Verifiable Homomorphic Secret Sharing

Georgia Tsaloli(✉), Bei Liang, and Aikaterini Mitrokotsa

Chalmers University of Technology, Gothenburg, Sweden
{tsaloli,lbei,aikmitr}@chalmers.se

Abstract. In this paper, we explore the *multi-server* (*i.e.,* multiple servers are employed to perform computations) and *multi-client* (*i.e.,* multiple clients outsource joint computations on their joint inputs) scenario that avoids single points of failure and provides higher security and privacy guarantees. More precisely, we introduce the notion of *verifiable homomorphic secret sharing* (VHSS) for multi-input, that allows n clients to outsource joint computations on their joint inputs to m servers without requiring any communication between the clients or the servers; while providing the *verifiable capability* to any user to confirm that the final output (rather than each share) is correct. Our contributions are two-fold: *(i)* we provide a detailed example for casting Shamir's secret sharing scheme over a finite field $\mathbb{F}$ as an n-client, m-server, t-secure perfectly secure, additive HSS scheme for the function f that sums n field elements, and *(ii)* we propose an instantiation of an n-client, m-server, t-secure computationally secure, multiplicative VHSS scheme for the function f that multiplies n elements under the hardness assumption of the fixed inversion problem in bilinear maps.

Keywords: Function secret sharing · Homomorphic secret sharing Verifiable computation

1 Introduction

The emergence of ubiquitous computing has led to multiple heterogeneous devices with increased connectivity and have formed the Internet of Things (IoT). These IoT devices are often constrained regarding resources (*i.e.,* memory, bandwidth and computational resources) and thus, require the assistance of more powerful but often untrusted servers in order to store, process and perform computations on the collected data, leading to what is known as *cloud-assisted computing.* An important challenge in this cloud-assisted computing paradigm is how to protect the *security* and *privacy* of the participants considering the clients' resource-constraints, especially in the *multi-client* setting. Although the classical cloud-computing paradigm traditionally involves one client, we argue that a *multi-client* setting is more realistic since often an aggregator has to perform computations from data collected from multiple users. This is, for instance, the case when it is required to compute statistics for data collected from multiple users in order to monitor electricity consumption via smart metering, clinical data or even the safety of buildings or environmental conditions from data collected from multiple sensors.

Although a major part of existing work focuses on the *single client*, *single server* setting (*i.e.,* a single client outsourcing a computation to a single server) [2,10,11,13,17,18,21], we argue that not only the *multi-client* setting [12,15] is more realistic but also the *multi-server* setting (*i.e.,* multiple servers are employed in order to perform the computations) [1] provides better security guarantees and avoids single points of failure. The multi-server setting could also be adopted in multiple online services when users need to perform queries (*e.g.,* statistics on available data) to service providers, while at the same time have guarantees that no information can be inferred from the users' queries by the servers. For instance, a user (client) may split her query into multiple shares and send each share to a different server [22]. Similarly, in a smart electricity consumption application setting, multiple servers could be employed to collect data for the electricity consumption from multiple sensors (clients). As long as at least one of the servers is honest and does not collude with the others, the servers cannot recover any sensitive information. However, given responses from all the servers, the user can compute the answer to her query. This multi-server paradigm provides higher security guarantees since single points of failures are avoided.

In this paper, we consider the problem of outsourcing computations and providing strong security and privacy guarantees when: *(i) multiple-clients* outsource joint computations on their joint secret inputs, *(ii) multiple-servers* are employed for the computations, and *(iii) anyone* can verify that the combination of the shares is correct. More precisely, we investigate how we may outsource computations from multiple clients to multiple untrusted servers without requiring any communication between the clients or the servers *i.e.,* all information required for the computations are shared publicly and thus no communication overhead is required. We consider functional secret sharing schemes that can be employed to compute a function (addition or multiplication) of multiple secrets. This is achieved by enabling the servers to locally convert the shares of the different secrets into a (multiplicative or additive) function of their shares. Furthermore, the servers are able to locally generate shares of a proof that guarantees that the product of all the shares is correct. We focus on specific functions, the addition and the multiplication, and we employ, as building tools, *verifiable homomorphic secret sharing* schemes. The result is the definition and the first concrete construction of a *verifiable multiplicative homomorphic secret sharing* scheme.

