and the auxiliary information $(\mathrm{mk}, \mathrm{aux})$ are generated by running $V\mathrm{Gen}(pp)$. $\mathscr{A}$ is given inputs pp and aux and access to oracle $\mathcal{O}_T$ and outputs a forge $(\Lambda^\star, \mathrm{ID}^\star, f^\star)$. The challenger carries out AVerify $(\mathrm{mk}, \Lambda^\star, \mathrm{ID}^\star, f^\star)$ and obtains $(\vec{m}, \mu)$; if $(\vec{m}, \mu) \neq (\perp, \perp)$ and $(\nexists(\mathbf{m}: (\forall i \in [n]: (\vec{m}_i, \mathrm{id}^*) \in \mathrm{SIG}\ [\mu]) \wedge f^\star\ (\vec{m}_1 \ldots, \vec{m}_n) = \vec{m})$, it outputs 1 and otherwise outputs 0. The HPRA scheme Σ is T-unforgeable; if for all PPT adversaries $\mathscr{A}$, there is a negligible function $\mathrm{negl}(\cdot)$ such that $\Pr[\mathbf{Exp}_{\mathscr{A}}^{T-\mathrm{unforge}}(\Sigma, \kappa, n, \ell) = 1] \leq \mathrm{negl}(\kappa)$.

There is an optional property for HPRA and output privacy, which simulates the situation that the aggregator learns nothing about either the inputs or the function's output. In order to formally give a definition of output privacy, we define an oracle RoS as follows:

(i) $\mathrm{RoS}(i, (\mathbf{m}, b))$: If $\mathrm{S}[i] = \perp$, it returns $\perp$. Otherwise, it samples μ uniformly at random, and if $b = 0$, for $i = 1, \ldots, n$, it computes $\sigma_i \leftarrow \mathrm{Sign}(S[i][2], \vec{m}_i, \mu)$. Else, it randomly chooses $\vec{r} = (r_1, \ldots, r_n) \leftarrow (\mathscr{M}^\ell)^n$, and for $i = 1, \ldots, n$, it computes $\sigma_i \leftarrow \mathrm{Sign}(S[i][2], \vec{r}_i, \mu)$ and returns $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$.

Definition 8. (output privacy). For the HPRA scheme Σ, assuming an experiment $\mathbf{Exp}_{\mathscr{A}}^{\mathrm{outpriv}}(\Sigma, \kappa, n, \ell)$ with a PPT adversaries $\mathscr{A}$: public parameter pp is gained by executing $\mathrm{Gen}(1^\kappa, \ell)$, the MAC key, and the auxiliary information $(\mathrm{mk}, \mathrm{aux})$ are gained by executing $V\mathrm{Gen}(pp)$. $\mathscr{A}$ is given input pp and access to oracle $\mathcal{O}$. A random bit $b \leftarrow \{0, 1\}$ is chosen by the challenger, and a challenge ciphertext $\vec{\sigma}$ is computed and given to $\mathscr{A}$. $\mathscr{A}$ continues to have oracle access to QUOTE $\mathcal{O} \coloneqq \{\mathrm{SG}, \mathrm{SKey}, \mathrm{RoS}(b), \mathrm{SR}, \mathrm{VR}, \mathrm{VRKey}\}$ $\mathcal{O} \coloneqq \{\mathrm{SG}, \mathrm{SKey}, \mathrm{RoS}(b), \mathrm{SR}, \mathrm{VR}, \mathrm{VRKey}\}$ and outputs a guess b' of b.

An HPRA for a family of function classes $\mathscr{F}$ is output private; if for all PPT adversaries $\mathscr{A}$, there is a negligible function $\mathrm{negl}(\cdot)$ such that $\Pr[\mathrm{Exp}_{\mathscr{A}}^{\mathrm{outpriv}}(\Sigma, \kappa, n, \ell) = 1] \leq (1/2) + \mathrm{negl}(\kappa)$.

3. The HPRA to MVC Transform

Following the definition in Section 2, some similarities between the two models are figured out through observation: (1) the clients in MVC and signers in HPRA play similar roles in providing inputs; (2) the server in MVC and the aggregator

in HPRA play similar roles in computing an encoded output with a corresponding proof; (3) the first client in MVC and the receiver in HPRA play similar roles in extracting the evaluation result and verifying its correctness. A very straightforward idea is that we can achieve the goal of transforming HPRA to MVC by constructing a mapping of the participants in the two schemes as follows, which can be depicted in Figure 3: (1) let the aggregator in HPRA take over the work of the server in MVC; (2) merge the receiver with the first signer in HPRA, and let this merged participant take over the work of the first client in MVC; (3) let each of the rest signers play a different client in MVC.

Let $\Sigma = (\text{Gen}, \text{SGen}, \text{VGen}, \text{Sign}, \text{Verify}, \text{SRGen}, \text{VRGen}, \text{Agg}, \mathcal{A}\text{Verify})$◁ be an HPRA scheme for a function family $\mathcal{F}$. The general transform from Σ to $\Pi = (\{\text{KeyGen}_j\}_{j=1}^{n}, \text{EnFunc}, \{\text{EnInput}_j\}_{j=1}^{n}, \{\text{EnInput}_j\}_{j=1}^{n}, \text{Compute}, \text{Verify})$, an MVC scheme for $\mathcal{F}$, will be explained in elaborate as follows. Let $pp \leftarrow \Sigma.\text{Gen}(1^{\kappa}, \ell)$ be a set of public parameters. In the proposed MVC scheme, we consider the computation of a function f on inputting $(\vec{x}_1^{(i)}, \ldots, \vec{x}_n^{(i)})$, where $\vec{x}_j^{(i)} \in \mathcal{M}^{\ell}$ is a vector over a finite field of each client P_j.

(i) $(pk_1, sk_1) \leftarrow \text{KeyGen}_1(pp)$. On inputting public parameter pp, client P_1 executes HPRA's signer's key generation algorithm $\Sigma.\text{SGen}(pp)$ and obtains an identifier id_1 and a public/private key pair $(\text{sk}_1, \text{pk}_1)$. Client P_1 runs receiver's key generation algorithm. $\Sigma.\text{VGen}(pp)$ gets an MAC key mk and auxiliary information aux. On private key sk_1 and auxiliary information aux, client P_1 executes signer's re-encryption key generation algorithm $\Sigma.\text{SRGen}(sk_1, \text{aux})$, and obtains re-encryption key rk_1. Client P_1 sets $pk_1 = (\text{id}_1, \text{pk}_1, \text{aux}, \text{rk}_1)$, $sk_1 = (\text{sk}_1, \text{mk})$.

(ii) $(pk_j, sk_j) \leftarrow \text{KeyGen}_j(pp, pk_1)$. On inputting public parameter pp and client P_1's public key pk_1, for $j = 2, \ldots, n$, each client P_j executes HPRA's signer's key generation algorithm $\Sigma.\text{SGen}(pp)$ and obtains an identifier id_j and a public/private key pair $(\text{sk}_j, \text{pk}_j)$. On inputting private key sk_j and auxiliary information aux, client P_j executes signer's re-key generation algorithm $\Sigma.\text{SRGen}(sk_j, \text{aux})$ and obtains a re-encryption key rk_j. Client P_j individually sets $pk_j = (\text{id}_j, \text{pk}_j, \text{rk}_j)$, $sk_j = \text{sk}_j$.

(iii) $(\phi, \xi) \leftarrow \text{EnFunc}(\vec{pk}, sk_1, f)$. For $j = 1 \ldots n$, takes a public key pk_j and the private key sk_1 of client P_1, client P_1 executes $\Sigma.\text{VRGen}(pk_j, \text{mk}, \text{rk}_j)$ and obtains an aggregation key ak_j. Client P_1 sets the encoded function $\phi = (f, \vec{\text{ak}})$ and the decoding secret $\xi = \varnothing$.

(iv) $(\chi_1^{(i)}, \tau^{(i)}) \leftarrow \text{EnInput}_1(i, \vec{pk}, sk_1, \xi, \vec{x}_1^{(i)})$. When out sourcing the i-th computation to the server, it takes a time period i, the public keys $\vec{pk}$, the private key sk_1 of client P_1, an input message vector $\vec{x}_1^{(i)}$, the decoding secret ξ, the first client P_1 gets a tag $\mu \in \mathbb{G}$, executes the sign algorithm $\Sigma.\text{Sign}(\text{sk}_1, \vec{x}_1^{(i)}, \mu)$, gets a signature σ_1, and sets $\chi_1^{(i)} = \sigma_1$, $\tau^{(i)} = \text{mk}$.

(v) $\chi_j^{(i)} \leftarrow \text{EnInput}_j(i, \vec{pk}, sk_j, \vec{x}_j^{(i)})$. When outsourcing the i-th computation to the server, each client P_j takes a time period i, the public keys $\vec{pk}$, the signer secret key sk_j of client P_j, and input message vector $\vec{x}_j^{(i)}$; P_j (with $j \neq 1$) obtains a tag $\mu \in \mathbb{G}$, and then he executes the algorithm $\Sigma.\text{Sign}(\text{sk}_j, \vec{x}_j^{(i)}, \mu)$, gets signature σ_j, and sets $\chi_j^{(i)} = \sigma_j$.

(vi) $\omega^{(i)} \leftarrow \text{Compute}(i, \vec{pk}, \phi, \vec{\chi}^{(i)})$. Given the public keys $\vec{pk}$, the encoded function ϕ, and the encoded inputs $\vec{\chi}^{(i)}$, the server executes the algorithm $\Sigma.\text{Agg}(\text{ak}, \vec{\chi}^{(i)}, \mu, \phi)$, gets aggregate authenticated message vector Λ, and sets $\omega^{(i)} = \Lambda$.

(vii) $y^{(i)} \cup \{\perp\} \leftarrow \text{Verify}(i, \vec{pk}, \xi, \tau^{(i)}, \omega^{(i)})$. Take the public keys $\vec{pk}$, the decoding secrets $(\xi, \tau^{(i)})$, and an encoded output $\omega^{(i)}$ as inputs, and the first client P_1 executes the receiver's verification algorithm $\Sigma.\mathcal{A}\text{Verify}(\text{mk}, \Lambda, \text{ID}, f)$ which outputs pair of message vector and tag $(\vec{m}, \mu)$, on success, and $(\perp, \perp)$ otherwise. Set $y^{(i)} = \vec{m}$ and otherwise return $\perp$ informing that the server tries to cheat.

Scheme Π should satisfy the properties of correctness, soundness, and privacy. While the correctness is quite obvious, we start with soundness.

Theorem 1. *(soundness). If Σ is a T-unforgeable HPRA scheme, where $T =$ "Aggregator," then Π described above is a sound MVC scheme.*

Proof. This study demonstrates that if there is a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ for which break soundness (Definition 2) of Π and let $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{sound}}[\Pi, f, \kappa, n] = 1] \geq \text{non} - \text{negl}(\kappa)$. A PPT adversary $\mathcal{B}$ can be constructed that breaks the T-unforgeability (Definition 7) of Σ for $T =$ "Aggregator." $\mathcal{B}$ is given inputs $(1^{\kappa}, n, \ell)$ and an oracle $\mathcal{O}_T := \{\text{SG}, \text{Sig}, \text{SR}, \text{VR}, \text{VRKey}\}$, and its goal is to output a forge that can successfully convince the challenger. In detail, the following holds:

Experiment $\mathbf{Exp}_{\mathcal{B}}^{T-\text{unforge}}[\Sigma, \kappa, n, \ell]$:

(1) The challenger executes $pp \leftarrow \text{Gen}(1^{\kappa}, \ell)$, $(\text{mk}, \text{aux}) \leftarrow \text{VGen}(pp)$ and sends pp and aux to $\mathcal{B}$.

