A Homomorphic Proxy Re-authenticators based Efficient Multi-client Non-interactive Verifiable Computation Scheme

Shuaijianni Xu[1,2] and Liang Feng Zhang[1]

[1] *School of Information Science and Technology, ShanghaiTech University, Shanghai, China*

[2] *Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai, China*

Keywords: Outsourcing Computation, Multi-client Verifiable Computation, Homomorphic Proxy Re-authenticator.

Abstract: In TCC 2013, Choi, Katz, Kumaresan, and Cid introduced a multi-client verifiable computation (MVC) model for outsourcing computations to cloud. MVC allows multiple non-communicating clients to outsource the computation of a function f over a series of joint inputs to a powerful but untrusted cloud server, ensuring that the input of each client will be secret from all the other entities and rejects any incorrect results from the server. They also proposed a construction of MVC, which heavily depends on fully homomorphic encryption (FHE) and garbled circuits (GCs), thus lacks practical relevance. In this paper, we propose a general transformation from the homomorphic proxy re-authenticator (HPRA) of Derler, Ramacher and Slamanig (FC 2017) to MVC. Our MVC schemes will be significantly more efficient, as long as the underlying HPRA is free of FHE and GCs. By applying the transformation to an HPRA scheme of Derler, Ramacher and Slamanig, we obtained an MVC scheme for computing the linear combinations of vectors. Our implementation shows that the new MVC scheme is significantly more efficient, both in terms of client computation and server computation. To our best knowledge, this is the first implementable MVC scheme to date.

1 INTRODUCTION

The past years have witnessed an increasing amount of attention spent on the problem of securely outsourcing computations, due to the popularity of cloud computing and the proliferation of mobile devices. Outsourcing computation allows clients (e.g. mobile devices or sensors) to offload the computation of a function to powerful cloud servers. However, the servers are not always trusted and various security concerns arise. For example, the servers may have strong financial incentives to run quick but incorrect computations to minimize the consumption of their computing resources; the servers may also abuse the client's data (e.g., function, input and output), as long as the data is not encrypted. How to guarantee the correctness of the server's results and how to protect the clients' data have been two fundamental security problems in the field of outsourcing computations.

In the single-client scenario, (Gennaro et al., 2010) introduced a model of non-interactive verifiable computation (VC) for ensuring the correctness of the server's results. The VC scheme of (Gennaro et al., 2010) allows a computationally weak client to outsource the computation of a function f on many inputs $x_1, x_2, \cdots, x_n$ to a server, but only results in impractical constructions due to their dependance on expensive cryptographic primitives such as FHE and GCs. (Benabbas et al., 2011; Parno et al., 2013; Ben-Sasson et al., 2013; Ben-Sasson et al., 2014; Braun et al., 2013; Cormode et al., 2012; Setty et al., 2013; Wahby et al., 2015; Papamanthou et al., 2013) provided significantly more efficient constructions for restricted classes of functions.

The model of (Gennaro et al., 2010) has been extended for outsourcing computation problems in many different scenarios. One of them allows multiple clients to outsource computations on their joint inputs to a cloud server. Such an extension is meaningful. For example, due to the lack of infrastructure (e.g., very limited cell coverage), n sensor nodes may be unable to directly connect to each other but are able to communicate with a central server. The sensors are likely to outsource the computation of a function f over a series of their joint data. The server is responsible to collect the data and then reports the results, based on which the sensors determine the data should be further observed.

(Choi et al., 2013) extended the single-client VC model of (Gennaro et al., 2010) to a multi-client non-

Xu, S. and Zhang, L.
A Homomorphic Proxy Re-authenticators based Efficient Multi-client Non-interactive Verifiable Computation Scheme.
DOI: 10.5220/0009158101950206
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 195-206
ISBN: 978-989-758-399-5; ISSN: 2184-4356
Copyright © 2022 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

interactive verifiable computation (MVC) model. An n-client MVC scheme allows n non-communicating clients to outsource the computation of a function f over a series of joint inputs $(x_1^{(1)}, \cdots, x_n^{(1)}), \cdots$ to a powerful cloud server, ensuring that the input of each client will be secret from all the other entities and the clients will reject any incorrect results from the server. (Choi et al., 2013) also proposed a construction of MVC for outsourcing the computation of any boolean circuit f. However, their scheme still heavily depends on FHE and GCs and thus is impractical.

To move toward more practical MVC schemes, a very intuitive idea would be replacing these expensive cryptographic primitives(i.e., FHE, GCs) with more practical algorithms. Unfortunately, if we are limited to the outsourcing of functions as generic as any boolean circuits and want to ensure the confidentiality of the client's data, then the tools such as FHE are somehow indispensable. In this paper, we mainly focus on the construction of efficient MVC schemes for outsourcing specific computations, and in particular, computing the linear combination of vectors over a finite field.

Our Contribution. Our first contribution is a general transformation from the homomorphic proxy re-authenticator (HPRA) of (Derler et al., 2017), a tool that adds security and verifiability guarantees to multi-user data aggregation scenarios, to MVC. We modified definition of some MVC algorithms, and ensure that these changes affect neither the functionality nor the security of MVC. Our transformation results in MVC schemes with the following properties:

- No malicious server is able to convince the honest clients to accept an incorrect output.
- Each client's input is hidden from the server and other clients; the result of the computation is hidden from the server.
- The involved parties do not need to communicate with each other or share a secret key. Instead, they use independent secret keys to both encrypt and authenticate their respective inputs.
- Comparing with the HPRA, the scheme introduce no extra computations; it is free of FHE/GCs and thus significantly more efficient, as long as the underlying HPRA is free of FHE/GCs.
- Comparing with the HPRA, the security of the scheme requires no extra assumptions.

Our second contribution is the implementation of a specific MVC scheme, which can be obtained by applying the general transformation to a specific HPRA scheme of (Derler et al., 2017). Our MVC allows n clients, each has a vector of dimension ℓ, to jointly compute a linear combination of their vectors. To our best knowledge, this is the first implementable MVC scheme to date. If we treat the multiplication of two matrices as a set of linear combination computations, then our scheme is applicable to outsource matrix multiplications. For the purpose of comparison, we consider the multiplication of two 100×100 matrices. (Parno et al., 2013) established a model for evaluating the efficiency of the FHE+GCs based VC schemes such as (Gennaro et al., 2010; Choi et al., 2013). The analysis of (Parno et al., 2013) shows that, in order to multiply two 100×100 matrices, the MVC scheme of (Choi et al., 2013) would take $\geq 10^{11}$ seconds at the client. The experiments show that the clients and the server in our MVC scheme only need $\approx 10^3$ seconds. Our MVC is significantly faster at both the clients and the server.

Our Techniques. The HPRA scheme is a protocol between three types of parties: a set of signers, an aggregator and a receiver. It allows the signers to authenticate their data under their own keys and allows an aggregator to transform these signatures to a MAC under the receiver's key. Most importantly, the aggregator can evaluate f on the inputs and produce an aggregate authenticated message vector corresponding to the function's output, which allows the receiver to perform necessary verifications. A main difference between MVC and HPRA is that: in HPRA, there is a receiver who obtains the result of the computation and performs the verification; while in MVC, the first client is responsible for verification. In order to construct MVC from HPRA, a very natural idea is to establish a mapping from the parties in HPRA to those in MVC. We merge the receiver with the first signer in HPRA and regard them as the first client in MVC; let the remaining signers in HPRA play the roles of all other clients in MVC; let the aggregator in HPRA play the role of a cloud server in MVC. In Section 3, we show that given an aggregator unforgeable, input private and output private HPRA scheme, our transformation yields an MVC scheme that satisfies all the properties of correctness, soundness, privacy against the first client and privacy against the server.

Related Work. (Gennaro et al., 2010) introduced a *single-client* non-interactive verifiable computation model and constructed a scheme that heavily depends on FHE and GCs. (Benabbas et al., 2011) constructed more efficient verifiable computation schemes for restricted classes of functions. Following this line, the up-to-date implementations of efficient (single-client) systems (Parno et al., 2013; Ben-Sasson et al., 2013; Ben-Sasson et al., 2014; Braun et al., 2013; Cormode et al., 2012; Setty et al., 2013; Wahby et al., 2015; Papamanthou et al., 2013) for outsourcing computations shows that in this area we are on the verge of

achieving practical efficiency.