Homomorphic Secret Sharing. A threshold secret sharing scheme [20] allows a dealer to randomly split a secret x into m shares, $(x^1, \ldots, x^m)$, such that certain subsets of the shares can be used to reconstruct the secret and others reveal nothing about it. Motivated by the powerful cryptographic functionality of fully homomorphic encryption (FHE) [14,19] which supports arbitrary computations on encrypted inputs, Boyle *et al.* [8] introduced the natural notion of *homomorphic secret sharing* (HSS) that achieves some of the functionality offered by FHE [6]. An HSS scheme supports computations on shared inputs based on local computations on their shares. More concretely, there is a local evaluation algorithm Eval and a decoder algorithm Dec satisfying the following homomorphism requirement. Given a description of a function F, the algorithm $\mathsf{Eval}(F, x^j)$ maps an input share x^j to a corresponding output share y^j, such that $\mathsf{Dec}(y^1, \ldots, y^m) = F(x)$. Analogously to the *output compactness* requirement of FHE, in HSS the output shares are *compact* in the sense that the output length of Eval, and hence the complexity of Dec, depends only on the output length of F and the security parameter, but not on the input length of F. The simplest type of HSS is the *additive HSS*, where the Dec algorithm computes $F(x)$ as the sum $y^1 + \ldots + y^m$ in some finite Abelian group, which is the first instance of HSS considered in the literature by Benaloh [4]. Boyle *et al.* [9] naturally consider a multi-input variant of HSS, where inputs $x_1, \ldots, x_n$ are independently shared, Eval locally maps the j-th shares of the n inputs to the j-th output share, and Dec outputs $F(x_1, \ldots, x_n)$.

Our Contributions. In this paper, we introduce the notion of *verifiable homomorphic secret sharing* (VHSS) for multi-input. We call a multi-input homomorphic secret sharing (HSS) scheme verifiable if the scheme enables the clients (users) to locally generate shares of a proof which confirms that the combination of the shares (rather than each share) is correct. We expect that the verifiability property can be employed for making multi-party computations (MPC) secure in the presence of an active adversary by accepting the output only if the correctness is verified.

Firstly, we provide a detailed example for casting Shamir's secret sharing scheme [20] over a finite field $\mathbb{F}$ as a n-client, m-server, t-secure perfectly secure, additive HSS scheme for the function f that sums n field elements. Such a scheme exists if and only if $m > n{\cdot}t$. Secondly, we propose an instantiation of an n-client, m-server, t-secure computationally secure, multiplicative VHSS scheme for the function f that multiplies n elements under the hardness assumption of the fixed inversion problem in bilinear maps. More precisely, we present a scheme where there are n clients $c_1, \ldots, c_n$ each of whom shares its secret input x_i to m servers $s_1, \ldots, s_m$. Each server's share of x_i is denoted as x_{ij}. For each $j \in \{1, \ldots, m\}$, the server s_j that possesses n shared inputs $x_{1j}, \ldots, x_{nj}$ generates a share y^j as well as a share σ^j of a proof that the product of m shares is correct. In our multiplicative VHSS instantiation, each client c_i has x_i but also $\widetilde{x}_i$ such that $g^{\widetilde{x}_i} = x_i$ where g denotes a generator of the multiplicative group of $\mathbb{F}$.

1.1 Related Work

Multiplicative Secret Sharing. A multiplicative secret sharing scheme allows two parties to multiply two secret-shared field elements by locally converting their shares of the two secrets into an additive sharing of their product. Barkol *et al.* [3] consider a different natural extension of the basic multiplication property of secret sharing that is called d-multiplication. The d-multiplication property generalizes standard multiplication by considering a multiplication of d (rather than two) secrets. Specifically, a secret sharing with d-multiplication allows multiplying d secret-shared field elements by enabling the players to locally convert shares of d different secrets into an additive sharing of their product. They also proved that d-multiplicative schemes exist if and only if no d unauthorized sets of players cover the whole set of players. In particular, t-private d-multiplicative secret sharing among m players is possible only if $m > d{\cdot}t$ where t-private means that every set of t players is unauthorized.

In fact, d-multiplicative secret sharing (d-multiplicative SS) among m players is a specific case of Boyel *et al.*'s [9] multi-input variant of HSS, where the Eval algorithm of HSS can be specified as the MULT algorithm of d-multiplicative SS, while the Dec algorithm of HSS can be specified as the summation operation on the outcomes of the m local computations.

Verifiable Multiplicative Secret Sharing. Following Barkol *et al.*'s [3] work on d-multiplicative secret sharing, Yoshida *et al.* [23] introduced the notion of verifiably d-multiplicative SS, which enables the players to locally generate an additive sharing of a proof that the sum of shares (rather than each share) is correct. Actually, our verifiable HSS for multi-input is a more general notion for verifiably d-multiplicative SS, since we generalize the reconstructing operation on local outcomes and local proofs, *e.g.,* using the algorithms FinalEval and FinalProof respectively.

We need to note that in both works of d-multiplicative SS [3] and verifiably d-multiplicative SS [23], no instantiation of a verifiable multiplicative HSS scheme was proposed. On the contrary, the authors assume that the local computation algorithms MULT and PF on d shares exist without though providing any instantiation. In this paper, we instantiate for the first time the MULT and PF algorithms of a verifiable multiplicative secret sharing scheme as a product on the d shares and bilinear map operations respectively.