(2) For $j = 1, \ldots n$, $\mathcal{B}$ queries the following oracles and obtains $(\text{id}_j, \text{pk}_j) \leftarrow \text{SG}(j)$, $\text{rk}_j \leftarrow \text{SR}(j)$; $\mathcal{B}$ also queries the oracle $\text{VR}(j)$ to produce ak_j and queries $\text{ak}_j \leftarrow \text{VRKey}(j)$.

(3) $\mathcal{B}$ then sets $\text{pk}_1 = (\text{id}_1, pk_1, \text{aux}, \text{rk}_1)$; for $j = 2 \ldots n$, $pk_j = (\text{id}_j, \text{pk}_j, \text{rk}_j)$, $\phi = (f, \text{ak})$, and sends them to $\mathcal{A}$. Let f be the function on which $\mathcal{A}$ can break MVC soundness.

(4) $\mathcal{B}$ initializes a counter $i := 0$.

(5) Whenever $\mathcal{A}$ queries its encryption Oracle $\mathcal{IN}$ with inputs $(\vec{x}_1, \ldots, \vec{x}_n)$, $\mathcal{B}$ answers the queries as follows:

(a) Set $i := i + 1$.
(b) Record $(\vec{x}_1^{(i)}, \ldots, \vec{x}_n^{(i)}) := (\vec{x}_1, \ldots, \vec{x}_n)$.

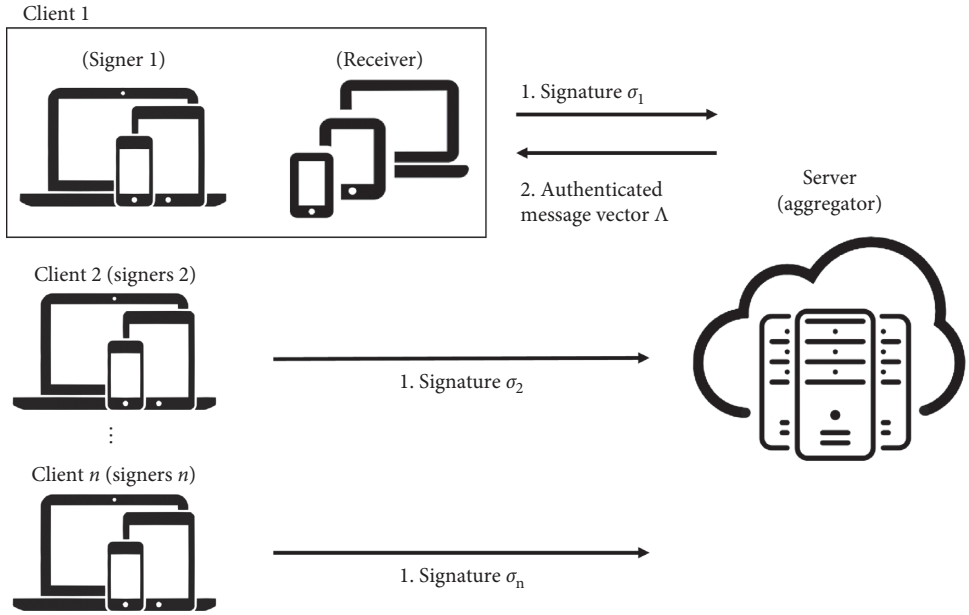

Figure 3: Idea of transform.

(c) Query and get $(\sigma_j^{(i)}, \mu) \leftarrow \mathrm{Sig}(\{1, \ldots, n\}, x^{(i)})$. Let $\mathbf{x}^{(i)} = (\overrightarrow{x}_1^{(i)}, \ldots, \overrightarrow{x}_n^{(i)})$, and denote $\overrightarrow{\sigma}^{(i)} = (\sigma_1^{(i)}, \ldots, \sigma_n^{(i)})$.
(d) Set and return $\overrightarrow{\chi}^{(i)} := \overrightarrow{\sigma}^{(i)}$.

(6) When $\mathscr{A}$ outputs $\omega^\star$, $\mathscr{B}$ sets $\Lambda^\star := \omega^\star$ and sends $(\Lambda^\star, \mathrm{ID}^\star, f^\star)$ to the challenger, where $\mathrm{ID}^\star := (\mathrm{id}_1, \ldots, \mathrm{id}_n)$, $f^\star = f$.
(7) The challenger executes the algorithm AVerify $(\mathrm{mk}, \Lambda^\star, \mathrm{ID}^\star, f^\star)$.
(8) If $(\overrightarrow{m}, \mu) \neq (\perp, \perp)$ and $(\nexists(\mathbf{x}: (\forall j \in [n]: (\overrightarrow{x}_j, \mathrm{id}^\star) \in\in \mathrm{SIG}[\mu]) \wedge f^\star(\overrightarrow{x}_1 \ldots, \overrightarrow{x}_n) = \overrightarrow{m})$ (i.e., 1. $f^\star(\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n) = \overrightarrow{m}$, and at least one $\overrightarrow{x}_j$ has not been queried by $\mathscr{A}$ in the i-th query or 2. $f^\star(\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n) \neq \overrightarrow{m}$) outputs 1 and otherwise outputs 0.

A PPT adversary $\mathscr{A}$ will find a $\omega^\star$ such that the challenger gets $y^* \notin \{\perp, f(\overrightarrow{x}_1^{(i)}, \ldots, \overrightarrow{x}_n^{(i)})\}$ when running Verify if $\mathscr{A}$ is considered to be able to break Π with non-negl(κ) As Π.Verify in our transform directly calls $\Sigma.A$Verify, if such a ω^* is produced, $\mathscr{B}$ can directly consider ω^* as an input of $\Sigma.A$Verify. Clearly, if $\mathscr{A}$ can cheat Π.Verify and make $\mathbf{Exp}_{\mathscr{A}}^{\mathrm{sound}}[\Pi, f, \kappa, n] = 1$, $\mathscr{B}$ can definitely also fool $\Sigma.A$Verify and let $\mathbf{Exp}_{\mathscr{B}}^{T-\mathrm{unforge}}(\Sigma, \kappa, n, \ell)$ equal to 1. Then, we successfully construct such an adversary $\mathscr{B}$ who breaks T-unforgeability when $T =$ "Aggregator" with non-negl(κ): $\Pr[\mathrm{Exp}_{\mathscr{B}}^{T-\mathrm{unforge}}(\Sigma, \kappa, n, \ell) = 1] \geq \Pr[\mathrm{Exp}_{\mathscr{A}}^{\mathrm{sound}}(\Pi, f^*, \kappa, n) = 1] \geq \mathrm{non} - \mathrm{negl}(\kappa)$.

However, it has been proved that Σ is a T-unforgeable HPRA [3], which contradicts our derivation; that is, soundness is guaranteed in MVC based on the T-unforgeability of HPRA. □

Theorem 2. *(privacy against the first client). If Σ is an input private HPRA scheme, then Π is an MVC scheme with the privacy against the first client.*

Proof. The privacy against the first client (Definition 3) can be directly derived from the input privacy of HPRA. Recall Definition 6, an HPRA Π scheme is called input private if for all $(\overrightarrow{m}_{11}, \ldots, \overrightarrow{m}_{n1})$ and $(\overrightarrow{m}_{12}, \ldots, \overrightarrow{m}_{n2})$, where $f(\overrightarrow{m}_{11}, \ldots, \overrightarrow{m}_{n1}) = f(\overrightarrow{m}_{12}, \ldots, \overrightarrow{m}_{n2})$, and the following distributions are identical:

$$\begin{aligned} \Lambda_1 &\leftarrow \Pi.\mathrm{Agg}\left(\overrightarrow{\mathrm{ak}}, \overrightarrow{\sigma}_1, \mu, f\right), \\ \Lambda_2 &\leftarrow \Pi.\mathrm{Agg}\left(\overrightarrow{\mathrm{ak}}, \overrightarrow{\sigma}_2, \mu, f\right), \end{aligned} \tag{3}$$

where $\overrightarrow{\sigma}_1$ is the signatures of $(\overrightarrow{m}_{11}, \ldots, \overrightarrow{m}_{n1})$ from Σ.Sign and $\overrightarrow{\sigma}_2$ is the signatures of $(\overrightarrow{m}_{12}, \ldots, \overrightarrow{m}_{n2})$ from Σ.Sign.

Recall Definition 3; an MVC scheme Σ is called privacy against the first client if the view of the first client when executing Σ with clients holding inputs $(\overrightarrow{x}_1, \overrightarrow{x}_2, \ldots, \overrightarrow{x}_n)$ is distributed identically from the view of the first client when running Σ with clients holding inputs $(\overrightarrow{x}_1, \overrightarrow{x}_2', \ldots, \overrightarrow{x}_n')$.

Since the first client P_1 has no other opportunity to access information from all other entities when performing Σ, except when running the algorithm Verify: P_1 obtains an encoded output $\omega^{(i)}$, which may reveal some information. As a result, it is only necessary to prove that the distributions below are identical:

$$\begin{aligned} \omega^{(i)} &\longleftarrow \Sigma.\mathrm{Compute}\left(i, \overrightarrow{pk}_j, \phi, \overrightarrow{\chi}^{(i)}\right), \\ \omega'^{(i)} &\longleftarrow \Sigma.\mathrm{Compute}\left(i, \overrightarrow{pk}_j, \phi, \overrightarrow{\chi}'^{(i)}\right). \end{aligned} \tag{4}$$

Let $\vec{\chi}^{(i)}$ be the encoded inputs of $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$, generated by $\Pi.\text{EnInput}_1$ and $\Pi.\text{EnInput}_j$. And let $\vec{\chi}'^{(i)}$ be the encoded inputs of $(\vec{x}_1, \vec{x}_2', \ldots, \vec{x}_n')$, generated by $\Pi.\text{EnInput}_1$ and $\Pi.\text{EnInput}_j$. Without loss of generality, we set $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$ and $(\vec{x}_1, \vec{x}_2', \ldots, \vec{x}_n')$, be any two vectors with the same first component such that $f(\vec{x}_1, \vec{x}_2, (\vec{x}_1, \vec{x}_2', \ldots, \vec{x}_n')$ at the same time. Intuitively speaking, both $\Pi.\text{EnInput}_1$ and $\Pi.\text{EnInput}_j$ directly call the algorithm $\Sigma.\text{Sign}$, except that $\Pi.\text{EnInput}_1$ returns mk additionally; in other words, $\vec{\chi}^{(i)} = \vec{\sigma}_1$ and $\vec{\chi}'^{(i)} = \vec{\sigma}_2$. Furthermore, $\Pi.\text{Compute}$ calls $\Sigma.\text{Agg}$, and $\omega^{(i)}$ and $\omega'^{(i)}$ are calculated exactly in the same way as in the definition of input privacy of HPRA, i.e., $\omega(i) = \Lambda_1$ and $\omega'^{(i)} = \Lambda_2$. Because of the input privacy of HPRA, $\omega^{(i)}$ and $\omega'^{(i)}$ are identical. As a result, privacy against the first client of MVC scheme Σ is guaranteed. □

Theorem 3. *(privacy against the server). If Σ is an output private HPRA scheme, then Π will be an MVC scheme with the privacy against the server property.*

Proof. If there is a PPT adversary $\mathcal{A}$ for which privacy against the server of Π (Definition 4) on function f does not hold, then a PPT adversary $\mathcal{B}$ that breaks the output privacy of Σ (Definition 8) can be constructed. $\mathcal{B}$ is given inputs $1^\kappa, n, \ell$ and access to an oracle $\mathcal{O} := \{\text{SG}, \text{SKey}, \text{RoS}\ (b), \text{SR}, \text{VR}, \text{VRKey}\}$. We follow the assumption of MVC that all the clients are honest but curious, and we do not consider corrupted client situation. In detail, the following holds:

(i) Experiment $\mathbf{Exp}_{\mathcal{B}}^{\text{outpriv}}[\Sigma, \kappa, n, \ell]$:

(1) The challenger carries out $pp \leftarrow \text{Gen}(1^\kappa, \ell)$, $(\text{mk}, \text{aux}) \leftarrow \text{VGen}(pp)$ and selects a random bit $b \leftarrow \{0, 1\}$ and then returns pp and aux to $\mathcal{B}$.