(Choi et al., 2013) extended the single-client VC model of (Gennaro et al., 2010) to a setting where the inputs are provided by multiple clients. They provided an FHE-based construction for outsourcing any boolean circuits, which did not settle the efficiency problem, either. Moreover, the soundness may be broken when the server is allowed to send ill-formed responses to the first client and then see if that client rejects. This problem was addressed in (Gordon et al., 2015) with a scheme having stronger security guarantees against a malicious server or an arbitrary set of malicious colluding clients. However, their scheme also requires FHE and GCs. In particular, their online phase efficiency was sacrificed in order to achieve the stronger security guarantees.

(Goldwasser et al., 2014) gave a construction of multi-input functional encryption (MIFE) based on indistinguishability obfuscation (iO). Their scheme allowed multiple clients to delegate computations to an untrusted server and could achieve better efficiency. However, MIFE inherently required the existence of iO, which is a stronger assumption. (Fiore et al., 2016) constructed a multi-key homomorphic authenticator (multi-key HA), which allowed multiple clients to store data on a cloud server and delegate a computation on the data to the server. But the multi-key HAs never guaranteed the privacy of the clients' data. Based on multi-key HAs, (Derler et al., 2017) introduced the notion of homomorphic proxy re-authenticators (HPRA), which allowed distinct clients to authenticate their data under their own keys and verifies the correctness of the computation result under a re-authenticator key. Also based on multi-key HAs, (Schabhüser et al., 2019) constructed a multi-key support outsourced computation, which was claimed to be context-hiding. However, they did not provide a proper encryption scheme hence cannot effectively protect input privacy.

Organization. Section 2 highlights the preliminaries including MVC and HPRA. Section 3 presents our HPRA to MVC transformation. In section 4, we present a implementation of a concrete instantiation and discuss the experimental results. Finally, section 5 concludes the work.

2 PRELIMINARIES

In this section we recall the models and definitions of multi-client verifiable computation (MVC) and homomorphic proxy re-authenticator (HPRA).

2.1 MVC

The notion of multi-client non-interactive verifiable computation was introduced in (Choi et al., 2013). In an n-party MVC there are n clients $P_1, P_2, \ldots, P_n$ who wish to outsource the computations of f over a series of joint inputs to a server multiple times. For every $i \geq 1$, in the i-th evaluation the clients $P_1, P_2, \ldots, P_n$'s inputs are denoted as $x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}$ respectively. P_1 is designated to learn $y^{(i)} = f(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$. MVC guarantees both the secrecy of each client's input and the correct reconstruction of all function values.

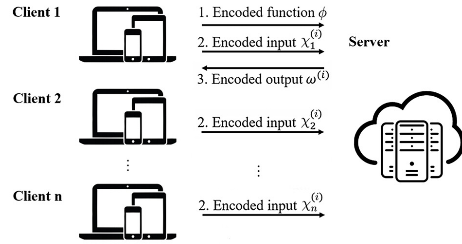

Figure 1: Multi-client Verifiable Computation.

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

- $(pk_1, sk_1) \leftarrow \mathsf{KeyGen}_1(\mathsf{pp})$. *The client* P_1 *will run this algorithm on a set* pp *of public parameters to obtain a* public key pk_1 *and a* private key sk_1.
- $(pk_j, sk_j) \leftarrow \mathsf{KeyGen}_j(\mathsf{pp}, pk_1)$. *For* $j = 2, \cdots, n$, *the client* P_j *will run this algorithm to obtain a* public key pk_j *and a* private key sk_j.
- $(\phi, \xi) \leftarrow \mathsf{EnFunc}(\overrightarrow{pk}, sk_1, f)$. *The client* P_1 *will run this algorithm with* $\overrightarrow{pk} = (pk_1, pk_2, \ldots, pk_n)$ *and* sk_1 *to encode any* $f \in \mathcal{F}$ *as an* encoded function ϕ, *which will be sent to the server, and generate a* decoding secret ξ, *which will be kept private by the client.*
- $(\chi_1^{(i)}, \tau^{(i)}) \leftarrow \mathsf{EnInput}_1(i, \overrightarrow{pk}, sk_1, \xi, x_1^{(i)})$. *When outsourcing the i-*th *computation to the server,* P_1 *will run this algorithm to convert its input* $x_1^{(i)}$ *as an* encoded input $\chi_1^{(i)}$, *which will be sent to the server, and generate a* decoding secret $\tau^{(i)}$, *which will be kept private by the client.*
- $\chi_j^{(i)} \leftarrow \mathsf{EnInput}_j(i, \overrightarrow{pk}, sk_j, x_j^{(i)})$. *When outsourcing the i-*th *computation to the server, each client* P_j $(j \neq 1)$ *will run this algorithm to convert its input* $x_j^{(i)}$ *as an* encoded input $\chi_j^{(i)}$, *which will be sent to the server. We denote* $\overrightarrow{\chi}^{(i)} = (\chi_1^{(i)}, \cdots, \chi_n^{(i)})$.

- $\omega^{(i)} \leftarrow \mathsf{Compute}(i, \overrightarrow{pk}, \phi, \overrightarrow{\chi}^{(i)})$. *The server will run this algorithm to get an* encoded output $\omega^{(i)}$.
- $y^{(i)} \cup \{\perp\} \leftarrow \mathsf{Verify}(i, \xi, \tau^{(i)}, \omega^{(i)})$. *The client* P_1 *will run this algorithm to output either a value* $y^{(i)} = f(x_1^{(i)}, \cdots x_n^{(i)})$*, or a symbol* $\perp$ *indicating that the server attempted to cheat.*

We modify the definition of some MVC algorithms, and ensure that: (1) these changes do not affect the functionality of MVC, that is, the modified MVC can still solve the same question; (2) these changes do not affect the security of MVC, as each participant does not get any additional information.

- We replace all security parameter 1^κ in inputs with a public parameter pp which contains both information of 1^κ and function f, as f is public.
- We run client P_1's algorithm KeyGen_1 first and let pk_1 be an input of client P_j's algorithm KeyGen_j , for $j = 2, \ldots, n$. This change causes other clients to wait for client P_1 finish KeyGen_1, but does not affect security. As (Choi et al., 2013) has assumed that there was a public-key infrastructure (PKI), such that all clients had public keys known to each other in MVC scheme.
- Our EnFunc needs sk_1 as input, which is also a reasonable change as EnFunc is run by client P_1.

Required by (Choi et al., 2013; Gordon et al., 2015), an MVC scheme should be *correct*, *sound* and *input private*. An MVC scheme is correct if all algorithms of the scheme are faithfully executed, then the verification algorithm will always produce the correct function output.

An MVC scheme is sound if no malicious server can convince the honest clients to accept an incorrect output, even if it is given access to **Oracle** $\mathcal{IN}$, which generates multiple input encodings as follows.

Oracle $\mathcal{IN}(x_1, \cdots, x_n)$

- $i := i + 1$;
- record $(x_1^{(i)}, \ldots, x_n^{(i)}) := (x_1, \ldots, x_n)$;
- $(\chi_1^{(i)}, \tau^{(i)}) \leftarrow \mathsf{EnInput}_1(i, \overrightarrow{pk}, sk_1, \xi, \overrightarrow{x_1^{(i)}})$;
- for $j = 2, \ldots, n : \chi_j^{(i)} \leftarrow \mathsf{EnInput}_j(i, \overrightarrow{pk}, sk_j, x_j^{(i)})$;
- output $(\chi_1^{(i)}, \ldots, \chi_n^{(i)})$.

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

An MVC scheme is *private* if every client's input is kept private from the server as well as the other clients. In MVC scheme, clients other than the first client clearly do not learn anything about each others' inputs. Therefore, the input privacy of MVC consists of two parts: *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 input vectors* $\vec{x}_0 = (x_1, x_2, \cdots, x_n), \vec{x}_1 = (x_1, x_2', \cdots, x_n')$ *with* $f(\vec{x}_0) = f(\vec{x}_1)$*, the view of* P_1 *when running an execution of* Π *with all clients holding* $\vec{x}_0$ *is indistinguishable from the view of* P_1 *when running an execution with all clients holding* $\vec{x}_1$.