Verifiable Functional Secret Sharing. Boyle *et al.* [5] after introducing the notion of functional secret sharing (FSS), they have also introduced the notion of verifiable FSS [7], where on the one hand a function f is split into m functions $f_1, \ldots, f_m$, described by the corresponding keys $k_1, \ldots, k_m$, such that for any input x we have that $f(x) = f_1(x) + \ldots + f_m(x)$ and every strict subset of the keys hides f; on the other hand, there is an additional m-parties interactive protocol Ver for verifying that the keys $(k_1^*, \ldots, k_m^*)$, generated by a potentially malicious client, are consistent with some f. Compared to Boyle *et al.*'s notion of verifiable FSS which is applied to the one client (one input) and multi-server setting, our VHSS works on the multi-client (multi-input) and multi-server setting. Furthermore, by employing a verification algorithm, Boyle *et al.*'s VFSS goal is to convince all involved parties that the function effectively shared by the client is consistent with some f. However, in our proposed notion of VHSS, the verification algorithm is employed to enable the servers to locally generate shares of a proof that guarantees that the combination (such as the product) of all shares (rather than each share) is correct.

Organization. The paper is organized as follows. In Sect. 2, we present the general definitions for the homomorphic secret sharing (HSS) and the verifiable homomorphic secret sharing (VHSS) schemes. In Sect. 3, we provide a concrete construction for the additive HSS scheme as well as a proof of its correctness and the corresponding security proof. In Sect. 4, we present our proposed multiplicative VHSS scheme as well as the assumption it relies on, the proposed concrete multiplicative VHSS construction and the corresponding proofs of correctness, verifiability, and security.

2 General Definitions for the HSS and the VHSS

In this section, we will formulate a general definition of homomorphic secret sharing (HSS) inspired by Boyle *et al.*'s [9] definition, which is the base of our *verifiable homomorphic secret sharing* (VHSS) definition that will follow.

We consider n clients $c_1, c_2, \dots, c_n$ that split their inputs $x_1, \dots, x_n$ between m servers using the algorithm **ShareSecret**, in such a way that each x_i is hidden from any t servers that could be corrupted. Each server s_j, having its share of the n inputs, applies the algorithm **PartialEval** in order to get and publish the partial share y^j. Finally, any user may apply the algorithm **FinalEval** in order to obtain $f(x_1, x_2, \dots, x_n)$ where f is a function such that $f : \mathcal{X} \mapsto \mathcal{Y}$ for $\mathcal{X}$ to be a domain and $\mathcal{Y}$ be a target set respectively.

The verifiable homomorphic secret sharing (VHSS) scheme is based on the HSS, and it provides additionally the notion of *verifiability*. Most precisely, each server s_j applies the algorithm **PartialEval** to obtain the partial share y^j but also it applies the algorithm **PartialProof** to compute σ^j, where the latter is the share of the proof that the final computation is correct. Furthermore, any user that would like to get y by running the algorithm **FinalEval** is also able to run the algorithm **FinalProof** which gives the proof σ that the value y is correct. By employing the algorithm **Verify**, each user is able to check that what she gets is actually the output that corresponds to $f(x_1, x_2, \dots, x_n)$.

We will now give the definitions of a general *homomorphic secret sharing* (HSS) scheme and a *verifiable homomorphic secret sharing scheme* (VHSS).

Definition 1. *An n-client, m-server, t-secure homomorphic secret sharing (HSS) scheme for a function $f : \mathcal{X} \mapsto \mathcal{Y}$, is a 3-tuple of PPT algorithms (**ShareSecret**, **PartialEval**, **FinalEval**) which are defined as follows:*

– $(x_{i1}, x_{i2}, \dots, x_{im}) \leftarrow$ ***ShareSecret***$(1^\lambda, i, x_i)$*: On input 1^λ, where λ is the security parameter, $i \in \{1, \dots, n\}$ which is the index for the client c_i and $x_i \in \mathcal{X}$ which is her secret input, the algorithm **ShareSecret** outputs m shares for the corresponding secret input x_i.*

– $y^j \leftarrow$ ***PartialEval***$(j, (x_{1j}, x_{2j}, \dots, x_{nj}))$*: On input $j \in \{1, \dots, m\}$ which denotes the index of the server s_j, and $x_{1j}, x_{2j}, \dots, x_{nj}$ which are the shares of the n secret inputs that the server s_j has, the algorithm **PartialEval** outputs $y^j \in \mathcal{Y}$.*

- $y \leftarrow$***FinalEval***$(y^1, y^2, \ldots, y^m)$*: On input* $y^1, y^2, \ldots, y^m$*, which are the shares of* $f(x_1, x_2, \ldots, x_n)$ *that the* m *servers have, the algorithm* ***FinalEval*** *outputs* y*, the final result for* $f(x_1, x_2, \ldots, x_n)$*.*

The algorithms (**ShareSecret**, **PartialEval**, **FinalEval**) should satisfy the following correctness and security requirements:

• **Correctness**: For any n secret inputs $x_1, \ldots, x_n$, for all $(x_{i1}, x_{i2}, \ldots, x_{im})$ computed for all $i \in [n]$ from the algorithm **ShareSecret**, for all y^j computed for all $j \in [m]$ from the algorithm **PartialEval**, the scheme should satisfy the following correctness requirement:

$$\Pr\left[\mathbf{FinalEval}(y^1, y^2, \ldots, y^m) = f(x_1, x_2, \ldots, x_n)\right] = 1.$$