(2) For $j = 1, \ldots, n$, $\mathcal{B}$ queries the oracles and obtains $(\text{id}_j, \text{pk}_j) \leftarrow \text{SG}(j)$, $\text{rk}_j \leftarrow \text{SR}(j)$. $\mathcal{B}$ also queries the oracle $\text{VR}(j)$ to yield ak_j, and queries $\text{ak}_j \leftarrow \text{VRKey}(j)$.

(3) $\mathcal{B}$ then sets $pk_1 = (\text{id}_1, \text{pk}_1, \text{aux}, \overrightarrow{\text{rk}}_1)$, for $j = 2, \ldots, n$, $pk_j = (\text{id}_j, \text{pk}_j, \text{rk}_j)$, $\phi = (f, \overrightarrow{\text{ak}})$, and sends them to $\mathcal{A}$. Let f be the function on which $\mathcal{A}$ can break the MVC soundness.

(4) $\mathcal{B}$ initializes a counter $i := 0$.

(5) When $\mathcal{A}$ queries the encryption oracle $\mathcal{IN}$ with inputs $(\vec{x}_1, \ldots, \vec{x}_n)$, $\mathcal{B}$ answers the queries as follows:

(a) Set $i := i + 1$.
(b) Record $(\vec{x}_1^{(i)}, \ldots, \vec{x}_n^{(i)}) := (\vec{x}_1, \ldots, \vec{x}_n)$.
(c) Query the oracle and obtain $(\vec{\sigma}^{(i)}, \mu) \leftarrow \text{Sig}(\{1, \ldots, n\}, (\vec{x}_j^{(i)})_{j \in [n]})$.
(d) Let $\vec{\chi}^{(i)} := \vec{\sigma}^{(i)}$ and return $\vec{\chi}^{(i)}$ to $\mathcal{A}$.

(ii) When $\mathcal{A}$ outputs $\mathbf{x}^0 = (\vec{x}_1^0, \ldots, \vec{x}_n^0)$, $\mathbf{x}^1 = (\vec{x}_1^1, \ldots, \vec{x}_n^1)$, $\mathcal{B}$ picks a random bit $b_0 \leftarrow \{0, 1\}$ and works as follows:

Queries $\text{RoS}(j, \mathbf{x}^{b_0}, b)$ and gets $((\sigma_1^{b_0}, c_1^{b_0}), \ldots, (\sigma_n^{b_0}, c_n^{b_0}))$.
Returns the challenge ciphertext $\vec{\chi}^{b_0} := \vec{\sigma}^{b_0}$ to $\mathcal{A}$.

(iii) While $\mathcal{A}$ gives a bit b_1 to $\mathcal{B}$, $\mathcal{B}$ outputs 1, if $b_0 = b_1$; otherwise, $\mathcal{B}$ outputs 0.

Intuitively speaking, when $b_0 = b_1$, $\mathcal{A}$ outputs a successful guess $b' = b$ of which one of the inputs $\mathbf{x}^0, \mathbf{x}^1$ is signed, then $\mathcal{B}$ guesses that its challenge string given by $\text{RoS}(b)$ must be a signature and outputs a guess $b' = 1$ of b. Otherwise, while if $\mathcal{A}$ does not succeed, $\mathcal{B}$ guesses that its challenge string given by $\text{RoS}(b)$ must be random and generates a guess bit $b' = 0$ of b. When the privacy against the server (Definition 3) of Π against $\mathcal{A}$ does not hold, we have

$$\begin{aligned}\text{Adv}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n) &= \left|\Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, 0) = 1\right]\right. \\ &\quad \left. - \Pr\left[\text{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, 1) = 1\right]\right| \\ &\geq \text{non} - \text{negl}(\kappa).\end{aligned} \tag{5}$$

Without loss of generality, let $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}[\Pi, f, \kappa, n, 1] = 1]$ be bigger than $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}[\Pi, f, \kappa, n, 0] = 1]$.

As $\mathbf{x}^{b_0} = (\vec{x}_1^{b_0}, \ldots, \vec{x}_n^{b_0})$ is encrypted by $\text{RoS}(b)$ when $b = 0$, then the view of $\mathcal{A}$ when runs as a subroutine by $\mathcal{B}$ is indistinguishable to the view of $\mathcal{A}$ in experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}[\Pi, f, \kappa, n, b_0]$. As a result,

$$\begin{aligned}\Pr\left[\mathbf{Exp}_{\mathcal{B}}^{\text{outpriv}}[\Sigma, \kappa, n, \ell] = 1 | b = 0\right] &= \Pr[b_0 = 0] \\ &\quad - \Pr[b_0 = 0] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, 0) = 1\right] \\ &\quad + \Pr[b_0 = 1] \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{priv}}(\Pi, f, \kappa, n, 1) = 1\right].\end{aligned} \tag{6}$$

Because b_0 is randomly picked by $\mathcal{B}$, $\Pr[b_0 = 0] = \Pr[b_0 = 1] = (1/2)$, we have

$$\Pr\left[\mathbf{Exp}_{\mathcal{B}}^{\text{outpriv}}[\Sigma, \kappa, n, \ell] = 1 \middle| b = 0\right] \geq \frac{1}{2} + \frac{1}{2} \cdot \text{non} - \text{negl}(\kappa). \tag{7}$$

A string of random numbers is encrypted by $\text{RoS}(b)$ when $b = 1$, then the view of $\mathcal{A}$ when runs as a subroutine by $\mathcal{B}$ is indistinguishable to the view of $\mathcal{A}$ in an experiment of guessing random numbers. Thus,

$$\Pr\left[\mathbf{Exp}_{\mathcal{B}}^{\text{outpriv}}(\Sigma, \kappa, n, \ell) = 1 | b = 1\right] = \frac{1}{2}. \tag{8}$$

Considering the above two equations, because the challenger chooses b randomly, $\Pr[b = 0] = \Pr[b = 1] = (1/2)$, if adversary $\mathcal{A}$ can break the privacy against the server and $\mathcal{B}$ can determine whether its challenge string is a true signature or a random string with probability:

$$\Pr\left[\mathbf{Exp}_{\mathscr{B}}^{\text{outpriv}}(\Sigma,\kappa,n,\ell)=1\right]=\frac{1}{2}\cdot\Pr\left[b'=1|b=0\right]+\frac{1}{2}\cdot\Pr\left[b'=1|b=1\right]\geq\frac{1}{2}+\frac{1}{4}\cdot\text{non}-\text{negl}(\kappa). \tag{9}$$

But the output privacy has been confirmed [3], which leads to a contradiction. In other words, privacy against the server is guaranteed in our MVC scheme based on the output privacy of HPRA. □

4. A Concrete Instantiation

To explain our transform in more detailed way, we give a concrete instantiation. Essentially, we build an MVC for the family of linear function F over $\mathbb{Z}_q$ based on a concrete HPRA scheme 4 in [3].

4.1. Bilinear Map. Let $\mathbb{G}_1=\langle g_1\rangle$, $\mathbb{G}_2=\langle g_2\rangle$, and $\mathbb{G}_T=\langle g\rangle$ be cyclic groups of prime order q. Let g_1, g_2 be generators of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and a paring e: $\mathbb{G}_1\times\mathbb{G}_2\longrightarrow\mathbb{G}_T$ is a bilinear map if the following features hold:

(i) Bilinearity: $e(u^a,v^b)=e(u,v)^{ab}$ for any $u\in\mathbb{G}_1, v\in\mathbb{G}_2$ and $a,b\in\mathbb{Z}_q$.

(ii) Nondegeneracy: $e(g_1,g_2)\neq 1_{\mathbb{G}_T}$.

(iii) Computability: there is an efficient algorithm to compute $e(u,v)$ for any inputs $u\in\mathbb{G}_1$, $v\in\mathbb{G}_2$.

For simplicity, assuming there is a symmetric bilinear map such that $\mathbb{G}_1=\mathbb{G}_2=\langle g\rangle$. Let BG$\leftarrow$BGGen$(1^\kappa)$ be a bilinear group generation algorithm, and the output BG is the generator $(g,\mathbf{g})$ of $\mathbb{G}$ and $\mathbb{G}_T$ with a bilinear pairing e: $\mathbb{G}\times\mathbb{G}\longrightarrow\mathbb{G}_T$.

4.2. Overview of a Detailed HPRA Scheme. Scheme 4 in [3] is said to be an input private, T-unforgeable, and output private HPRA scheme. They allow the signer to report an encrypted data with its signature under their own keys; the aggregator re-encrypts the ciphertexts and converts them to the ciphertexts under the receiver's key and homomorphically evaluates the function f on ciphertexts.

For more details, to achieve the output privacy and the homomorphism, they employ an ElGamal-like encryption scheme: encode message $x\in\mathbb{Z}_q$ into the exponent and encrypt g^x. One can decrypt and get $x'=g^x$ and then additionally compute $x=\log_g x'$ to obtain x, with size of the message space being polynomial in security parameter κ. As one can verify the guesses for signed messages using solely the signatures, they blind signer P_j's signature with a random element g^{r_j}. Unfortunately, because of some flaws in this scheme, even if the aggregator performs honestly, the evaluation result still cannot pass AVerify; that is, the Scheme 4 in [3] is not a correct HPRA scheme.

We slightly modified Scheme 4 in [3] and turned it into a correct HPRA scheme: we blinded the input od signer P_j with a random element g^{r_j} instead of blinding signer of P_j with a random element g^{r_j} and prevented verifying guesses for signed messages as well. It has been proven that the revised HPRA scheme still maintains input privacy, T-unforgeability, and output privacy (see Appendix A for more details of the origin scheme 4 in [3] and how we revised it; Appendix B gives the security proof).

4.3. The Construction. We describe n clients where each client P_j holds an ℓ-length input $\vec{x}_j=(\vec{x}_j[1],\ldots,\vec{x}_j[\ell])\in(\mathbb{Z}_q)^\ell$ and a tag $\mu\in\mathbb{G}$ which is related to the set of inputs. Here, f is a linear function represented by its coefficient $(q_1,\ldots,q_n)$. It is worth noting that when the inputs of f are vectors, e.g., given n ℓ-length input vectors $(\vec{x}_1,\ldots,\vec{x}_\ell)$, we define f as a function calculated column by column, that is,

$$f(\vec{x}_1,\ldots,\vec{x}_n):=\left(\sum_{j=1}^{n}\left[\vec{x}_j1\right]\cdot q_j,\ldots,\sum_{j=1}^{n}\left[\vec{x}_j\ell\right]\cdot q_j\right). \tag{10}$$

An additional symbol is placed at the end of the ciphertext to indicate the state of the ciphertext: "1" stands for first-level ciphertext, which is not allowed to re-encrypt, "2" stands for second-level ciphertext, which is allowed to be re-encrypted, and "R" means that this ciphertext has already been re-encrypted. In this instantiation, we describe the process for one time period i to simplify.