The definition of privacy against the server requires that the encoded inputs from two distinct inputs should be indistinguishable to the server, even when malicious server is given access to the encoded inputs generation oracle **Oracle** $\mathcal{IN}$.

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

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

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

2.2 HPRA

The notion of homomorphic proxy re-authenticator was introduced in (Derler et al., 2017). An HPRA scheme involves three types of parties: a set of *signers*, an *aggregator* and a *receiver*. It allows n signers to authenticate data items $\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n$ with signatures under their own keys, and allows the aggregator

to transform these signatures to a signature under the receiver's key. In addition, the aggregator can evaluate a function f on the inputs so that the resulting signature corresponds to the evaluation of f, and produce an aggregate authenticated message vector Λ.

Definition 5. (HPRA) *A* homomorphic proxy re-authenticator *scheme* $\Sigma = (\mathsf{Gen}, \mathsf{SGen}, \mathsf{VGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SRGen}, \mathsf{VRGen}, \mathsf{Agg}, \mathsf{AVerify})$ *consists of nine polynomial-time algorithms as below and shown in Figure 2.*

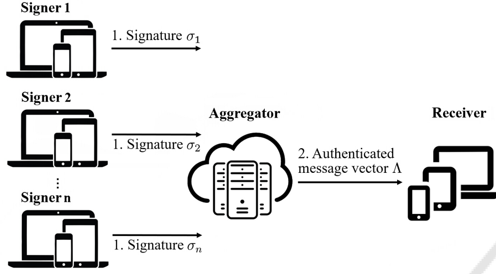

Figure 2: Homomorphic Proxy Re-Authenticator.

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa, \ell)$*: Given a security parameter* 1^κ *and a constant* ℓ*, outputs a set of* public parameters pp*, which explicitly defines a* message space $\mathcal{M}^\ell$*, a* function family $\mathcal{F} = \{f | f : (\mathcal{M}^\ell)^n \to \mathcal{M}^\ell\}$*, and a* tag space.
- $(\mathsf{id}, \mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SGen}(\mathsf{pp})$*: Each signer* P_i *will run this algorithm on public parameters* pp *to output a* signer key *consist of an identifier* id*, a* private key sk *and a* public key pk.
- $(\mathsf{mk}, \mathsf{aux}) \leftarrow \mathsf{VGen}(\mathsf{pp})$*: The receiver will run this algorithm on public parameters* pp *to output a* MAC key mk *and* auxiliary information aux.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \vec{m}, \mu)$*: Each signer will run 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* $\overrightarrow{\sigma} = (\sigma_1 \ldots \sigma_n)$.
- $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \vec{m}, \mu, \sigma)$*: Any entity who knows* $\vec{m}$ *can run this algorithm to verify the validation of a signature* σ*, and outputs a bit* b.
- $\mathsf{rk}_i \leftarrow \mathsf{SRGen}(\mathsf{sk}_i, \mathsf{aux})$*: Each signer will run this algorithm to generate a* re-encryption key rk_i.
- $\mathsf{ak}_i \leftarrow \mathsf{VRGen}(\mathsf{pk}_i, \mathsf{mk}, \mathsf{rk}_i)$*: The receiver will run this algorithm to generate an* aggregation key ak_i*, which will be sent to the aggregator.*
- $\Lambda \leftarrow \mathsf{Agg}(\overrightarrow{\mathsf{ak}}, \overrightarrow{\sigma}, \mu, f)$*: The aggregator will run this algorithm to compute an* aggregate authenticated message vector Λ*. Here* $\overrightarrow{\mathsf{ak}} = (\mathsf{ak}_1, \ldots, \mathsf{ak}_n)$.
- $(\vec{m}, \mu)/(\perp, \perp) \leftarrow \mathsf{AVerify}(\mathsf{mk}, \Lambda, \mathsf{ID}, f)$*: The receiver will run this algorithm to output a pair* $(\vec{m}, \mu)$*, otherwise output* $(\perp, \perp)$ *indicating that the server attempts to cheat.*

Required by (Derler et al., 2017), an HPRA scheme should be *correct*, *input private*, *signer unforgeable*, and *aggregator unforgeable*. *Correctness* of an HPRA requires that if all algorithms of the scheme are executed faithfully, then the Λ will always verify successfully and decode to the correct output $\vec{m} = f(\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n)$.

The *input Privacy* of HPRA captures the requirement that an aggregate authenticated message vector Λ should leak no more information about the signers' data $\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n$ beyond what can be inferred from $\vec{m} = f(\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n)$ and the description of f.

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

$$\{\mathsf{Agg}(\overrightarrow{\mathsf{ak}}, \mathsf{Sign}(\mathsf{sk}_i, \vec{m}_{i1}, \mu)_{i \in [n]}, \mu, f)\},$$
$$\{\mathsf{Agg}(\overrightarrow{\mathsf{ak}}, \mathsf{Sign}(\mathsf{sk}_i, \vec{m}_{i2}, \mu)_{i \in [n]}, \mu, f)\}.$$

The *signer unforgeability* of an HPRA scheme requires that, as long as the aggregator 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 unforgeability, where the aggregator is dishonest and the signers are honest.

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

- $\mathsf{SG}(i)$: Works as SGen, sets $\mathtt{S}[i] \leftarrow (\mathsf{id}, \mathsf{sk}, \mathsf{pk})$, returns $(\mathsf{id}, \mathsf{pk})$.
- $\mathsf{Skey}(i)$: Returns $\mathtt{S}[i]$.
- $\mathsf{Sig}(\{1, \ldots, n\}, (\vec{m}_i)_{i \in [n]})$: Works as Sign, sets $\mathtt{SIG}[\mu] \leftarrow \mathtt{SIG}[\mu] \cup \{\vec{m}_i, \mathtt{S}[i]\}$ for $i = 1, \ldots, n$, returns $\overrightarrow{\sigma} = (\sigma_1 \ldots \sigma_n)$ and μ.
- $\mathsf{SR}(i)$: Works as SRGen, returns $\mathtt{RK}[i] = \mathsf{rk}_i$.
- $\mathsf{VR}(i)$: Works as VRGen but without returning anything, sets $\mathtt{AK}[i] = \mathsf{ak}_i$.
- $\mathsf{VRKey}(i)$: Returns $\mathtt{AK}[i]$

- $\mathsf{A}(\overrightarrow{\sigma}, \{1,\ldots,n\}, \mu, f)$: Works as Agg, returns Λ

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

An HPRA scheme may provide *output privacy*, which models the situation that the aggregator learns neither the inputs nor the function's output. The formal definition of output privacy requires an oracle RoS, which is defined as below:

- $\mathsf{RoS}(i, (\vec{m}_i)_{i\in[n]}, b)$*: If* $\mathsf{S}[i] = \perp$*, returns* $\perp$*. Otherwise samples* μ *uniformly at random and if* $b = 0$ *computes* $(\sigma_i \leftarrow \mathsf{Sign}(\mathsf{S}[i][2], \vec{m}_i, \mu))_{i\in[n]}$*. Else randomly chooses* $(\vec{r}_i)_{i\in[n]} \leftarrow (\mathcal{M}^\ell)^n$ *and computes* $(\sigma_i \leftarrow \mathsf{Sign}(\mathsf{S}[i][2], \vec{r}_i, \mu))_{i\in[n]}$ *and returns* $\overrightarrow{\sigma} = (\sigma_1 \ldots \sigma_n)$.