• **Security**: Let T be the set of the corrupted servers with $|T| < m$. Consider the following semantic security challenge experiment:

1. The adversary $\mathcal{A}$ gives $(i, x_i, x_i') \leftarrow \mathcal{A}(1^\lambda)$ to the challenger where $i \in [n]$, $x_i \neq x_i'$ and $|x_i| = |x_i'|$.
2. The challenger picks a bit $b \in \{0, 1\}$ uniformly at random and computes $(\hat{x}_{i1}, \hat{x}_{i2}, \ldots, \hat{x}_{im}) \leftarrow \mathbf{ShareSecret}(1^\lambda, i, \hat{x}_i)$ where $\hat{x}_i = \begin{cases} x_i, & \text{if } b = 0 \\ x_i', & \text{otherwise} \end{cases}$.
3. The adversary outputs a guess $b' \leftarrow \mathcal{A}((x_{ij})_{j|s_j \in T})$, given the shares from the corrupted servers T.

Let $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ be the advantage of $\mathcal{A}$ in guessing b in the above experiment, where the probability is taken over the randomness of the challenger and of $\mathcal{A}$. The scheme (**ShareSecret**, **PartialEval**, **FinalEval**) is t-secure if for all $T \subset \{s_1, \ldots, s_m\}$ with $|T| \leq t$, and all PPT adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

Definition 2. *An* n*-client,* m*-server,* t*-secure verifiable homomorphic secret sharing (VHSS) scheme for a function* $f : \mathcal{X} \mapsto \mathcal{Y}$*, is a 6-tuple of PPT algorithms (**ShareSecret**, **PartialEval**, **PartialProof**, **FinalEval**, **FinalProof**, **Verify**) which are defined as follows:*

- $(x_{i1}, x_{i2}, \ldots, x_{im}, \tau_i) \leftarrow$ ***ShareSecret***$(1^\lambda, i, x_i)$*: On input* 1^λ*, where* λ *is the security parameter,* $i \in \{1, \ldots, n\}$ *which is the index for the client* c_i *and* $x_i \in \mathcal{X}$ *which is her secret input, the algorithm* ***ShareSecret*** *outputs* m *shares for the corresponding secret input* x_i *as well as a publicly available encoded value* τ_i *related to the secret* x_i*.*
- $y^j \leftarrow$ ***PartialEval***$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$*: On input* $j \in \{1, \ldots, m\}$ *which denotes the index of the server* s_j*, and* $x_{1j}, x_{2j}, \ldots, x_{nj}$ *which are the shares of the* n *secret inputs that the server* s_j *has, the algorithm* ***PartialEval*** *outputs* $y^j \in \mathcal{Y}$*.*
- $\sigma^j \leftarrow$***PartialProof***$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$*: On input* j *(the server's index) and the* n *shares* $x_{1j}, x_{2j}, \ldots, x_{nj}$*, the algorithm* ***PartialProof*** *outputs* σ^j*. This output is the share of the proof that the final output is correct.*

- $y \leftarrow$***FinalEval***$(y^1, y^2, \dots, y^m)$*: On input* $y^1, y^2, \dots, y^m$ *which are the shares of* $f(x_1, x_2, \dots, x_n)$ *that the* m *servers have, the algorithm* ***FinalEval*** *outputs* y*, the final result for* $f(x_1, x_2, \dots, x_n)$*.*
- $\sigma \leftarrow$***FinalProof***$(\sigma^1, \sigma^2, \dots, \sigma^m)$*: On input the shares* $\sigma^1, \sigma^2, \dots, \sigma^m$*, the algorithm* ***FinalProof*** *outputs* σ *which is the proof that* y *is the correct value.*
- $0/1 \leftarrow$***Verify***$(\tau_1, \dots, \tau_n, \sigma, y)$*: On input the final result* y *together with its proof* σ*, as well as the encoded values* $\tau_1, \dots, \tau_n$ *the algorithm* ***Verify*** *outputs either* 0 *or* 1*.*

The algorithms (**ShareSecret**, **PartialEval**, **PartialProof**, **FinalEval**, **FinalProof**, **Verify**) should satisfy the following correctness, verifiability and security requirements:

• **Correctness**: For any n secret inputs $x_1, \dots, x_n$, for all $(x_{i1}, x_{i2}, \dots, x_{im}, \tau_i)$ computed for all $i \in [n]$ from the algorithm **ShareSecret**, for all y^j and σ^j computed for all $j \in [m]$ from the algorithms **PartialEval** and **PartialProof** respectively, and for y and σ generated by the algorithms **FinalEval** and **FinalProof** respectively, the scheme should satisfy the following correctness requirement:

$$\Pr\left[\textbf{Verify}(\tau_1, \dots, \tau_n, y, \sigma) = 1\right] = 1.$$

• **Verifiability**: Consider n secret inputs $x_1, x_2, \dots, x_n \in \mathbb{F}$, T the set of corrupted servers with $|T| \leqslant m$ and a PPT adversary $\mathcal{A}$. Any PPT adversary who modifies the shares of the secret inputs for any j such that $s_j \in T$, can cause a wrong value to be accepted as $f(x_1, x_2, \dots, x_n)$ with negligible probability. We define the following experiment:

$\mathbf{Exp}_{\text{VHSS}}^{\text{Verif.}}(x_1, x_2, \dots, x_n, T, \mathcal{A})$:

1. For all $i \in [n]$, generate $(x_{i1}, x_{i2}, \dots, x_{im}, \tau_i) \leftarrow$ **ShareSecret**$(1^\lambda, i, x_i)$ and publish $\tau_i, i \in [n]$.
2. For all j such that $s_j \in T$, give $\begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix}$ to the adversary.
3. The adversary $\mathcal{A}$ outputs modified multiplicative shares $y^{j'}$ and $\sigma^{j'}$ for j such that $s_j \in T$. For j such that $s_j \notin T$, we define the multiplicative shares $y^{j'} =$ **PartialEval**$(j, (x_{1j}, x_{2j}, \dots, x_{nj}))$ and $\sigma^{j'} =$ **PartialProof**$(j, (x_{1j}, x_{2j}, \dots, x_{nj}))$.
4. Compute the modified final value $y' =$ **FinalEval**$(y^{1'}, y^{2'}, \dots, y^{m'})$ and the modified final proof $\sigma' =$ **FinalProof**$(\sigma^{1'}, \sigma^{2'}, \dots, \sigma^{m'})$.
5. If $y' \neq f(x_1, x_2, \dots, x_n)$ and **Verify**$(\tau_1, \dots, \tau_n, \sigma', y') = 1$, then output 1 else 0.

We require that for any n secret inputs $x_1, x_2, \dots, x_n \in \mathbb{F}$, any set T of corrupted servers and any PPT adversary $\mathcal{A}$ it holds:

$$\Pr[\mathbf{Exp}_{\text{VHSS}}^{\text{Verif.}}(x_1, x_2, \dots, x_n, T, \mathcal{A}) = 1] \leq \varepsilon.$$

• **Security**: Let T be the set of the corrupted servers with $|T| < m$. Consider the following semantic security challenge experiment:

1. The adversary $\mathcal{A}$ gives $(i, x_i, x_i') \leftarrow \mathcal{A}(1^\lambda)$ to the challenger where $i \in [n]$, $x_i \neq x_i'$ and $|x_i| = |x_i'|$.
2. The challenger picks a bit $b \in \{0, 1\}$ uniformly at random and computes $(\hat{x}_{i1}, \hat{x}_{i2}, \ldots, \hat{x}_{im}, \hat{\tau}_i) \leftarrow$ **ShareSecret**$(1^\lambda, i, \hat{x}_i)$ where $\hat{\tau}_i$ is an encoded value related to $\hat{x}_i$ and $\hat{x}_i = \begin{cases} x_i, \text{if } b = 0 \\ x_i', \text{otherwise} \end{cases}$.
3. The adversary outputs a guess $b' \leftarrow \mathcal{A}((x_{ij})_{j|s_j \in T}, (\tau_i)_{i \in [n]})$, given the shares from the corrupted servers T and the encoded values $\tau_1, \ldots, \tau_n$.

Let $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ be the advantage of $\mathcal{A}$ in guessing b in the above experiment, where the probability is taken over the randomness of the challenger and of $\mathcal{A}$. The VHSS scheme is t-secure if for all $T \subset \{s_1, \ldots, s_m\}$ with $|T| \leq t$, and all PPT adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

3 Additive Homomorphic Secret Sharing Scheme

In this chapter, we present a detailed example of the additive HSS scheme. Therefore, we consider n clients $c_1, \ldots, c_n$, their secret inputs $x_1, x_2, \ldots, x_n$ respectively and one or more users that would like to compute the sum of these secret inputs, that is, they want to compute $f(x_1, x_2, \ldots, x_n) = x_1 + x_2 + \ldots + x_n$ without knowing $x_1, \ldots, x_n$.