We fix a hash function H: $\mathbb{G}\times\mathbb{G}\longrightarrow\mathbb{G}$, which can later be used as a public random oracle. We also choose $\vec{g}=(g_1,\ldots,g_\ell)\leftarrow\mathbb{G}^\ell$ and publish the public parameter $pp\leftarrow(\text{BG},H,\vec{g},\ell)$:

(1) $(pk_1,sk_1)\leftarrow\text{KeyGen}_1(pp)$.

(a) On inputting public parameter pp, client P_1 randomly chooses a vector $\vec{a}_1=(\vec{a}_1[1],\ldots,\vec{a}_1[\ell+1])\xleftarrow{R}\mathbb{Z}_q^{\ell+1}$, and $\beta_1\xleftarrow{R}\mathbb{Z}_q$ yields an identifier and a public/private key pair set:

$$\begin{aligned}\text{id}_1&=g^{(1/\beta_1)},\\ \text{pk}_1&=\left(\left(\mathbf{g}^{\vec{a}_1[1]},\ldots,\mathbf{g}^{\vec{a}_1[\ell+1]}\right),g^{\beta_1},g^{(1/\beta_1)},pp\right),\\ \text{sk}_1&=(\vec{a}_1,\beta_1).\end{aligned} \tag{11}$$

(b) The first client P_1 randomly chooses $\alpha\xleftarrow{R}\mathbb{Z}_q$, $\vec{b}=(b_1,\ldots,b_{\ell+1})\xleftarrow{R}\mathbb{Z}_q^{\ell+1}$ and obtains an MAC key mk and the auxiliary information aux:

$$\begin{aligned}\text{mk}&=(\alpha,\vec{b}),\\ \text{aux}&=\left(g^{b_1},\ldots,g^{b_{\ell+1}}\right).\end{aligned} \tag{12}$$

(c) Client P_1 obtains the re-encryption key $\text{rk}_1=((g^{b_1})^{\vec{a}_1[1]},\ldots,(g^{b_{\ell+1}})^{\vec{a}_1[\ell+1]})$.

(d) Client P_1 sets and outputs:

$$pk_1 = (\mathrm{id}_1, \mathrm{pk}_1, \mathrm{aux}, \mathrm{rk}_1),$$
$$sk_1 = (\mathrm{sk}_1, \mathrm{mk}). \tag{13}$$

(2) $(pk_j, sk_j) \leftarrow \mathrm{KeyGen}_j(pp, pk_1)$.

(a) On inputting public parameter pp, client P_j randomly chooses $\overrightarrow{a}_j = (\overrightarrow{a}_j[1], \ldots, \overrightarrow{a}_j[\ell+1]) \xleftarrow{R} \mathbb{Z}_q^{l+1}$, and $\beta_j \xleftarrow{R} \mathbb{Z}_q$ produces an identifier and a public/private key pair set:

$$\mathrm{id}_j = g^{(1/\beta_j)},$$
$$\mathrm{pk}_j = \left(\left(\mathbf{g}^{\overrightarrow{a}_j[1]}, \ldots, \mathbf{g}^{\overrightarrow{a}_j[\ell+1]}\right), g^{\beta_j}, g^{(1/\beta_j)}, pp\right),$$
$$\mathrm{sk}_j = \left(\overrightarrow{a}_j, \beta_j\right). \tag{14}$$

(b) For $j = 2, \ldots, n$, client P_j obtains the re-encryption key $\mathrm{rk}_j = (g^{b_1 \overrightarrow{a}_j[1]}, \ldots, g^{b_{\ell+1} \overrightarrow{a}_j[\ell+1]})$.

(c) Client P_j sets and outputs:

$$pk_j = \left(\mathrm{id}_j, \mathrm{pk}_j, \mathrm{rk}_j\right),$$
$$sk_j = \mathrm{sk}_j. \tag{15}$$

(3) $(\phi, \xi) \leftarrow \mathrm{EnFunc}(\overrightarrow{pk}, sk_1, f)$.

(a) On inputting $\overrightarrow{pk}$, public parameter pp, MAC key mk, and function f, for $j = 1, \ldots, n$, client P_1 obtains aggregation key $\mathrm{ak}_j = (g^{(1/\beta_j)})^{\alpha}$.

(b) P_1 sets and outputs:

$$\phi = (f, \mathrm{ak}),$$
$$\xi = \varnothing. \tag{16}$$

(4) $(\chi_1, \tau) \leftarrow \mathrm{EnInput}_1(\overrightarrow{pk}, sk_1, \xi, \overrightarrow{x}_1)$. Client P_1 randomly chooses $\mu \in \mathbb{G}$, parses sk_1 as $(\mathrm{sk}_1, \mathrm{mk})$, parses pk_1 as $(\mathrm{id}_1, \mathrm{pk}_1, \cdot, \cdot)$, and randomly chooses $(r_1, k_1) \xleftarrow{R} \mathbb{Z}_q^2$.

(a) Client P_1 computes σ_1':

$$\sigma_1' = \left(H\left(\mu \| \mathrm{id}_1\right) \cdot \prod_{t=1}^{\ell} g_t^{\overrightarrow{x}_1[t]} \cdot g^{r_1 \beta_1}\right. \tag{17}$$

(b) Client P_1 computes the encryption $\overrightarrow{x}_1$ which consists of $\ell + 3$ components, denoted as $\overrightarrow{c}_1 = (\overrightarrow{c}_1[1], \ldots, \overrightarrow{c}_1[\ell+3])$.

The first component:

$$\overrightarrow{c}_1[1] = g^{k_1}. \tag{18}$$

The following components $\overrightarrow{c}_1[2]$ to $\overrightarrow{c}_1[\ell+1]$ are $\overrightarrow{x}_1$ covered by pk_1 and random number k_1, such that for all $t = 2, \ldots, \ell + 1$:

$$\overrightarrow{c}_1[t] = \mathbf{g}^{\overrightarrow{x}_1[t-1]} \cdot \left(\mathbf{g}^{\overrightarrow{a}_1[t-1]}\right)^{k_1}. \tag{19}$$

The $\ell + 2$-th component in $\overrightarrow{c}_1$ contains the information of random number r_1 instead of an element in $\overrightarrow{x}_1$:

$$\overrightarrow{c}_1[\ell+2] = \mathbf{g}^{r_1} \cdot \left(\mathbf{g}^{\overrightarrow{a}_1[\ell+1]}\right)^{k_1}. \tag{20}$$

Client P_1 sets the last component $\overrightarrow{c}_1[\ell+3] = 2$, in order to remark the state of the cipher text: $\overrightarrow{c}_1$ is a second-level ciphertext:

(c) Sets $\sigma_1 = (\sigma_1', \overrightarrow{c}_1)$ and outputs $(\chi_1, \tau) = (\sigma_1, \mathrm{mk})$.

(5) $\chi_j \leftarrow \mathrm{EnInput}_j(\overrightarrow{pk}, sk_j, \overrightarrow{x}_j)$. For $j = 2, \ldots, n$, client P_j parses sk_1 as $(\mathrm{sk}_1, \cdot)$, parses pk_j as $(\mathrm{id}_j, \mathrm{pk}_j, \cdot)$, and randomly chooses $(r_j, k_j) \xleftarrow{R} \mathbb{Z}_q^2$.

(a) Client P_j computes:

$$\sigma_j' = \left(H\left(\mu \| \mathrm{id}_j\right) \cdot \prod_{t=1}^{\ell} g_t^{\overrightarrow{x}_j[t]} \cdot g^{r_j \beta_j}\right. \tag{21}$$

(b) Client P_j computes the encryption $\overrightarrow{x}_j$ which consists of $\ell + 3$ components, denoted as $\overrightarrow{c}_j = (\overrightarrow{c}_j[1], \ldots, \overrightarrow{c}_j[\ell+3])$.

The first component:

$$\overrightarrow{c}_j[1] = g^{k_j}. \tag{22}$$

The following components $\overrightarrow{c}_j[2]$ to $\overrightarrow{c}_j[\ell+1]$ are information of $\overrightarrow{x}_j$ covered by pk_j and a random number k_j, such that for all $t = 2, \ldots, \ell + 1$:

$$\overrightarrow{c}_j[t] = \mathbf{g}^{\overrightarrow{x}_j[t-1]} \cdot \left(\mathbf{g}^{\overrightarrow{a}_j[t-1]}\right)^{k_j}. \tag{23}$$

The $\ell + 2$-th component in $\overrightarrow{c}_j$ contains the information of a random number r_j instead of $\overrightarrow{x}_j$:

$$\overrightarrow{c}_j[\ell+2] = \mathbf{g}^{r_j} \cdot \left(\mathbf{g}^{\overrightarrow{a}_j[\ell+1]}\right)^{k_j}. \tag{24}$$

Client P_j sets the last component $\overrightarrow{c}[\ell+3] = 2$ to remark the state of the ciphertext: $\overrightarrow{c}_j$ is a second-level ciphertext.

(c) Sets $\sigma_j = (\sigma_j', \overrightarrow{c}_j)$, outputs $(\chi_j, \tau) = (\sigma_j, \mathrm{mk})$, and denotes $\overrightarrow{\chi} = (\chi_1, \ldots, \chi_n)$.

(6) $\omega \leftarrow \mathrm{Compute}(\overrightarrow{pk}, (\phi, \overrightarrow{\chi}))$. Parses χ_j as σ_j, parses pk_1 as $(\cdot, \cdot, \cdot, \mathrm{rk}_1)$, parses pk_j for $j = 2, \ldots, n$ as $(\cdot, \cdot, \mathrm{rk}_j)$, parses ϕ as $(f, \overrightarrow{\mathrm{ak}})$.

(a) For $j = 1, \ldots, n$, the server computes a re-encryption of the client P_j's ciphertext c_j which consists of $2\ell + 3$ components, denoted as $\overrightarrow{c}_j' = (\overrightarrow{c}_j'[1], \ldots, \overrightarrow{c}_j'[2\ell+3])$.

The server first calculates pairings between $\overrightarrow{c}_j[1]$ and each component of rk_j and set the 1-st to $(\ell+1)$-th components to be the pairing result, that is, for all $t = 1, \ldots, \ell + 1$:

$$\overrightarrow{c}'_j[t] = e\left(\overrightarrow{c}_j[1], \left(g^{b_t}\right)^{\overrightarrow{a}_j[t]}\right). \tag{25}$$

For $t = \ell + 2, \ldots, 2\ell + 2$, the server directly sets

$$\overrightarrow{c}'_j[t] = \overrightarrow{c}_j[t - \ell]. \tag{26}$$

Server sets $\overrightarrow{c}'_j[\ell + 3] = R$ to remark the state of the ciphertext, where the $\overrightarrow{c}'_j$ is a re-encrypted ciphertext.

(b) After the server finished the above for each $\overrightarrow{c}_j$, where $j = 1, \ldots, n$, the server begins to evaluate the function f on these re-encrypt ciphertext, and the evaluation is denoted as $\overrightarrow{c}' = (\overrightarrow{c}'[1], \ldots, \overrightarrow{c}'[2\ell + 3])$.