Definition 8. **(Output Privacy)** *For the* HPRA *scheme* Σ*, consider an experiment with* $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{outpriv}}(\Sigma, \kappa, n, \ell)$ *respect to a PPT adversaries* $\mathcal{A}$*: Public parameter* pp *is generated by running* $\mathsf{Gen}(1^\kappa, \ell)$*, MAC key and auxiliary information* $(\mathsf{mk}, \mathsf{aux})$ *are generated by running* $\mathsf{VGen}(\mathsf{pp})$*.* $\mathcal{A}$ *is given input* pp *and access to oracle* O*. A random bit* $b \leftarrow \{0,1\}$ *is chosen by challenger, and a challenge ciphertext* $\overrightarrow{\sigma}$ *is computed and given to* $\mathcal{A}$*.* $\mathcal{A}$ *continues to have oracle access to* O*, and outputs a guess* b' *of* b*. Where the oracle* $O := \{\mathsf{SG}, \mathsf{SKey}, \mathsf{RoS}(b), \mathsf{SR}, \mathsf{VR}, \mathsf{VRKey}\}$*. An HPRA for a family of function classes* $\mathcal{F}$*, is* output private*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\cdot)$ *such that:* $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{outpriv}}(\Sigma, \kappa, n, \ell) = 1] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$.

3 OUR TRANSFORMATION

In this section, we show the general transformation from HPRA to MVC. As described in Section 2, an n-client MVC scheme involves two types of parties: n clients and a server. An n-signer HPRA schemes involves three types of parties: n signers, an aggregator and a receiver. Simple comparisons give us the following observations: (1) Both the clients in MVC and the signers in HPRA are responsible to provide inputs; (2) Both the server in MVC and the aggregator in HPRA are responsible to compute an encoded output and a proof; (3) Both the first client in MVC and the receiver in HPRA are responsible to perform the verification and the decryption.

Based on the above observations, a very natural idea of transforming HPRA to MVC is via the follow mapping between the participants of two schemes: (1) let the aggregator in HPRA play the role of the server in MVC; (2) let the first signer and the receiver in HPRA play the role of the first client in MVC; (3) for every $j = 2, 3, \ldots, n$, let the j-th signer in HPRA play the role of the j-th client in MVC. This natural idea gives a scheme in the standard MVC model, which can be depicted with Figure 3.

It remains to understand why the above transformation gives an MVC scheme with the desired properties of correctness, soundness, privacy against the first client, and privacy against the server, provided that the underlying HPRA scheme satisfies the properties of correctness, T-Unforgeability, input privacy and output privacy:

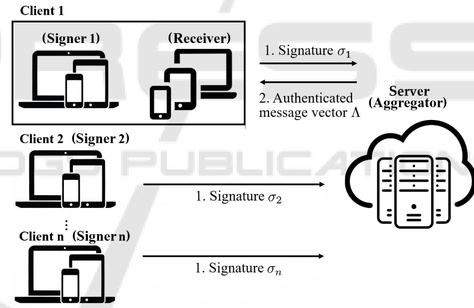

Figure 3: Idea of Transformation.

- The aggregator unforgeability (Definition 7) of HPRA requires that no dishonest aggregator can persuade the receiver to both accept and output a wrong function value, which naturally gives the soundness (Definition 2) of MVC.
- The output privacy (Definition 8) of HPRA requires that the aggregator should not be able to learn any information about the signers' inputs or the function's output. This property implies the property of privacy against the server (Definition 4) in MVC.
- The input privacy (Definition 6) of HPRA gives the privacy against the first client (Definition 3) in MVC. Our transformation would give an MVC scheme where the server learns the first client's information, including its private keys and inputs. However, these additional information will not af-

fect the security. In Definition 3, the first element of two challenge input vectors $\vec{x}_0$ and $\vec{x}_1$ must be equal, which prevents a malicious server from using the first client's information to gain any advantage in the security experiment.

Let $\Sigma = (\mathsf{Gen}, \mathsf{SGen}, \mathsf{VGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SRGen}, \mathsf{VRGen}, \mathsf{Agg}, \mathsf{AVerify})$ be an HPRA scheme for a function family $\mathcal{F}$. Our general transformation from Σ to $\Pi = (\{\mathsf{KeyGen}_j\}_{j=1}^n, \mathsf{EnFunc}, \{\mathsf{EnInput}_j\}_{j=1}^n, \{\mathsf{EnInput}_j\}_{j=1}^n, \mathsf{Compute}, \mathsf{Verify})$, an MVC scheme for $\mathcal{F}$, will be detailed in the remaining part of this section. Let $\mathsf{pp} \leftarrow \Sigma.\mathsf{Gen}\ (1^\kappa, \ell)$ be a set of public parameters. In the proposed MVC scheme we consider the computation of a function f on the input $(\vec{x}_1^{(i)}, \ldots, \vec{x}_n^{(i)})$, where each client P_j's input $\vec{x}_j^{(i)} \in M^\ell$ is a vector over a finite field. Our MVC scheme works as follows:

- $(pk_1, sk_1) \leftarrow \mathsf{KeyGen}_1(\mathsf{pp})$. *On input public parameter* pp*, the client* P_1 *runs HPRA's signer's key generation algorithm* $\Sigma.\mathsf{SGen}(\mathsf{pp})$ *and obtains an identifier* id_1 *and a public/private key pair* $(\mathsf{sk}_1, \mathsf{pk}_1)$*. Client* P_1 *runs receiver's key generation algorithm* $\Sigma.\mathsf{VGen}(\mathsf{pp})$ *obtains a MAC key* mk *and auxiliary information* aux*. On private key* sk_1 *and auxiliary information* aux*, client* P_1 *runs signer's re-encryption key generation algorithm* $\Sigma.\mathsf{SRGen}(\mathsf{sk}_1, \mathsf{aux})$*, and obtains re-encryption key* rk_1*. The client* P_1 *sets* $pk_1 = (\mathsf{id}_1, \mathsf{pk}_1, \mathsf{aux}, \mathsf{rk}_1)$*,* $sk_1 = (\mathsf{sk}_1, \mathsf{mk})$*.*
- $(pk_j, sk_j) \leftarrow \mathsf{KeyGen}_j(\mathsf{pp}, pk_1)$. *On inputs public parameter* pp *and client* P_1*'s public key* pk_1*, for* $j = 2 \ldots n$ *each client* P_j *runs HPRA's signer's key generation algorithm* $\Sigma.\mathsf{SGen}(\mathsf{pp})$ *and obtains an identifier* id_j *and a public/private key pair* $(\mathsf{sk}_j, \mathsf{pk}_j)$*. On inputs private key* sk_j *and auxiliary information* aux*, client* P_j *runs signer's rekey generation algorithm* $\Sigma.\mathsf{SRGen}(\mathsf{sk}_j, \mathsf{aux})$*, and obtains re-encryption key* rk_j*. Client* P_j *individually sets* $pk_j = (\mathsf{id}_j, \mathsf{pk}_j, \mathsf{rk}_j)$*,* $sk_j = \mathsf{sk}_j$*.*
- $(\phi, \xi) \leftarrow \mathsf{EnFunc}(\overrightarrow{pk}, sk_1, f)$*. For* $j = 1 \ldots n$*, takes a signer public key* pk_j *and client* P_1*'s private key* sk_1*, client* P_1 *runs* $\Sigma.\mathsf{VRGen}(\mathsf{pk}_j, \mathsf{mk}, \mathsf{rk}_j)$ *and obtains an aggregation key* ak_j*. The client* P_1 *sets the encoded function* $\phi = (f, \overrightarrow{\mathsf{ak}})$ *and decoding secret* $\xi = \varnothing$*.*
- $(\chi_1^{(i)}, \tau^{(i)}) \leftarrow \mathsf{EnInput}_1(i, \overrightarrow{pk}, sk_1, \xi, \vec{x}_1^{(i)})$*. When outsourcing the i-th computation to the server, takes a time period i, the public keys* $\overrightarrow{pk}$*, client* P_1*'s private key* sk_1*, an input message vector* $\vec{x}_1^{(i)}$*, the decoding secret* ξ*, the first client* P_1 *gets a tag* $\mu \in \mathbb{G}$*, runs the sign algorithm* $\Sigma.\mathsf{Sign}(\mathsf{sk}_1, \vec{x}_1^{(i)}, \mu)$*, gets a signature* σ_1*, set* $\chi_1^{(i)} = \sigma_1$*,* $\tau^{(i)} = \mathsf{mk}$*.*
- $\chi_j^{(i)} \leftarrow \mathsf{EnInput}_j(i, \overrightarrow{pk}, sk_j, \vec{x}_j^{(i)})$*. When outsourcing the i-th computation to the server, takes a time period i, the public keys* $\overrightarrow{pk}$*, client* P_j*'s signer secret key* sk_j *and input message vector* $\vec{x}_j^{(i)}$*, each client* P_j *(with* $j \neq 1$*) gets a tag* $\mu \in \mathbb{G}$*, then he runs the algorithm* $\Sigma.\mathsf{Sign}(\mathsf{sk}_j, \vec{x}_j^{(i)}, \mu)$*, gets signature* σ_j*, sets* $\chi_j^{(i)} = \sigma_j$*.*
- $\omega^{(i)} \leftarrow \mathsf{Compute}(i, \overrightarrow{pk}, \phi, \overrightarrow{\chi}^{(i)})$*. Given the public keys* $\overrightarrow{pk}$ *, the encoded function* ϕ*, and the encoded inputs* $\vec{\chi}^{(i)}$*, the server runs the aggregate algorithm* $\Sigma.\mathsf{Agg}(\overrightarrow{\mathsf{ak}}, \overrightarrow{\chi}^{(i)}, \mu, \phi)$*, gets aggregate authenticated message vector* Λ *and set* $\omega^{(i)} = \Lambda$*.*
- $y^{(i)} \cup \{\perp\} \leftarrow \mathsf{Verify}(i, \overrightarrow{pk}, \xi, \tau^{(i)}, \omega^{(i)})$*. Take the public keys* $\overrightarrow{pk}$*, the decoding secrets* $(\xi, \tau^{(i)})$*, and an encoded output* $\omega^{(i)}$*, the first client* P_1 *runs receiver's verification algotithm* $\Sigma.\mathsf{AVerify}(\mathsf{mk}, \Lambda, \mathsf{ID}, f)$ *which outputs a message vector and a tag* $(\vec{m}, \mu)$ *on success and* $(\perp, \perp)$ *otherwise. Sets* $y^{(i)} = \vec{m}$ *on success and returns* $\perp$ *otherwise.*