3.1 Construction of the Additive HSS

We consider m servers $s_1, \ldots, s_m$. Let $\mathbb{F}$ be a finite field with $|\mathbb{F}| > m$, λ be the security parameter, let, $\forall i \in [n]$, $\theta_{i1}, \ldots, \theta_{im}$ be distinct nonzero field elements, and let, for any $i \in \{1, \ldots, n\}$, $\lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{im}$ be field elements such that for any univariate polynomial p_i of degree t over $\mathbb{F}$ we have $p_i(0) = \sum_{j=1}^{m} \lambda_{ij} p_i(\theta_{ij})$. Each client c_i will distribute her secret's share to the servers so that the latter will compute the partial sum y^i and then, any user can easily compute the final sum by adding the partial sums that the servers have computed without having any information about the secret inputs. More precisely, we have the following algorithms:

1. **ShareSecret**$(1^\lambda, i, x_i)$: Pick a polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + \ldots + a_t X^t$ where $\{a_i\}_{i \in \{1, \ldots, t\}} \in \mathbb{F}$ are elements selected uniformly at random and t denotes the degree of the polynomial with $t \cdot n < m$. Notice that the free coefficient of p_i is the secret input x_i. Then, output $(x_{i1}, x_{i2}, \ldots, x_{im}) = (\lambda_{i1} \cdot p_i(\theta_{i1}), \lambda_{i2} \cdot p_i(\theta_{i2}), \ldots, \lambda_{im} \cdot p_i(\theta_{im}))$.
2. **PartialEval**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$: For the given j and for all $i \in [n]$, compute the sum of all $x_{ij} = \lambda_{ij} \cdot p_i(\theta_{ij})$. Output $y^j = \lambda_{1j} \cdot p_1(\theta_{1j}) + \lambda_{2j} \cdot p_2(\theta_{2j}) + \ldots + \lambda_{nj} \cdot p_n(\theta_{nj}) = \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})$.

3. **FinalEval**$(y^1, y^2, \ldots, y^m)$: Add the partial shares together and output $y = y^1 + \ldots + y^m$.

Now, each client c_i runs **ShareSecret** and gives $\lambda_{ij} \cdot p_i(\theta_{ij})$ to each server s_j. Table 1 shows how each client c_i distributes her secret input x_i to the servers. Then, each server s_j has the shares $\lambda_{1j} \cdot p_1(\theta_{1j}), \lambda_{2j} \cdot p_2(\theta_{2j}), \ldots, \lambda_{mj} \cdot p_m(\theta_{mj})$, thus, she computes the partial sum y^j after running **PartialEval** and she publishes it. Finally, any user is able to get the total sum y by running **FinalEval**.

Table 1. Additive Homomorphic Secret Sharing

Secret inputs	Servers				
	s_1	s_2	...	...	s_m
x_1	$\lambda_{11} \cdot p_1(\theta_{11})$	$\lambda_{12} \cdot p_1(\theta_{12})$	...	...	$\lambda_{1m} \cdot p_1(\theta_{1m})$
x_2	$\lambda_{21} \cdot p_2(\theta_{21})$	$\lambda_{22} \cdot p_2(\theta_{22})$	...	...	$\lambda_{2m} \cdot p_2(\theta_{2m})$
.	.	.			.
.	.	.			.
.	.	.			.
x_n	$\lambda_{n1} \cdot p_n(\theta_{n1})$	$\lambda_{n2} \cdot p_n(\theta_{n2})$	...	...	$\lambda_{nm} \cdot p_n(\theta_{nm})$
Partial sum	y^1	y^2	...	...	y^m
Total sum	y				

3.2 Correctness of the Additive HSS

We may now confirm that even though the clients in the additive HSS do not reveal their secret inputs $x_1, \ldots, x_n$, it is still possible for a user to compute the total sum with probability 1. It suffices to show that

$$y = x_1 + \ldots + x_n.$$

We can get to this as follows: By construction, it holds that $y = \sum_{j=1}^{m} y^j$ and $y^j = \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})$. This implies that

$$y = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_{ij} \cdot p_i(\theta_{ij}).$$

However, it is also true that $p_i(0) = \sum_{j=1}^{m} \lambda_{ij} p_i(\theta_{ij})$ which implies that $y = \sum_{i=1}^{n} p_i(0)$. Now, by construction (see **ShareSecret**), $p_i(0) = x_i$ which gives

$$y = \sum_{i=1}^{n} x_i = x_1 + x_2 + \ldots + x_n$$

as we wished.

3.3 Security of the Additive HSS

Theorem 1. *For all $T \subset \{s_1, \ldots, s_m\}$ with $|T| \leq m-1$ and all PPT adversaries $\mathcal{A}$, the additive HSS scheme (**ShareSecret**, **PartialEval**, **FinalEval**) is $(m-1)$-secure. It holds that $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.*

Proof. Let $|T| = m-1$, that is, there are $m-1$ corrupted servers.
The adversary $\mathcal{A}$ has $(m-1)n$ shares from the corrupted servers and no information that any $(m-1)$-tuple is related either to x_i or to x_i'. Consider, without loss of generality, that the first $m-1$ servers are the corrupted ones.