For $j = 1, \ldots, n$ and $t = 1, \ldots, 2\ell + 2$, the server calculates the q_j-th power of $\overrightarrow{c}'_j[t]$. Puts all these powered components together and considers it as a$n \times (2\ell + 2)$ matrix, and the server gets a $(2\ell + 2)$-length vector by adding the n components of the matrix in each column such that for all $t = 1, \ldots, 2\ell + 2$:

$$\overrightarrow{c}'[t] = \prod_{j=1}^{n} \overrightarrow{c}'_j[t]^{q_j}. \tag{27}$$

The server sets the last component $\overrightarrow{c}'[2\ell + 3] = R$ in order to remark the state of the ciphertext, where the new vector denoted as c' is a re-encrypted ciphertext.

(c) The server computes σ' with ak, where

$$\sigma' = \prod_{j=1}^{n} \left(e\sigma_j^{'q_j}, ak_j\right). \tag{28}$$

(d) The server sets and outputs $\omega = (\overrightarrow{c}', \sigma', \mu)$.

(7) $\overrightarrow{y} \cup \{\perp\} \leftarrow \text{Verify}(\overrightarrow{pk}, \xi, \tau, \omega)$. Parses ω as $(\overrightarrow{c}', \sigma', \mu)$, τ as mk, let $\overrightarrow{y} = (y_1, y_2, \ldots, y_\ell)$ denote the result of $f(\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n)$ evaluated by the server, and let r denote the random element evaluated by the server.

(a) If the last component of $\overrightarrow{c}'$ is label "R," the first client P_1 decrypts and obtains and $\perp$ otherwise. For $t = 1, \ldots, \ell$, the server calculates $\mathbf{g}^{y_t}$ and $\mathbf{g}^r$ as follows:

$$\begin{aligned} \mathbf{g}^{y_t} &= \overrightarrow{c}'[t+1] \cdot \overrightarrow{c}'[1]^{(-1/b_t)}, \\ \mathbf{g}^{r} &= \overrightarrow{c}'[\ell+2] \cdot \overrightarrow{c}'[1]^{(-1/b_{(\ell+1)})}. \end{aligned} \tag{29}$$

(b) Solving the discrete log problem and return $\overrightarrow{y}$ if the following equation holds and $\perp$ otherwise:

$$\sigma' \cdot (g^r)^{-\alpha} = \left(\prod_{j=1}^{n} \left(g^{q_j}, H\left(\mu \| \text{id}_j\right)\right) \cdot e\left(\prod_{t=1}^{\ell} \left(g_t^{y_t} g\right)\right)^{\alpha}\right). \tag{30}$$

4.4. Correctness. We now present the proof of the correctness, if the server runs Compute honestly, the result will pass Verify. We start with showing an honest server do return expected $\overrightarrow{y} = f(\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n)$, according to the equation (29), for $t = 0, \ldots, \ell$:

$$\begin{aligned} \mathbf{g}^{y_t} &= \prod_{j=1}^{n} \left(\mathbf{g}^{\overrightarrow{x}_j[t]} \cdot \mathbf{g}^{\overrightarrow{a}_j[t]k_j}\right)^{q_j} \cdot \left(e\left(g^{k_j}, g^{b_t \overrightarrow{a}_j[t]}\right)^{q_j}\right)^{(-1/b_t)} \\ &= \prod_{j=1}^{n} \mathbf{g}^{\overrightarrow{x}_j[t]q_j + \overrightarrow{a}_j[t]k_j q_j - \overrightarrow{a}_j[t]k_j q_j} = \prod_{j=1}^{n} \mathbf{g}^{\overrightarrow{x}_j[t]q_j}, \end{aligned} \tag{31}$$

and, for g^r in equation (29), we can also do the same calculation as above and find $\mathbf{g}^r = \prod_{j=1}^{n} \mathbf{g}^{r_j q_j}$.

Client P_1 requires an alternative decryption strategy for the vector components $(g^{y_1}, \ldots, g^{y_\ell}, g^r)$, as r is uniformly chosen in $\mathbb{Z}_q$ and can thus not be efficiently recovered, when decrypting the vector components $(\mathbf{g}^{y_1}, \ldots, \mathbf{g}^{y_\ell}, \mathbf{g}^r)$, and client P_1 cannot directly obtain the plaintext. Fortunately, obtaining $r \in \mathbb{Z}_q$ is unnecessary, and it is sufficient to unblind the signature by g_r (resp., $\mathbf{g}^r$). We now present the left side and the right side of the equation (30) always keeps equal as long as the server remains honest. Here, we re-emphasize that g_t is randomly chosen from $\mathbb{G}$ and do not mix it up with the generator g of group $\mathbb{G}_T$ The left-hand side of the equation (30) is equal to

$$\begin{aligned} \text{LHS} &= \prod_{j=1}^{n} e\left(\sigma_j^{'q_j}, \text{ak}_j\right) \cdot (\mathbf{g}^r)^{-\alpha}\Big) = \prod_{j=1}^{n} e\left(H\left(\mu_j \| \text{id}_j\right)\right) \\ &\quad \cdot \prod_{t=1}^{\ell} \left(g_t^{\overrightarrow{x}_j[t]} \cdot g^{r_j}\right)^{\beta_j q_j}, g^{(\alpha/\beta_j)}\Big) \cdot \mathbf{g}^{-\alpha r}\Big) \\ &= \prod_{j=1}^{n} e\left(\left(H\left(\mu_j \| \text{id}_j\right)\right) \cdot \prod_{t=1}^{\ell} \left(g_t^{\overrightarrow{x}_j[t]q_j}, g^{\alpha}\right)\right), \end{aligned} \tag{32}$$

and the right-hand side of the equation (30) is equal to

$$\begin{aligned}\mathrm{RHS} &= \left(\prod_{j=1}^{n} e\left(g^{q_j}, H\left(\mu_j \| \mathrm{id}_j\right)\right)\right).e\left(\prod_{t=1}^{\ell}\left(g_t^{\sum j=1^{n}\overrightarrow{x}_j[t]\cdot q_j g}\right)\right)^{\alpha} \\ &= \prod_{j=1}^{n} e\left(g^{\alpha}, H\left(\mu_j \| \mathrm{id}_j\right)^{q_j}\right)\cdot \prod_{j=1}^{n} e\left(\prod_{t=1}^{\ell}\left(g_t^{\overrightarrow{x}_j[t]\cdot q_j} g^{\alpha}\right)\right) = \prod_{j=1}^{n} e\left(\left(H\left(\mu_j \| \mathrm{id}_j\right)\right)\cdot \prod_{t=1}^{\ell}\left(g_t^{\overrightarrow{x}_j[t] q_j}, g^{\alpha}\right)\right).\end{aligned} \tag{33}$$

We can clearly find that the two sides of the equation (30) are equal and thus conclude that our detailed construction meets the correctness definition. Furthermore, following the security proof of general construction, the detailed construction is a sound, private against the first client, and private against server MVC scheme.

5. Implementation

To measure the performance of our transform, we implement concrete instantiation described in Section 5 and conduct a variant of experiments and focus on evaluating the performance overhead. All the experiments are run on a PC with an Intel Core i7-4790K CPU running at 3.60 GHz and 8 GB of RAM on ubuntu-16.04.1. The implementation is in C with the help of PBC (pairing-based cryptography) library, which is designed to be the backbone of implementations of pairing-based cryptosystems [22]. More specifically, we use the symmetric pairings constructed on the curve $y^2 = x^3 + x$ over the field F_q for some prime $q = 3 \bmod 4$ with $|q| = 512$.

5.1. Client Computation. Following the theoretical analysis, the length ℓ of the input vector, the maximum size $x_{\max}$ of the input vector element (any element in the input vector $\overrightarrow{x}_j^{(i)}$ of client P_j is smaller than $2^{x_{\max}}$), and the number of clients n in total largely influence the runtime of the algorithms. We give some notations for the basic cost units of our protocol: $(\mathrm{Mul}_{\mathbb{G}}/\mathrm{Mul}_{\mathbb{G}_T})$ represents the runtime of a multiplication in $(\mathbb{G}/\mathbb{G}_T)$. $(\mathrm{Exp}_{\mathbb{G}}/\mathrm{Exp}_{\mathbb{G}_T})$ represents the runtime of an exponentiation in $(\mathbb{G}/\mathbb{G}_T)$. $\mathrm{Pair}_{\mathbb{G}}$ represents the runtime of a pairing in $\mathbb{G}$. Log represents the runtime of solving a discrete log problem.

Key generation. The key generation process includes both KeyGen_1 and KeyGen_j. We theoretically analyzed the computational performance of our key generation process, which is shown in Table 1. Figure 4 depicts the key generation process cost for our scheme, and we set the maximum size $x_{\max}$ of input vector element to a fixed size 20 bits and observe how the total number n and the length ℓ of the input vector affect the performance:

(i) When n remains constant, the runtime of the key generation process changes uniformly with respect to ℓ.
(ii) When ℓ remains constant, the runtime of the key generation process changes uniformly with respect to n.
(iii) Compared to n, ℓ can determine the runtime to a greater extent.

Function encoding. We did not conduct the experimental analysis of function encoding process which only contains EncFunc, and as shown in Table 1, the runtime changes uniformly with respect to the total number of the clients.

Input encoding. We did not conduct the experimental analysis of clients' encoding process which only contains EnInput, as shown in Table 1, and the runtime changes uniformly with respect to the length ℓ of the input vector.

The algorithm Verify, involving computing discrete logarithm, is the only step in the verification process. Pollard's kangaroo algorithm [29] is the known fastest algorithm to solve the discrete logarithm problem for a general elliptic curve, and the time complexity is $O(\sqrt{x})$. x is the size of the input number of Pollard's kangaroo algorithm, which is mainly determined by the maximum size $x_{\max}$ of input vector element. Based on the theoretical analysis in Table 1, we can find that the verification process requires a total of ℓ discrete logarithmic calculations. We fix n to 2 as it has relatively small influence on runtime, and the algorithm performance on different $x_{\max}$ and ℓ is given in Figure 5 with log 2-type Y axis:

(i) When $x_{\max}$ remains constant, the runtime of verification process increases along with ℓ.
(ii) When ℓ remains constant, the runtime of verification process increases along with $x_{\max}$.
(iii) The increase in $x_{\max}$ has a greater impact on the runtime of verification process than the increase in ℓ.

Note that each ℓ discrete logarithm is computed separately; as a consequence, the ℓ computation can be parallelized and will greatly save the client's runtime. However, as we assume that clients are usually devices with limited computing power, performing parallel computation is obviously against our assumptions, so we do not implement Verify algorithm in parallelized way.

5.2. Server Computation. As server only carries out Compute, the experiments of the server's runtime focus on the parameters influence on the efficiency of Compute. We fix $x_{\max}$ to 20 bits for simplicity, and the results plotted in Figure 6 and Table 1 show the server's runtime in relation to the total number n of the clients and the input vector length ℓ:

(i) When n is fixed, the server's runtime increases along with the length ℓ.
(ii) When ℓ is fixed, the server's runtime increases along with total number n of the clients.