The scheme Π should satisfy the properties of *correctness, soundness* and *privacy*. While the correctness property is easy to verify, proofs for the following theorems can be found in Appendix.

Theorem 1. *If* Σ *is a* T*-unforgeable HPRA scheme, here* T = "Aggregator"*, then* Π *described above is a sound MVC scheme.*

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

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

4 IMPLEMENTATION

To evaluate the efficiency of our transformation, we apply our transformation on a concrete instantiation of HPRA, scheme 3 in (Derler et al., 2017), and conduct several experiments. The experiments is tested on personal computer with a Intel Core i7-4790K CPU running at 3.60GHz, 8GB of RAM running on Windows 10. And we use VMware Workstation Pro 12.0 to help us setup a build environment on ubuntu-16.04.1. The entire implementation is completely done in C programming language. We use the PBC (Pairing-Based Cryptography) library [1] to help us

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

generating suitable bilinear-map groups. More specifically, we use the type A curve defined in PBC library: $y^2 = x^3 + x$ over F_q for prime $q = 3 \bmod 4$ with $|q| = 512$.

We omit simulating the communication between the clients and server, as our work does not focus on the communication process and this simplification will not affect our analysis on computational complexity.

4.1 Client Computation

There are three variables that have a major impact on runtime: the length ℓ of the input vector, the maximum size of input vector element, denoted as $x_{\max}$ (any element of client P_j's input vector $\vec{x}_j^{(i)}$ is smaller than $2^{x_{\max}}$) and the total number n of the clients. We will first give a theoretical analysis on how these variables affect the clients' and server's runtime and then verify the correctness of the analysis through some carefully designed experiments described as follows. We define some symbols for simplification:

$Mul_{\mathbb{G}}$:	Multiplication operation time in $\mathbb{G}$.
$Exp_{\mathbb{G}}$:	Exponentiation operation time in $\mathbb{G}$.
$Mul_{\mathbb{G}_T}$:	Multiplication operation time in $\mathbb{G}_T$.
$Exp_{\mathbb{G}_T}$:	Exponentiation operation time in $\mathbb{G}_T$.
$Pair_{\mathbb{G}}$:	Pairing operation time in $\mathbb{G}$.
Log:	Time to solve the discrete log problem.

Pre-processing. The pre-processing is a process consist of KeyGen (including both KeyGen_1 and KeyGen_j) and EncFunc. The theoretical analysis of pre-processing runtime is shown in Table 1. Here we set $x_{\max}$ to 20 bits, the experimental results shown in the Figure 4 depicts that:

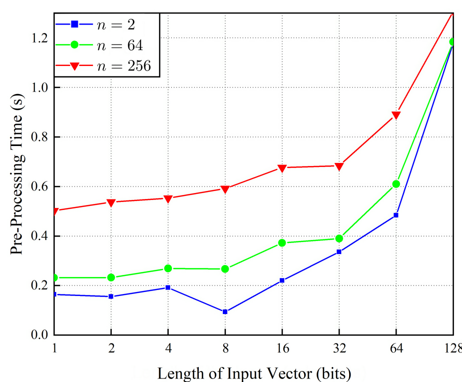

Figure 4: Client's Pre-processing Time.

- When the total number n of the clients is fixed, pre-processing time is proportional to the length ℓ of the input vector.
- When the length ℓ of the input vector is fixed, pre-processing time is proportional to n.
- As ℓ and n increase, ℓ has a greater impact on runtime than n.

Input Encoding. We omit the experiment on the clients' encoding process which only includes EnInput, as it is a efficient algorithm whose runtime only relates to length ℓ of the input vector as Table 1 revealed.

Verification Process. The verification process including only Verify algorithm which acctually consist of two parts: decryption part and verification part. We used the Pollard's kangaroo algorithm (Pollard, 1978) which is the fastest known way to break discrete log for a general elliptic curve with complexity $O(\sqrt{x_{\max}})$, to help us solving the discrete log problem in decryption part. Thus, the complexity of the decryption part is related to the maximum size $x_{\max}$ of input vector element and the length ℓ of the input vector as Table 1 revealed. The runtime of the verification part becomes negligible compared to the time required for decryption as $x_{\max}$ increase. Here we fix n to 2 for it has very little impact on the runtime, the experimental results shown in the Figure 5 with log 2 type Y axis indicates that:

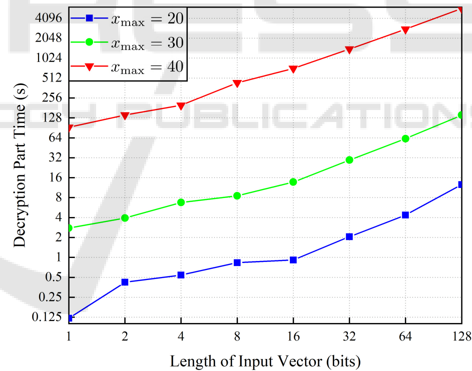

Figure 5: Client's Decryption Time.

- When the maximum size $x_{\max}$ of input vector element is fixed, the runtime verification process is proportional to the length ℓ of the input vector.
- When the length ℓ is fixed, the runtime of verification process is proportional to $x_{\max}$.
- As $x_{\max}$ and ℓ increase, $x_{\max}$ has a greater impact on runtime than ℓ.

4.2 Server Computation

As shown in Table 1, the server's runtime is related to the length ℓ of the input vector and the total number n of the clients. Here we set $x_{\max}$ to 20 bits for sim-

Table 1: Computational Complexity of Our MVC.