If we denote the shares for any $i \in [n]$ by $\hat{x}_{i1}, \ldots, \hat{x}_{im}$, it is true that $\sum_{j=1}^{m} \hat{x}_{ij} = \hat{x}_i$ for some i. We may also see this equality as $\hat{x}_{im} = \hat{x}_i - \sum_{j=1}^{m-1} \hat{x}_{ij}$. Then, any PPT adversary has no information whether $\hat{x}_{im} \in \mathcal{Y}$ is the m-th share of x_i or ${x_i}'$ and thus, may guess whether $\hat{x}_m$ corresponds to x_i or ${x_i}'$ with probability 1/2. That gives that the adversary can guess whether $\hat{x}_i$ is x_i or ${x_i}'$ with probability 1/2 as well. Therefore, it holds that $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

4 Multiplicative Verifiable Homomorphic Secret Sharing Scheme

In this chapter, we present a concrete instantiation of the multiplicative verifiable homomorphic secret sharing (VHSS) scheme for which we use the notion of the bilinear maps. A bilinear map is a function that is defined as follows:

Definition 3. *Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_k$ be cyclic groups of the same order. A bilinear map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_k$ is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_k$ such that for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z}$,*

$$e(u^a, v^b) = e(u, v)^{ab}.$$

In our instantiation, however, we consider the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_k$ where $\mathbb{G}, \mathbb{G}_k$ are cyclic groups of the same order.

For the security of the multiplicative VHSS, we need that the inversion of the bilinear map when one of the inputs is fixed does not exist. We call such an inversion by the fixed inversion problem (FI). More formally, the FI problem is defined as follows:

Definition 4 [16]**.** *For a fixed $g \in \mathbb{G}$ and any given $h \in \mathbb{G}_k$, the problem which finds an inverse image g' such that $e(g, g') = h$ is called fixed inversion problem (FI).*

In order to meet the multiplicative VHSS's security requirement, we yield the following assumption:

Assumption 1. *Given the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_k$ and the values $g \in \mathbb{G}_1, g' \in \mathbb{G}_2$ and $h \in \mathbb{G}_k$, we assume that the fixed inversion problem (FI) is hard.*

What is more, assuming that the FI problem is hard implies that the discrete logarithm problem (DLP) is hard. More precisely, we give what the discrete logarithm problem is as well as the relation between FI and DLP:

Definition 5. *For any given two values g and g^a in a group $\mathbb{G}$ the problem which computes a is called the discrete logarithm problem (DLP).*

Observation 1 [16]**.** *If the fixed inversion (FI) problem is hard, then the discrete logarithm problem (DLP) in $\mathbb{G}_k$ is hard.*

4.1 Construction of the Multiplicative VHSS

Let us consider n clients $c_1, \ldots, c_n$ who will again split their secret inputs $x_1, x_2, \ldots, x_n$ to m servers $s_1, \ldots, s_m$, $\mathbb{F}$ to be a finite field with $|\mathbb{F}| > m$ and λ be the security parameter. The clients (users) are able to compute their secret inputs' product by multiplying the partial products that have been computed by each server. In other words, a user can compute $f(x_1, x_2, \ldots, x_n) = x_1 \cdot x_2 \cdot \ldots \cdot x_n$ without knowing $x_1, \ldots, x_n$.

In this setting, each client c_i has x_i but also $\widetilde{x}_i$ such that $g^{\widetilde{x}_i} = x_i$ where g denotes a generator of the multiplicative group of $\mathbb{F}$. Furthermore, each server will not only publish the partial product but also a share of a proof. As a result, any user is able to use the shares of the proof in order to obtain the proof and verify that the final product is correct.

Let, for any $i \in \{1, \ldots, n\}$, $\theta_{i1}, \ldots, \theta_{im}$ be distinct nonzero field elements and $\lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{im}$ be field elements ("Lagrange coefficients") such that for any univariate polynomial p_i of degree t over $\mathbb{F}$ we have $p_i(0) = \sum_{j=1}^{m} \lambda_{ij} p_i(\theta_{ij})$. For any $j \in \{1, \ldots, m\}$, the share of the proof that will be published by the server s_j is denoted by σ^j. We consider the following algorithms:

1. **ShareSecret**$(1^\lambda, i, \widetilde{x}_i)$: Pick a polynomial p_i of the form $p_i(X) = \widetilde{x}_i + a_1 X + a_2 X^2 + \ldots + a_t X^t$ where $a_i, i \in \{1, \ldots, n\}$ are elements selected uniformly at random, t denotes the degree of the polynomial with $t \cdot n < m$ and $\widetilde{x}_i$ its free coefficient. Then, the algorithm outputs $(x_{i1}, x_{i2}, \ldots, x_{im}, \tau_i) = (g^{\lambda_{i1} \cdot p_i(\theta_{i1})}, g^{\lambda_{i2} \cdot p_i(\theta_{i2})}, \ldots, g^{\lambda_{im} \cdot p_i(\theta_{im})}, e(g, g^{\widetilde{x}_i}))$. Note that $\tau_i = e(g, g^{\widetilde{x}_i})$.
2. **PartialEval**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$: For the given j and for all $i \in [n]$, multiply all x_{ij}, that is, compute $x_{1j} \cdot x_{2j} \cdot \ldots \cdot x_{nj} = g^{\lambda_{1j} \cdot p_1(\theta_{1j})} \cdot g^{\lambda_{2j} \cdot p_2(\theta_{2j})} \cdot \ldots \cdot g^{\lambda_{nj} \cdot p_n(\theta_{nj})} = \prod_{i=1}^{n} g^{\lambda_{ij} \cdot p_i(\theta_{ij})} = y^j$. Output y^j.
3. **PartialProof**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$: For the given j and $x_{1j}, x_{2j}, \ldots, x_{nj}$, compute the partial proof, that is, the share of the proof, $\sigma^j = e(g, x_{1j} \cdot x_{2j} \cdot \ldots \cdot x_{nj}) = e(g, y^j)$ where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_k$. Output σ^j.
4. **FinalEval**$(y^1, y^2, \ldots, y^m)$: Multiply the partial products y^j for $j \in [m]$, that is, compute $y^1 \cdot y^2 \cdot \ldots \cdot y^m = y$. Output y.
5. **FinalProof**$(\sigma^1, \sigma^2, \ldots, \sigma^m)$: Multiply the partial proofs to get $\sigma^1 \cdot \ldots \cdot \sigma^m = \sigma$. Output σ.
6. **Verify**$(\tau_1, \ldots, \tau_n, \sigma, y)$: Check that $\prod_{i=1}^{n} \tau_i = \sigma$ and $\prod_{i=1}^{n} \tau_i = e(g, y)$. Output: 1 if both are satisfied or 0 otherwise.