Table 1: Computational complexity of our MVC

	$\text{Mul}_{\mathbb{G}}$	$\text{Exp}_{\mathbb{G}}$	$\text{Mul}_{\mathbb{G}_T}$	$\text{Exp}_{\mathbb{G}_T}$	$\text{Pair}_{\mathbb{G}}$	Log
Key generation	0	$2\ell + n + 4$	0	$\ell + 1$	0	0
Function encoding	0	n	0	0	0	0
Input encoding	$\ell + 1$	$\ell + 3$	ℓ	2ℓ	0	0
Verification process	$\ell - 1$	$\ell + n$	$\ell + n + 2$	$\ell + 4$	$\ell + n$	ℓ
Server computation	0	n	$(2\ell + 3)(n - 1) + 1$	$(2\ell + 2)n$	$\ell + n + 1$	0

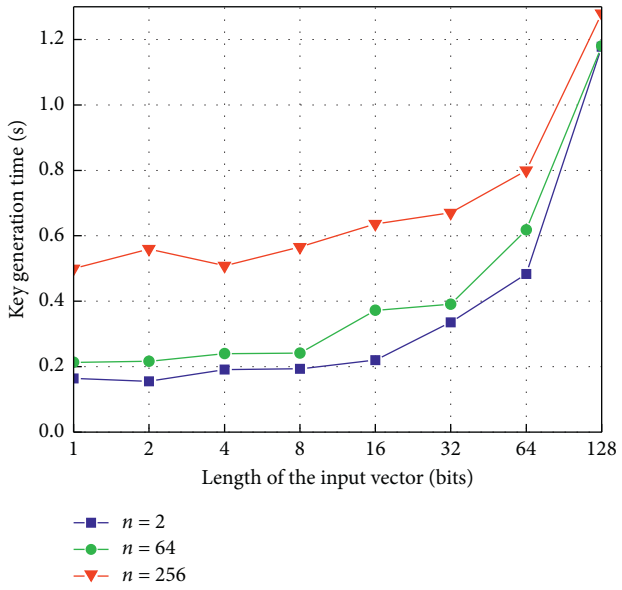

Figure 4: Client's key generation time.

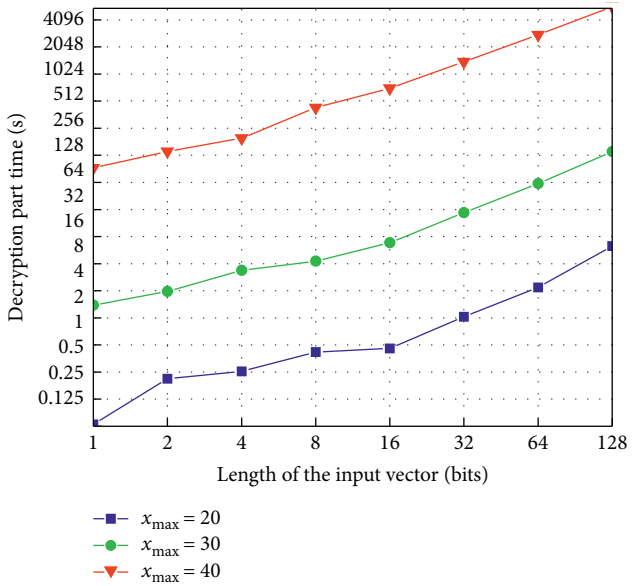

Figure 5: Client's decryption time.

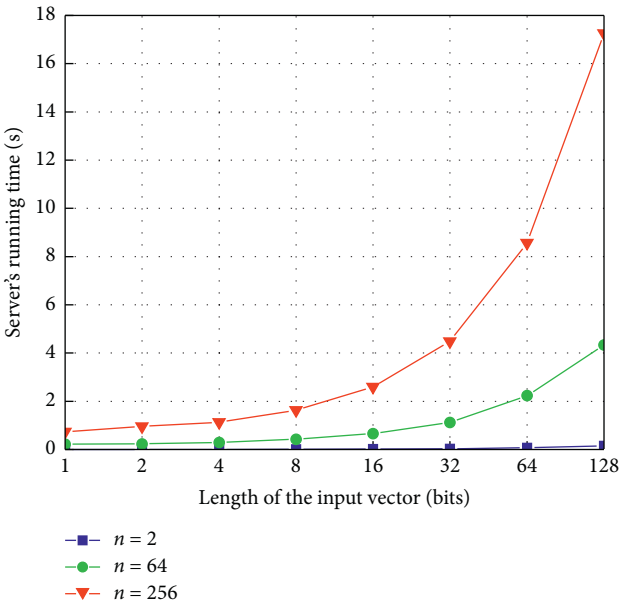

Figure 6: Server's runtime.

5.3. Analysis and Comparison

5.3.1. Efficiency Analysis. The client computation overhead may be even higher than that of the server in some cases. According to both theoretical and experimental analyses, we show how to avoid these inefficiencies and under what circumstances an instance can be regarded as an efficient solution. Assume that the computational complexity of all group operations is constant and that $x_{\max} + \log_2 n - 1$ is the expected length of the evaluation results of the linear combinations of the n clients' input vector with the maximum element size $x_{\max}$. We hope that the runtime of any client in the scheme will be less than that of the server; however, client P_1 runs most algorithms on the client side. Here, we denote the computational complexity of P_1 as T_{client}:

$$T_{\text{client}} = 12\ell + 5n + 12 + \ell\sqrt{x_{\max} + \log_2 n - 1},$$
$$T_{\text{client}} \leq 7n + (4n - 1)\ell - 1. \tag{34}$$

By the above inequality, one can indicate that when n becomes sufficiently large and $x_{\max}$ stays rather small, the runtime required on the server side will exceed the runtime required on the client side, and the gap will become more significant as n grows. Although the runtime of both sides is $O(n)$, the coefficient of n on the server side is obviously larger and hence more sensitive to the increase in n. We fixed ℓ to 128 and let $x_{\max}$ be rather small, i.e., 20 bits. These theoretical analyses correspond well with the experimental results shown in Figure 7.

5.3.2. Performance Relative to Prior Scheme. Parno et al. [4] built a model to estimate the performance of FHE + GC-based VC schemes according to the results for FHE [30]. Because of the similarities between FHE + GCs based VC

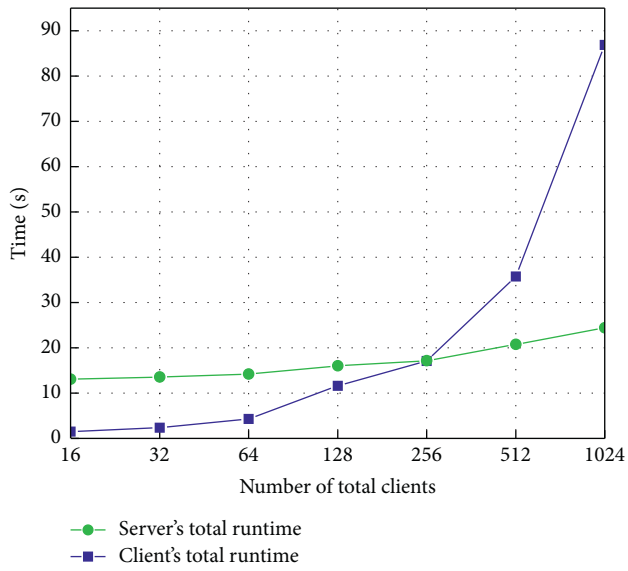

Figure 7: Total runtime.

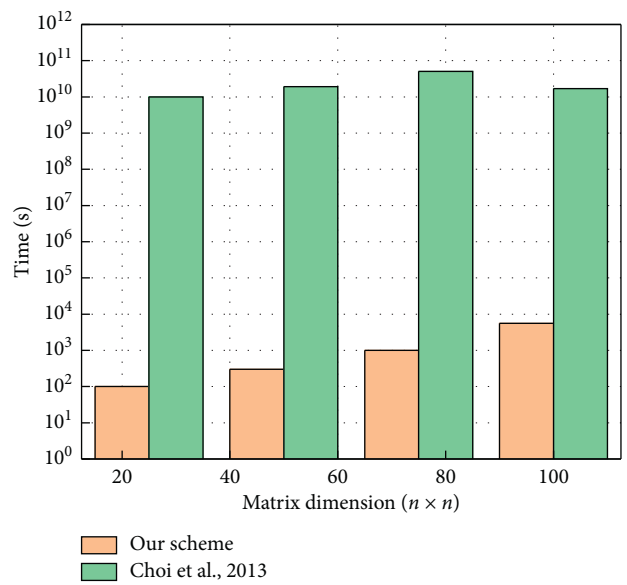

Figure 8: Comparison with [2].

and MVC, we can likewise use the method to estimate the performance of MVC.

Parno et al. [4] used multiplication of two $(t \times t)$ matrices M_1 and M_2 as test application, and each component in M_1 and M_2 is at most 32 bits. As the multiplication of two $(t \times t)$ matrices can be viewed as $(t \times t)$ linear combinations of vectors with length t, we set both the clients' number n and the length ℓ of the input vectors to t, and let the maximum size $x_{\max}$ of input vector elements equal to 32 bits in order to construct an MVC with the same computational scaling as above. We plot the time complexity for the verification process in Figure 8. Throughout the results, it can be observed that our scheme significantly improves performance and can be considered applicable in a practical setting.

6. Conclusions

We combined MVC with an efficient cryptographic protocol HPRA by providing a general transform. We also gave and implemented a concrete instantiation of MVC scheme for outsourcing the linear combinations of vectors over a finite field. The instantiation and implementation could give the way for the future research on the designs and optimizations of outsourcing computation. Our implementation requires solving discrete logarithm problem. How to avoid the usage of these expensive operations is an open question for further research.

Appendix

A. Details of the Modification to Scheme 4 in [3]

Intuitively speaking, Scheme 4 in [3] combines their HPRA scheme, Scheme 2 without output privacy with a homomorphic proxy re-encryption scheme (HPRE), and Scheme 3 to construct an output private scheme which consists of nine algorithms (Gen, SGen, VGen, Sign, Verify, SRGen, VRGen, Agg, *A*Verify) as follows:

(i) $\mathrm{Gen}(1^\kappa)$: fix an HPRE = $(\mathcal{P}, \mathcal{G}, \overrightarrow{\mathcal{E}}, \overrightarrow{\mathcal{D}}, \mathcal{RG}, \mathcal{RE}, \mathcal{EV})$ for class $\mathcal{F}_{\mathrm{lin}}$ and HPRA.$\mathcal{EV}$ = (**Gen**, **SGen**, **VGen**, **Sign**, **Verify**, **SRGen**, **VRGen**, **Agg**, **AVerify**) from Scheme 2 such that $\mathcal{M}_{\mathrm{PRA}} \subseteq \mathcal{M}_{\mathrm{PRE}}$, run $pp_s \leftarrow \mathrm{Gen}(1^\kappa, \ell)$, $pp_e \leftarrow \mathcal{P}(1^\kappa, \ell + 1)$ and output $pp \leftarrow (pp_s, pp_e)$.

(ii) $\mathrm{SGen}(pp)$: run $(\mathbf{id}, \mathbf{sk}, \mathbf{pk}) \leftarrow \mathbf{SGen}(pp_s)$, $(\mathrm{rsk}, \mathrm{rpk}) \leftarrow \mathcal{G}(pp_e)$ and output $(\mathrm{id}, sk, pk) \leftarrow (\mathbf{id}, (\mathbf{sk}, \mathrm{rsk}, \mathrm{rpk}), \mathbf{pk})$.

(iii) $\mathrm{VGen}(pp)$: carry out $(\mathbf{mk}, \mathbf{aux}) \leftarrow \mathbf{VGen}(pp_s)$, $(\mathrm{rsk}, \mathrm{rpk}) \leftarrow \mathcal{G}(pp_e)$ and output $(\mathrm{mk}, \mathrm{aux}) \leftarrow ((\mathbf{mk}, \mathrm{rsk}), (\mathbf{aux}, \mathrm{rpk}))$.