	$Mul_{\mathbb{G}}$	$Exp_{\mathbb{G}}$	$Mul_{\mathbb{G}_T}$	$Exp_{\mathbb{G}_T}$	$Pair_{\mathbb{G}}$	Log
Pre-processing	0	$2\ell+2n+4$	0	$\ell+1$	0	0
Input Encoding	$\ell+1$	$\ell+3$	ℓ	2ℓ	0	0
Decryption Part	0	0	$\ell+1$	$\ell+1$	0	$\ell+1$
Verification Part	$\ell-1$	$\ell+n$	$n+1$	3	$\ell+n$	0
Server Computation	0	n	$(2\ell+3)(n-1)+1$	$(2\ell+2)n$	$\ell+n+1$	0

plicity, the experimental results shown in the Figure 6 below further demonstrate our theory:

- When the total number n of the clients is fixed, server's runtime is proportional to the length ℓ of the input vector.
- When the length ℓ is fixed, server's runtime is proportional to n.

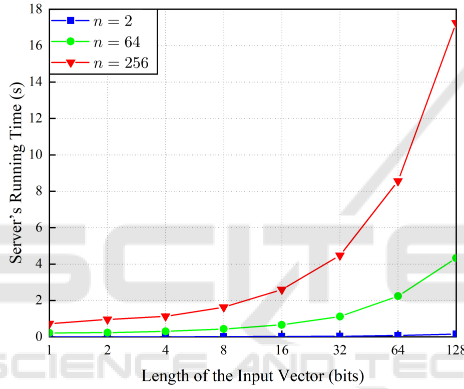

Figure 6: Server's runtime.

maximum size $x_{\max}$ stays rather small, server's runtime will exceed client's runtime.

The experimental results shown in the Figure 7 below further demonstrate our theory. Here we fixed ℓ to 128, and set $x_{\max}=20$ bits which is rather small. As n grows, the server's runtime will exceed the clients' total runtime, and the gap between these two will become more and more significant.

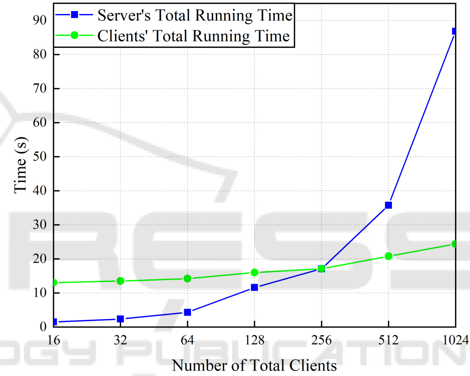

Figure 7: Total runtime.

4.3 Analysis and Comparisons

Efficiency Analysis. In some cases, the client computation time is even higher than server computation time, and we are trying to avoid it. Below we analyze under what circumstances this concrete instantiation is an efficient solution from both theory and experiments. We assume that the time required for all group operations is constant. And $\sqrt{x_{\max}+\log_2 n-1}$ represents the expected value of the sum of n clients' input vector elements with maximum size $x_{\max}$.
Theoretically speaking, the total runtime of client P_1:

$$T_{\text{client}} = 12\ell+5n+12+(\ell+1)\sqrt{x_{\max}+\log_2 n-1}.$$

In general, we hope that any client takes will less time than the server needs in the process, otherwise, this MVC scheme will not be efficient, that is:

$$T_{\text{client}} \leq 7n+(4n-1)\ell-1.$$

By observing the inequality above, when the total number n of the clients grows big enough and the

Comparisons with Previous Work. As MVC is a extend non-interactive verifiable computation scheme (Gennaro et al., 2010) where inputs are provided by multiple clients, both MVC and non-interactive verifiable computation use the expensive FHE and GCs technology in the same way, each client runs the encryption and decryption algorithm only once in one round computing. (Parno et al., 2013) established a model for evaluating the efficiency of the FHE+GCs based VC schemes such as (Gennaro et al., 2010; Choi et al., 2013). More specifically, they simulated an experiment on two matrices with parameter t, takes as input two $(t\times t)$ matrices M_1 and M_2, and outputs the $t\times t$ matrix $M_1\cdot M_2$, each component in M_1 and M_2 is at most 32 bits.

In order to fit experiment in (Parno et al., 2013), we set both the total number of the clients n and length of the input vector ℓ equal to t, the maximum size of input vector component equals to 32 bits, and t clients will jointly outsource t linear combination functions in order to get the result of matrix multiplication.

Here we plot the time necessary for the verify process, as showing in Figure 8, our verify algorithms is much more efficient than the original MVC scheme. We believe these numbers show that our protocol is applicable in a practical setting.

Unfortunately, since we used the Pollard's kangaroo algorithm (Pollard, 1978) to solve the discrete log problem in the decryption. The time we spend in the decryption process is still relatively long, it is acceptable when the input length is shorter than 50-bit.

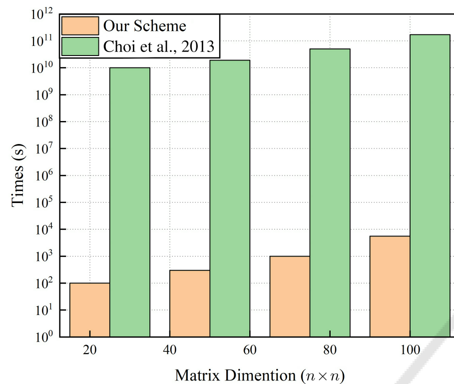

Figure 8: Comparison with (Choi et al., 2013).

5 CONCLUSION

In this paper, we provide a general transformation from HPRA to MVC. We also implemented an MVC scheme for computing the linear combinations of vectors over a finite field. To our best knowledge, this is the first implementable MVC scheme for outsourcing specific computation to date. Our implementation requires the computation of discrete logarithms. How to avoid the the expensive operations of computing discrete logarithms is an open question for further research.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China (Grant No. 61602304). The authors thank Zidong Lu for useful helping provided in experiment environment configuration and C programming.

REFERENCES

Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., and Virza, M. (2013). Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Annual Cryptology Conference*, pages 90–108. Springer.

Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. (2014). Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium*, pages 781–796.

Benabbas, S., Gennaro, R., and Vahlis, Y. (2011). Verifiable delegation of computation over large datasets. In *Annual Cryptology Conference*, pages 111–131. Springer.

Braun, B., Feldman, A. J., Ren, Z., Setty, S., Blumberg, A. J., and Walfish, M. (2013). Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 341–357. ACM.

Choi, S. G., Katz, J., Kumaresan, R., and Cid, C. (2013). Multi-client non-interactive verifiable computation. In *Theory of Cryptography Conference*, pages 499–518. Springer.

Cormode, G., Mitzenmacher, M., and Thaler, J. (2012). Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM.

Derler, D., Ramacher, S., and Slamanig, D. (2017). Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation. In *International Conference on Financial Cryptography and Data Security*, pages 124–142. Springer.

Fiore, D., Mitrokotsa, A., Nizzardo, L., and Pagnin, E. (2016). Multi-key homomorphic authenticators. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 499–530. Springer.

Gennaro, R., Gentry, C., and Parno, B. (2010). Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer.

Goldwasser, S., Gordon, S. D., Goyal, V., Jain, A., Katz, J., Liu, F.-H., Sahai, A., Shi, E., and Zhou, H.-S. (2014). Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 578–602. Springer.

Gordon, S. D., Katz, J., Liu, F.-H., Shi, E., and Zhou, H.-S. (2015). Multi-client verifiable computation with stronger security guarantees. In *Theory of Cryptography Conference*, pages 144–168. Springer.

Papamanthou, C., Shi, E., and Tamassia, R. (2013). Signatures of correct computation. In *Theory of Cryptography Conference*, pages 222–242. Springer.

Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE.

Pollard, J. M. (1978). Monte carlo methods for index computation mod p. *Mathematics of computation*, 32(143):918–924.

Schabhüser, L., Butin, D., and Buchmann, J. (2019). Context hiding multi-key linearly homomorphic authenti-

cators. In *Cryptographers Track at the RSA Conference*, pages 493–513. Springer.

Setty, S., Braun, B., Vu, V., Blumberg, A. J., Parno, B., and Walfish, M. (2013). Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84. ACM.

Wahby, R. S., Setty, S. T., Ren, Z., Blumberg, A. J., and Walfish, M. (2015). Efficient ram and control flow in verifiable outsourced computation. In *NDSS*.