(iv) $\mathrm{Sign}(sk, \overrightarrow{m}, \tau)$: parse sk as $(\mathbf{sk}, \cdot, \mathrm{rpk})$, choose $r \xleftarrow{R} \mathbb{Z}_q$, and output $\sigma \leftarrow (\sigma' \cdot g^r, \overrightarrow{c})$, where

$$\begin{aligned} &(\sigma', \cdot) \leftarrow \mathbf{Sign}(\mathbf{sk}, \overrightarrow{m}, \tau), \\ &\overrightarrow{c} \leftarrow \mathcal{E}^2_{\ell+2}(\mathrm{rpk}, \overrightarrow{m} \| r). \end{aligned} \tag{A.1}$$

(v) $\mathrm{SRGen}(sk_i, \mathrm{aux})$: parse sk_i as $(\mathbf{sk}_i, \mathrm{rsk}_i, \mathrm{rpk}_i)$ and aux as $(\mathbf{aux}, \mathrm{rpk})$. Obtain $\mathbf{rk}_i \leftarrow \mathbf{SRGen}(\mathbf{sk}_i, \mathbf{aux})$ and $\mathrm{prk}_i \leftarrow \mathcal{RG}(\mathrm{rsk}_i, \mathrm{rpk})$ and return $\mathrm{rk}_i \leftarrow (\mathbf{rk}_i, \mathrm{prk}_i)$.

(vi) $\mathrm{VRGen}(pk_i, \mathrm{mk}, \mathrm{rk}_i)$: parse pk_i as $\mathbf{pk}_i$ and mk as $(\mathbf{mk}, \cdot)$, obtain $\mathbf{ak}_i \leftarrow \mathbf{VRGen}(\mathbf{pk}_i, \mathbf{mk})$, and return $\mathrm{ak}_i \leftarrow (\mathbf{ak}_i, \mathrm{rk}_i)$.

(vii) $\mathrm{Agg}((ak_i)_{i\in[n]}, (\sigma_i)_{i\in[n]}, \tau, f)$: for $i \in [n]$, parse ak_i as $(\mathbf{ak}_i, (\mathbf{rk}_i, \mathrm{prk}_i))$ and σ_i as $(\sigma'_j \cdot g^r, \overrightarrow{c}_i)$. Output $\Lambda \leftarrow (\overrightarrow{c}', \mu, \tau)$, where

$$\begin{aligned} &(\overrightarrow{c}'_j \leftarrow \mathcal{RE}(\mathrm{prk}_i, \overrightarrow{c}_i)_{i\in[n]}, \\ &(\overrightarrow{c}', \mu, \tau) \leftarrow \mathbf{Agg}(\mathbf{ak}_i)_{i\in[n]}, (\sigma'_i \cdot g^r, c'_i)_{i\in[n]}, f). \end{aligned} \tag{A.2}$$

(viii) $A\mathrm{Verify}(\mathrm{mk}, \Lambda, \mathrm{ID}, f)$: parse mk as (**mk**, rsk) and Λ as $(\vec{c}', \mu, \tau)$, get $\vec{m}'\|r \leftarrow \mathscr{D}^1_{\ell+1}(\mathrm{rsk}, \vec{c})$ and return $(\vec{m}, \tau)$ if the following holds and $(\perp, \perp)$ otherwise:

$$\mathbf{AVerify}\left(\mathbf{mk}, \left(\vec{m}, \mu \cdot (g^r)^{-1}, \tau\right), \mathrm{ID}, f\right) = 1. \tag{A.3}$$

Because one can verify guesses for the signed messages only according to the signatures, the signature $\sigma_{j,}$ of signer P_j is blinded with a random element g^{r_j} by setting $\sigma_j \leftarrow (\sigma'_j \cdot g^{r_j})$. According to the definition of $A\mathrm{Verify}(\cdot)$ and $\mathbf{AVerify}(\cdot)$, we know that $A\mathrm{Verify}(\cdot)$ returns if and only if the following equation holds:

$$\mu \cdot (\mathbf{g}^r)^{-1} = \prod_{i=1}^{n} e\left(g_i^{\omega}, H\left(\tau \| g^{\beta_i}\right)\right) \cdot e\left(\prod_{i=1}^{\ell} (g_i^{m_i} g)^{\alpha}\right). \tag{A.4}$$

We can see that $(\mathbf{g}^r)^{-1}$, where $r = \sum_{j=1}^{n} r_j \cdot q_j$, is multiplied on the left-hand side of equation (A.4) to eliminate the effect of the random elements g^{r_j}. Here, we offer more details on the calculation of both sides of equation (A.4). For the sake of simplicity, ℓ takes 1, and then, the left-hand side is equal to

$$\begin{aligned} \mathrm{LHS} &= \prod_{j=1}^{n} e\left(\left(H\left(\mu_j \| \mathrm{id}_j\right) \cdot g_1^{m_j}\right)^{\beta_j \omega_j} \cdot g^{r_j \omega_j}, \left(g^{1/\beta_j}\right)^{\alpha}\right) \cdot (\mathbf{g}^r)^{-1} \\ &= \prod_{j=1}^{n} \left(e\left(\left(H\left(\mu_j \| \mathrm{id}_j\right) \cdot g_1^{m_j}\right)^{\omega_j}, g^{\alpha}\right) \cdot \left(g^{r_j \omega_j}\right)^{\alpha/\beta_j}\right) \cdot (\mathbf{g}^r)^{-1}. \end{aligned} \tag{A.5}$$

The right-hand side of the equation (A.4) is equal to

$$\begin{aligned} \mathrm{RHS} &= \left(\prod_{j=1}^{n} e\left(g^{\omega_j}, H\left(\mu_j \| \mathrm{id}_j\right)\right) \cdot e\left(g_i^{\sum} j = 1^{n} x_j \cdot \omega_i, g\right)\right)^{\alpha} \\ &= \prod_{j=1}^{n} e\left(g^{\alpha}, H\left(\mu_j \| \mathrm{id}_j\right)^{\omega_j}\right) \cdot \prod_{j=1}^{n} e\left(g_1^{x_j \cdot \omega_j}, g^{\alpha}\right) \\ &= \prod_{j=1}^{n} e\left(\left(H\left(\mu_j \| \mathrm{id}_j\right) \cdot g_1^{x_j}\right)^{\omega_j}, g^{\alpha}\right). \end{aligned} \tag{A.6}$$

The two sides of equation (A.4) are not equal, thus Scheme 4 does not meet the correctness definition. The equation will be equal if the last term on the left-hand side becomes $(\mathbf{g}^r)^{-\sum_j (\alpha/\beta_j)}$. However, this is unreasonable, as β_j is part of the j-th signer's private key sk_j while $A\mathrm{Verify}$ is run by the receiver.

To fix this flaw, we blind the input of P_j with a random element g^{r_j} by the set $(\sigma'_j, \cdot) \leftarrow \mathbf{Sign}(\mathbf{sk}_j, \vec{x}_j \| r_j, \mu)$, instead of blinding the signature of P_j with a random element g^{r_j}, and prevent the verification of guesses for signed messages as well. In this way, the evaluation of items related to random elements r_j and items related to inputs x_j are consistent, which makes the item related to r_j on the left side of equation (A.4) become $\mathbf{g}^{-\alpha r}$ and removes β_j. In detail, we modify Sign, Agg, and $A\mathrm{Verify}$ as follows:

(i) $\mathrm{Sign}(sk, \vec{m}, \tau)$: parse sk as $(\mathbf{sk}, \cdot, \mathrm{rpk})$, select $r \xleftarrow{R} \mathbb{Z}_q$, and output $\sigma \leftarrow (\sigma', \vec{c})$, where

$$\begin{aligned} &(\sigma', \cdot) \leftarrow \mathbf{Sign}(\mathbf{sk}, \vec{m} \| r, \tau), \\ &\vec{c} \leftarrow \mathscr{E}^2_{\ell+2}(\mathrm{rpk}, \vec{m} \| r). \end{aligned} \tag{A.7}$$

(ii) $\mathrm{Agg}((\mathrm{ak}_i)_{i\in[n]}, (\sigma_i)_{i\in[n]}, \tau, f)$: for $i \in [n]$ parse ak_i as $(\mathbf{ak}_i, (\mathbf{rk}_i, \mathrm{prk}_i))$, σ_i as $(\sigma'_i, \vec{c}_i)$. Return $\Lambda \leftarrow (\vec{c}', \mu, \tau)$, where

$$\begin{aligned} &\left(\vec{c}'_j \leftarrow \mathscr{R}\mathscr{E}(\mathrm{prk}_i, \vec{c}_i)\right)_{i\in[n]}, \\ &(\vec{c}', \mu, \tau) \leftarrow \mathbf{Agg}(\mathbf{ak}_i)_{i\in[n]}, (\sigma'_i, c'_i)_{i\in[n]}, f). \end{aligned} \tag{A.8}$$

(iii) $A\mathrm{Verify}(\mathrm{mk}, \Lambda, \mathrm{ID}, f)$: parse mk as (**mk**, rsk) and Λ as $(\vec{c}', \mu, \tau)$, get $\vec{m}'\|r \leftarrow \mathscr{D}^1_{\ell+1}(\mathrm{rsk}, \vec{c}')$, and return $(\vec{m}, \tau)$ if the equation as follows holds and $(\perp, \perp)$ otherwise:

$$\mathbf{AVerify}\left(\mathbf{mk}, \left(\vec{m}, \mu \cdot (\mathbf{g}^r)^{-\alpha}, \tau\right), \mathrm{ID}, f\right) = 1. \tag{A.9}$$

B. Proof of Security

We have already proved the correctness in Section 5, so we omit it here. Now we need to prove that this modified HPRA still maintains input privacy, T-unforgeability, and output privacy.

In fact, input privacy, aggregator unforgeability, and output privacy are not affected by the modification. Our blinding approach extends the vector $\vec{m}$ with the element r; i.e., the second input of **Sign** changes from $\vec{m}$ to $\vec{m}\|r$, which certainly does not affect these properties. The input privacy of the output private HPRA can be reduced to the input privacy of Scheme 2 [3].

The T-unforgeability is the only property that may be affected, more precisely, aggregator unforgeability. Since signing an input vector $\vec{m}_i$ or signing an extending input vector $\vec{m}_i$ with and random element r_i, $\vec{m}_i\|r_i$ do not influence the signer unforgeability.

The T-unforgeability of the output private HPRA can be reduced to the aggregator unforgeability of Scheme 2 when $T =$ "Aggregator, which can be further reduced to the eBCDH assumption [3]. Here, we re-certify following the original proof in [3]. We first give the definition of eBCDH assumption.

Definition 9. (eBCDH). The extended bilinear computational Diffie–Hellman assumption holds relative to $\mathrm{BG} \leftarrow \mathrm{BGGen}(1^{\kappa})$; if for all PPT adversaries $\mathscr{A}$, there is a negligible function $\mathrm{negl}(\cdot)$ such that

$$\Pr\left[\begin{matrix} u, v, w \xleftarrow{R} \mathbb{Z}_q & : h = e(g, g)^{uvw} \\ h \leftarrow \mathscr{A}\left(BG, g^{1/u}, g^{u}, g^{v}, g^{w}\right) & \end{matrix}\right] \leq \mathrm{negl}(\kappa). \tag{B.1}$$

If there is an efficient algorithm $\mathscr{A}$ breaking T-unforgeability, then we can design an algorithm $\mathscr{R}$ that breaks the eBCDH assumption. $\mathscr{R}$ maintains two sets: Rnd and H, both are initially empty. $\mathscr{R}$ obtains an $e\text{BCDH}_\kappa \leftarrow (g^{(1/\beta)}, g^\beta, g^{(\alpha/\beta)}, g^\gamma)$ related to the security parameter κ and executes the modified algorithm Gen' to obtain pp as follows:

(i) $\text{Gen}'(e\text{BCDH}_\kappa, \ell)$: for $i \in [\ell]$, select $\text{Rnd}[i] \leftarrow (s_i, t_i) \xleftarrow{R} \mathbb{Z}_q^2$ and set $g_i \leftarrow (g^\gamma)^{s_i} g^{t_i}$. Fix H: $\mathbb{Z}_q \longrightarrow \mathbb{G}$ and output $pp \leftarrow (\text{BG}, H, (g_i)_{i\in[\ell]}, \ell)$.

(ii) Then, $\mathscr{R}$ starts $\mathscr{A}$ on inputting pp and aux = $\varnothing$, where $\mathscr{R}$ simulates the oracles as follows:

(iii) $H(x)$: if $x \in H$ return $H[x][1]$. Otherwise, choose $(\rho, \nu) \xleftarrow{R} \mathbb{Z}_q^2$, set $H[x] \leftarrow ((g^\gamma)^\rho g^\nu, \rho, \nu)$, and return $H[x][1]$.

(iv) $\text{SG}(i)$: choose $\xi \xleftarrow{R} \mathbb{Z}_q^*$ and set $S[i] \leftarrow (\xi, (g^\beta)^\xi, (g^{(1/\beta)})^{(1/\xi)})$. Then, return $((g^\beta)^\xi, (g^{(1/\beta)})^{(1/\xi)})$.

(v) $\text{Sig}((j_i)_{i\in[n]}, (\vec{m}_{j_i})_{i\in[n]})$ work as the original oracl except: randomly choose $\tau \xleftarrow{R} \mathbb{Z}_q$, and for all $i \in [n]$, choose $(\nu_i, r_i) \xleftarrow{R} \mathbb{Z}_q^2$. If $\tau \| S[j_i][2] \in H$ for any $i \in [n]$, abort. Otherwise, for all $i \in [n]$, set $\rho_i \leftarrow -\sum_{k\in[\ell]} \text{Rnd}[k][1] \cdot m_{j_i}[k] \cdot \text{Rnd}[k][2]$ and $H[\tau \| S[j_i][2]] \leftarrow ((g^\gamma)^{\rho_i} g^{\nu_i}, \rho_i, \nu_i)$, compute $\sigma \leftarrow \nu_i + \sum_{k\in[\ell]} [m_{j_i}k] \cdot r_i \cdot \text{Rnd}[k][2]$, and return $(\sigma_{j_i} \leftarrow S[j_i][2]^\sigma)_{i\in[n]}$ and τ.

(vi) $\text{VR}(i)$: if $S[i] = \perp$, $\text{RK}[i] = \perp$, $\text{AK}[i] \neq \perp$, and return $\perp$. Otherwise, set $\text{AK}[i] \leftarrow (g^{(\alpha/\beta)})^{(1/S[i][1])}$.

If $\mathscr{A}$ finally returns a valid forgery $(\Lambda^*, \text{ID}^*, f^*) = ((\vec{m}^*, \mu^*, \tau^*), \text{ID}^*, f^*)$ with $f^* = (w_i^*)_{i\in[n]}$, we know that it is of the following form:

$$\begin{aligned}\mu^* &= \left(e\left(\prod_{i\in[\ell]}\left(g_i^{m_i^*} g\right) \cdot \prod_{i\in[n]} e\left(g^{w_i^*}, H\left(\tau^* \| \text{id}_i^*\right)\right)^\alpha \cdot (\mathbf{g}^\alpha)^r\right)\right) \\ &= (g^{\alpha\gamma})^{\sum_{i\in[\ell]} \text{Rnd}_i^*[i][1]} (g^\alpha)^{\sum_{i\in[\ell]} \text{Rnd}_i^*[i][2]} \cdot (\mathbf{g}^\alpha)^r \\ &\quad \cdot (g^{\alpha\gamma})^{\sum_{i\in[n]} [w_i^* \cdot H[\tau^* \| pk_i^*][2]]} (\mathbf{g}^\alpha)^{\sum_{i\in[n]} [w_i^* \cdot H[\|\tau^* pk_i^*][3]]}.\end{aligned} \tag{B.2}$$

Then, we let

$$\begin{aligned}\varphi &= \sum_{i\in[\ell]} \text{Rnd}_i^*[i][1] + \sum_{i\in[n]} w_i^* \cdot H\left[\tau^* \| pk_i^*\right][2] + r, \\ \nu &= \sum_{i\in[\ell]} \text{Rnd}_i^*[i][2] + \sum_{i\in[n]} w_i^* \cdot H\left[\tau^* \| pk_i^*\right][3],\end{aligned} \tag{B.3}$$

and output $\mathbf{g}^{\alpha\gamma} \leftarrow (\mu^* \cdot e(g^\beta, g^{(\alpha/\beta)})^{-\nu})^{(1/\varphi)}$ as eBCDH solution. The simulation above can be considered indistinguishable to the original security game: all the values are identically distributed, while the signature is uniquely determined by the responses of the random oracle and the key also indistinguishable to a real signature. Thus, if there is an efficient algorithm $\mathscr{A}$ breaking aggregator unforgeability, we can construct an efficient algorithm $\mathscr{R}$ which turns $\mathscr{A}$ into an efficient eBCDH solver, which is in conflict with eBCDH assumption.

Data Availability

No data were used to support this study.

Disclosure

A preliminary version of this paper appears in the Proceedings of the 6th International Conference on Information Systems Security and Privacy, 2020, under the title "A Homomorphic Proxy Re-Authenticators Based Efficient Multi-Client Non-Interactive Verifiable Computation Scheme."

Conflicts of Interest

The author declares that there are no conflicts of interest.

References

[1] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *Advances in Cryptology—CRYPTO 2010*, pp. 465–482, Springer, Berlin, Germany, 2010.

[2] S. G. Choi, J. Katz, R. Kumaresan et al., "Multi-client non-interactive verifiable computation," in *Theory of Cryptography*, pp. 499–518, Springer, Berlin, Germany, 2013.

[3] D. Derler, S. Ramacher, and D. Slamanig, "Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, pp. 124–142, Springer, Anguilla, UK, 2017.

[4] B. Parno, J. Howell, C. Gentry et al., "Pinocchio: nearly practical verifiable computation," in *Proceedings of the IEEE symposium on security and privacy*, pp. 238–252, IEEE, Berkeley, CA, USA, May 2013.

[5] S. Setty, B. Braun, V. Vu et al., "Resolving the conflict between generality and plausibility in verified computation," in *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 71–84, ACM, Prague, Czech Republic, April 2013.

[6] R. Gennaro, C. Gentry, B. Parno et al., "Quadratic span programs and succinct NIZKs without PCPs," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 626–645, Springer, Athens, Greece, May 2013, Advances in Cryptology—EUROCRYPT 2013 p.

[7] E. Ben-Sasson, A. Chiesa, D. Genkin et al., "SNARKs for C: verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology-CRYPTO 2013*, pp. 90–108, Springer, Berlin, Germany, 2013.

[8] E. Ben-Sasson, A. Chiesa, E. Tromer et al., "Succinct non-interactive zero knowledge for a von Neumann architecture," in *Proceedings of the USENIX Security Symposium*, pp. 781–796, Washington, DC, USA, August 2014.

[9] M. Backes, M. Barbosa, D. Fiore et al., "ADSNARK: nearly practical and privacy-preserving proofs on authenticated data," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 271–286, IEEE, San Jose, CA, USA, May 2015.

[10] R. S. Wahby, S. T. V. Setty, Z. Ren et al., "Efficient RAM and control flow in verifiable outsourced computation," in

Proceedings 2015 Network and Distributed System Security Symposium, San Diego, CA, USA, February 2015.

[11] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proceedings of the Annual Cryptology Conference*, pp. 111–131, Springer, Santa Barbara, CA, USA, May 2011, Advances in Cryptology—CRYPTO 2011.

[12] B. Braun, A. J. Feldman, Z. Ren et al., "Verifying computations with state," in *Proceedings of the twenty-fourth ACM Symposium on Operating Systems Principles*, pp. 341–357, ACM, Farmington, MI, USA, November 2013.

[13] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pp. 90–112, ACM, Cambridge, MA, USA, January 2012.

[14] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography*, pp. 222–242, Springer, Berlin, Germany, 2013.

[15] J. Ye, Z. Xu, and Y. Ding, "Secure outsourcing of modular exponentiations in cloud and cluster computing," *Cluster Computing*, vol. 19, no. 2, pp. 811–820, 2016.

[16] X. Bultel, M. L. Das, H. Gajera et al., "Verifiable private polynomial evaluation," in *Proceedings of the International Conference on Provable Security*, pp. 487–506, Springer, Singapore, Singapore, December 2017.

[17] S. Xu, Y. He, and L. F. Zhang, "Cryptanalysis of tran-pang-deng verifiable homomorphic encryption," in *Proceedings of the International Conference on Information Security and Cryptology*, pp. 59–70, Springer, Xi'an, China, November 2017.

[18] S. D. Gordon, J. Katz, F. H. Liu et al., "Multi-client verifiable computation with stronger security guarantees," in *Proceedings of the Theory of Cryptography Conference*, pp. 144–168, Springer, Warsaw, Poland, March 2015.

[19] S. Goldwasser, S. D. Gordon, V. Goyal et al., "Multi-input functional encryption," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 578–602, Springer, Copenhagen, Denmark, May 2014.

[20] D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin, "Multi-key homomorphic authenticators," *IET Information Security*, vol. 13, no. 6, pp. 618–638, 2019.

[21] L. Schabhüser, D. Butin, and J. Buchmann, "Context hiding multi-key linearly homomorphic authenticators," in *Proceedings of the Topics in Cryptology—CT-RSA*, pp. 493–513, Springer, San Francisco, CA, USA, March 2019.

[22] https://crypto.stanford.edu/pbc/.

[23] B. Lynn, *On the Implementation of Pairing-Based cryptosystems*, Stanford University, Stanford, CA, USA, 2007.

[24] A. Miyaji, M. Nakabayashi, and S. Takano, "New explicit conditions of elliptic curve traces for FR-reduction," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 84, no. 5, pp. 1234–1243, 2001.

[25] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proceedings of the International Workshop on Selected Areas in Cryptography*, pp. 319–331, Springer, Waterloo, Canada, August 2005.

[26] D. Freeman, "Constructing pairing-friendly elliptic curves with embedding degree 10," in *Proceedings of the International Algorithmic Number Theory Symposium*, pp. 452–465, Springer, Berlin, Germany, July 2006.

[27] D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two," *IEEE Transactions on Information Theory*, vol. 30, no. 4, pp. 587–594, 1984.

[28] E. Barker and Q. Dang, "NIST special publication 800-57 Part 1, revision 4," NIST, Technical Report, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2016.

[29] J. M. Pollard, "Monte Carlo methods for index computation $({\rm mod}\ p)$," *Mathematics of Computation*, vol. 32, no. 143, p. 918, 1978.

[30] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Proceedings of the Annual Cryptology Conference*, pp. 850–867, Springer, Santa Barbara, CA, USA, August 2012.