APPENDIX

Proof of Theorem 1

We show that if there exists a polynomial-time adversary (PPT) $\mathcal{A}$ for which break soundness of MVC (Definition 2), that is, $\Pr[\mathbf{Exp}^{\text{sound}}_{\mathcal{A}}[\Pi, f, \kappa, n] = 1] \geq$ non-negl(κ) then we can construct a PPT adversary $\mathcal{B}$ that breaks the T-unforgeability of Σ when $\mathsf{T} =$ "Aggregator". $\mathcal{B}$ is given inputs $(1^\kappa, n, \ell)$ and an oracle $O_\mathsf{T} := \{\mathsf{SG}, \mathsf{Sig}, \mathsf{SR}, \mathsf{VR}, \mathsf{VRKey}\}$. In detail:

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

1. *The challenger runs* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa, \ell)$, $(\mathsf{mk}, \mathsf{aux}) \leftarrow \mathsf{VGen}(\mathsf{pp})$*, sends* pp *and* aux *to* $\mathcal{B}$.
2. *For* $j = 1, \ldots n$, $\mathcal{B}$ *queries the following oracles and gets* $(\mathsf{id}_j, \mathsf{pk}_j) \leftarrow \mathsf{SG}(j)$, $\mathsf{rk}_j \leftarrow \mathsf{SR}(j)$, $\mathcal{B}$ *also queries the oracle* $\mathsf{VR}(j)$ *to generate* ak_j*, and queries* $\mathsf{ak}_j \leftarrow \mathsf{VRKey}(j)$.
3. $\mathcal{B}$ *then sets* $pk_1 = (\mathsf{id}_1, \mathsf{pk}_1, \mathsf{aux}, \mathsf{rk}_1)$*, for* $j = 2 \ldots n$, $pk_j = (\mathsf{id}_j, \mathsf{pk}_j, \mathsf{rk}_j)$, $\phi = (f, (\mathsf{ak}_j)_{j \in [n]})$ *and sends them to* $\mathcal{A}$*. Here* f *is 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 input* $(\vec{x}_1, \ldots, \vec{x}_n)$, $\mathcal{B}$ *answers the queries in following way:*
   a. *Sets* $i := i + 1$.
   b. *Records* $(\vec{x}^{(i)}_1, \ldots \vec{x}^{(i)}_n) := (\vec{x}_1, \ldots, \vec{x}_n)$.
   c. *Queries the oracle and gets* $(\overrightarrow{\sigma}^{(i)}, \mu) \leftarrow \mathsf{Sig}((j)_{j \in [n]}, (\vec{x}^{(i)}_j)_{j \in [n]})$.
   d. *Sets and returns* $\overrightarrow{\chi}^{(i)} := \overrightarrow{\sigma}^{(i)}$.
6. *When* $\mathcal{A}$ *outputs* $\omega^\star$, $\mathcal{B}$ *sets* $\Lambda^\star := \omega^\star$ *and sends* $(\Lambda^\star, \mathsf{ID}^\star, f^\star)$ *to the challenger, here* $\mathsf{ID}^\star := (\mathsf{id}_1, \ldots, \mathsf{id}_n)$, $f^\star = f$.
7. *The challenger runs* $\mathsf{AVerify}(\mathsf{mk}, \Lambda^\star, \mathsf{ID}^\star, f^\star)$.
8. *If* $(\vec{m}, \mu) \neq (\perp, \perp)$ *and* $(\nexists (\vec{x}_j)_{j \in [n]} : (\forall j \in [n] : (\vec{x}_j, \mathsf{id}^*) \in \mathsf{SIG}[\mu]) \wedge f^\star(\vec{x}_1 \ldots, \vec{x}_n) = \vec{m})$ *(i.e., 1.* $f^\star(\vec{x}_1, \ldots, \vec{x}_n) = \vec{m}$ *, and at least one* $\vec{x}_j$ *have not been queried by* $\mathcal{A}$ *in the i-th querry or 2.* $f^\star(\vec{x}_1, \ldots, \vec{x}_n) \neq \vec{m}$*), outputs 1; otherwise outputs 0.*

If a PPT adversary $\mathcal{A}$ can break MVC with non-negl(κ), then $\mathcal{A}$ can find a $\omega^\star$, such that challenger gets $y^* \notin \{\perp, f(x^{(i)}_1, \ldots, x^{(i)}_n)\}$ when running Verify. As Π.Verify in our transformation directly calls Σ.AVerify, if $\mathcal{A}$ produces such a ω^*, then $\mathcal{B}$ can directly take ω^* as an input of Σ.AVerify. Clearly, when $\mathcal{A}$ can cheat Π.Verify and let $\mathbf{Exp}^{\text{sound}}_{\mathcal{A}}[\Pi, f, \kappa, n] = 1$, $\mathcal{B}$ can definitely cheat Σ.AVerify and let $\mathbf{Exp}^{\text{T-unforge}}_{\mathcal{B}}(\Sigma, \kappa, n, \ell) = 1$. Then We successfully construct such a $\mathcal{B}$ which breaks T-unforgeability when T = "Aggregator" with non-negl(κ): $\Pr[\mathbf{Exp}^{\text{T-unforge}}_{\mathcal{B}}(\Sigma, \kappa, n, \ell) = 1] \geq \Pr[\mathbf{Exp}^{\text{sound}}_{\mathcal{A}} (\Pi, f^*, \kappa, n) = 1] \geq$ non-negl(κ). However, as the T-unforgeability has been proven in HPRA, this makes a contradiction. Therefore, there is no PPT adversary $\mathcal{A}$ for which soundness of MVC does not hold.

Proof of Theorem 2

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

$$\Lambda_1 \leftarrow \Pi.\mathsf{Agg}(\overrightarrow{\mathsf{ak}}, \overrightarrow{\sigma_1}, \mu, f),\ \Lambda_2 \leftarrow \Pi.\mathsf{Agg}(\overrightarrow{\mathsf{ak}}, \overrightarrow{\sigma_2}, \mu, f).$$

Where $\overrightarrow{\sigma_1}$ is the signatures of $(\vec{m}_{11}, \ldots, \vec{m}_{n1})$ from Σ.Sign algorithm, $\overrightarrow{\sigma_2}$ is the signatures of $(\vec{m}_{12}, \ldots, \vec{m}_{n2})$ from Σ.Sign algorithm.

Recall the Definition 3, an MVC scheme Σ is called privacy against the first client if the view of the first client when running Σ with clients holding inputs $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$ is indistinguishable from the view of the first client when running Σ with clients holding inputs $(\vec{x}_1, \vec{x}'_2, \ldots, \vec{x}'_n)$. As the first client P_1 has no other opportunity to access information from all other entities when running Σ, except when running Verify algorithm: P_1 gets an encoded output $\omega^{(i)}$ which may reveals some information. Therefore, we only need to prove that the following distributions are identical:

$$\omega^{(i)} \leftarrow \Sigma.\mathsf{Compute}(i, (pk_j)_{j \in [n]}, \phi, \overrightarrow{\chi}^{(i)}),$$
$$\omega'^{(i)} \leftarrow \Sigma.\mathsf{Compute}(i, (pk_j)_{j \in [n]}, \phi, \overrightarrow{\chi}'^{(i)}).$$

Here $\overrightarrow{\chi}^{(i)}$ is the encoded from $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$ using $\Pi.\mathsf{EnInput}_1$ and $\Pi.\mathsf{EnInput}_j$, and $\overrightarrow{\chi}'^{(i)}$ is the encoded from $(\vec{x}_1, \vec{x}'_2, \ldots, \vec{x}'_n)$ using $\Pi.\mathsf{EnInput}_1$ and $\Pi.\mathsf{EnInput}_j$.

Without loss of generality, we can set $(\vec{m}_{11}, \ldots, \vec{m}_{n1}) := (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$ and $(\vec{m}_{12}, \ldots, \vec{m}_{n2}) := (\vec{x}_1,$

$\vec{x}'_2,\ldots,\vec{x}'_n)$, be any two vectors with same first component and $f(\vec{x}_1,\vec{x}_2,\ldots,\vec{x}_n) = f(\vec{x}_1,\vec{x}'_2\ldots,\vec{x}'_n)$. Very intuitively speaking, $\Pi.\mathsf{EnInput}_1$ and $\Pi.\mathsf{EnInput}_j$ both directly call the algorithm $\Sigma.\mathsf{Sign}$, except that $\Pi.\mathsf{EnInput}_1$ returns mk additionally, i.e., $\overrightarrow{\chi}^{(i)} = \overrightarrow{\sigma_1}$, $\overrightarrow{\chi}'^{(i)} = \overrightarrow{\sigma_2}$. Furthermore, $\Pi.\mathsf{Compute}$ algorithm exactly calls $\Sigma.\mathsf{Agg}$, $\omega^{(i)}$ and $\omega'^{(i)}$ are calculate exactly the same way as in the definition of input privacy of HPRA, i.e., $\omega(i) = \Lambda_1$ and $\omega'^{(i)} = \Lambda_2$. Due to the input privacy of HPRA, $\omega^{(i)}$ and $\omega'^{(i)}$ are clearly identical. Therefore, we proved that MVC scheme Σ is privacy against the first client.

Proof of Theorem 3

We show that if there exists a PPT adversary $\mathcal{A}$ for which privacy against the server of MVC on function f does not hold, then we can construct a PPT adversary $\mathcal{B}$ that break the output privacy of HPRA. $\mathcal{B}$ is given input $1^\kappa, n, \ell$ and access to an oracle $\mathcal{O} := \{\mathsf{SG}, \mathsf{SKey}, \mathsf{RoS}(b), \mathsf{SR}, \mathsf{VR}, \mathsf{VRKey}\}$. We follow the assumption of MVC that all the clients are honest but curious, in detail:

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

1. *The challenger runs* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa,\ell)$, $(\mathsf{mk}, \mathsf{aux}) \leftarrow \mathsf{VGen}(\mathsf{pp})$ *and chooses a random bit* $b \leftarrow \{0,1\}$*, then sends* pp *and* aux *to* $\mathcal{B}$.
2. *For* $j = 1,\ldots n$, $\mathcal{B}$ *queries the following oracles and gets* $(\mathsf{id}_j, \mathsf{pk}_j) \leftarrow \mathsf{SG}(j)$, $\perp \leftarrow \mathsf{SKey}(j)$, $\mathsf{rk}_j \leftarrow \mathsf{SR}(j)$. $\mathcal{B}$ *also queries the oracle* $\mathsf{VR}(j)$ *to generate* ak_j*, and queries* $\mathsf{ak}_j \leftarrow \mathsf{VRKey}(j)$.
3. $\mathcal{B}$ *then sets* $pk_1 = (\mathsf{id}_1, \mathsf{pk}_1, \mathsf{aux}, \mathsf{rk}_1)$*, for* $j = 2\ldots n$, $pk_j = (\mathsf{id}_j, \mathsf{pk}_j, \mathsf{rk}_j)$, $\phi = (f, \overrightarrow{ak})$ *and sends them to* $\mathcal{A}$*. Here* f *is 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 input* $(\vec{x}_1,\ldots,\vec{x}_n)$, $\mathcal{B}$ *answers the queries in following way:*
   a. *Sets* $i := i+1$.
   b. *Records* $(\vec{x}_1^{(i)},\ldots\vec{x}_n^{(i)}) := (\vec{x}_1,\ldots,\vec{x}_n)$.
   c. *Queries the oracle and gets* $((\overrightarrow{\sigma}^{(i)},\mu) \leftarrow \mathsf{Sig}(\{1,\ldots,n\},(\vec{x}_j^{(i)})_{j\in[n]})$.
   d. *Sets and returns* $\overrightarrow{\chi}^{(i)} := \overrightarrow{\sigma}^{(i)}$ *to* $\mathcal{A}$.
6. *When* $\mathcal{A}$ *outputs* $(\vec{x}_1^0,\ldots,\vec{x}_n^0),(\vec{x}_1^1,\ldots,\vec{x}_n^1)$, $\mathcal{B}$ *chooses a random bit* $b_0 \leftarrow \{0,1\}$ *and works as following:*
   a. *Queries* $\mathsf{RoS}(j,\vec{x}_1^{b_0},\ldots,\vec{x}_n^{b_0},b)$ *and obtains* $((\sigma_1^{b_0},c_1^{b_0}),\ldots,(\sigma_n^{b_0},c_n^{b_0}))$.
   b. *Returns the challenge ciphertext* $\overrightarrow{\chi}^{b_0} := \overrightarrow{\sigma}^{b_0}$ *to* $\mathcal{A}$.
7. *When* $\mathcal{A}$ *returns a bit* b_1 *to* $\mathcal{B}$, $\mathcal{B}$ *outputs* 1*, if* $b_0 = b_1$*, otherwise, outputs* 0.

Intuitively speaking, when $b_0 = b_1$, $\mathcal{A}$ figures out which one of inputs $(\vec{x}_1^0,\ldots,\vec{x}_n^0),(\vec{x}_1^1,\ldots,\vec{x}_n^1)$ is signed, and $\mathcal{B}$ tends to believe that $\mathsf{RoS}(b)$ does sign one of the inputs, and outputs a guess for the value of b, $b' = 1$. Otherwise, when $\mathcal{A}$ cannot figure out which one of inputs is signed, $\mathcal{B}$ tends to believe that $\mathsf{RoS}(b)$ signs a random string. Then $\mathcal{B}$ will outputs a guess for the value of b, $b' = 0$.

Recall the definition of private against the server in Section 2 Definition 3, we can figure out that when the privacy against the server of MVC against $\mathcal{A}$ does not hold:

$$\mathbf{Adv}^{\text{priv}}_{\mathcal{A}}(\Pi,f,\kappa,n) = \big|\Pr[\mathbf{Exp}^{\text{priv}}_{\mathcal{A}}(\Pi,f,\kappa,n,0)=1] - \Pr[\mathbf{Exp}^{\text{priv}}_{\mathcal{A}}(\Pi,f,\kappa,n,1)=1]\big| \geq \mathsf{non\text{-}negl}(\kappa).$$

Without loss of generality, we assume that the probability $\Pr[\mathbf{Exp}^{\text{priv}}_{\mathcal{A}}[\Pi,f,\kappa,n,1] = 1]$ is bigger than the probability $\Pr[\mathbf{Exp}^{\text{priv}}_{\mathcal{A}}[\Pi,f,\kappa,n,0] = 1]$. If $b = 0$, $\mathsf{RoS}(b)$ encrypts $(\vec{x}_1^{b_0},\ldots,\vec{x}_n^{b_0})$, then the view of $\mathcal{A}$ when run as a sub-routine by $\mathcal{B}$ is distributed identically to the view of $\mathcal{A}$ in experiment $\mathbf{Exp}^{\text{priv}}_{\mathcal{A}}[\Pi,f,\kappa,n,b_0]$. Thus,

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

Since b_0 is randomly chosen by $\mathcal{B}$, $\Pr[b_0 = 0] = \Pr[b_0 = 1] = 1/2$, we have:

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

If $b = 1$, $\mathsf{RoS}(b)$ encrypts a string of random numbers, then the view of $\mathcal{A}$ when run as a sub-routine by $\mathcal{B}$ is distributed identically to the view of $\mathcal{A}$ in an experiment of guessing random numbers. Thus,

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

Combining the above two equations, since b is randomly chosen by challenger, $\Pr[b = 0] = \Pr[b = 1] = 1/2$, if the output privacy of MVC can be broken by $\mathcal{A}$ with non-negligible probability $\mathsf{non\text{-}negl}(\kappa)$, we have:

$$\begin{aligned}\Pr[\mathbf{Exp}^{\text{outpriv}}_{\mathcal{B}}(\Sigma,\kappa,n,\ell)=1] &= 1/2\cdot\Pr[b'=1|b=0]\\ &+1/2\cdot\Pr[b'=1|b=1] \geq 1/2 + 1/4\cdot\mathsf{non\text{-}negl}(\kappa).\end{aligned}$$

However, as the output privacy has been proven in HPRA, makes a contradiction. Thus, if the output privacy is hold, then our MVC scheme also maintains the privacy against the server.