Each client c_i splits her secret input x_i by running **ShareSecret** for $\widetilde{x}_i$ and give to each server s_j the share $x_{ij} = g^{\lambda_{ij} \cdot p_i(\theta_{ij})}$. **ShareSecret** also outputs the encoded value $\tau_i, i \in [n]$ which will be published by the client c_i. Table 2 shows how the secret inputs are distributed. Then, the server s_j runs **PartialEval** to compute and publish the partial product $y^j = \prod_{i=1}^{n} g^{\lambda_{ij} \cdot p_i(\theta_{ij})}$. Each server s_j applies also the algorithm **PartialProof** to compute and publish a partial proof $\sigma^j = e(g, y^j)$. Now, any user, not having any secret input $x_1, \ldots, x_n$, applies **FinalEval** to get the value $y = f(x_1, x_2, \ldots, x_n) = x_1 \cdot x_2 \cdot \ldots \cdot x_n$ and **FinalProof** to get the proof $\sigma = \sigma^1 \cdot \ldots \cdot \sigma^m$ that y is correct. Any user may now run the algorithm **Verify** to confirm that y is actually the product of $x_1, x_2, \ldots, x_n$, which will be the case if **Verify** outputs 1.

Table 2. Multiplicative Homomorphic Secret Sharing

Secret inputs	Public	Servers				
	value	s_1	s_2	...	...	s_m
x_1	τ_1	$g^{\lambda_{11} \cdot p_1(\theta_{11})}$	$g^{\lambda_{12} \cdot p_1(\theta_{12})}$	...	...	$g^{\lambda_{1m} \cdot p_1(\theta_{1m})}$
x_2	τ_2	$g^{\lambda_{21} \cdot p_2(\theta_{21})}$	$g^{\lambda_{22} \cdot p_2(\theta_{22})}$	...	...	$g^{\lambda_{2m} \cdot p_2(\theta_{2m})}$
.	.	.	.			.
.	.	.	.			.
.	.	.	.			.
x_n	τ_n	$g^{\lambda_{n1} \cdot p_n(\theta_{n1})}$	$g^{\lambda_{n2} \cdot p_n(\theta_{n2})}$			$g^{\lambda_{nm} \cdot p_n(\theta_{nm})}$
Partial product		y^1	y^2			y^m
Partial proof		σ^1	σ^2	...	...	σ^m
(Product, proof)		(y, σ)				

4.2 Correctness of the Multiplicative VHSS

To confirm the correctness of the multiplicative VHSS, it suffices to show that $\prod_{i=1}^{n} \tau_i = \sigma$ and $\prod_{i=1}^{n} \tau_i = e(g, y)$. In fact,

$$\begin{aligned}
\sigma &= \sigma^1 \cdot \ldots \cdot \sigma^m = e(g, y^1) \cdot e(g, y^2) \cdot \ldots \cdot e(g, y^m) \\
&= e(g, \prod_{i=1}^{n} g^{\lambda_{i1} \cdot p_i(\theta_{i1})}) \cdot e(g, \prod_{i=1}^{n} g^{\lambda_{i2} \cdot p_i(\theta_{i2})}) \cdot \ldots \cdot e(g, \prod_{i=1}^{n} g^{\lambda_{im} \cdot p_i(\theta_{im})}) \\
&= e(g, g^{\sum_{i=1}^{n} \lambda_{i1} \cdot p_i(\theta_{i1})}) \cdot e(g, g^{\sum_{i=1}^{n} \lambda_{i2} \cdot p_i(\theta_{i2})}) \cdot \ldots \cdot e(g, g^{\sum_{i=1}^{n} \lambda_{im} \cdot p_i(\theta_{im})}) \\
&= e(g, g)^{\sum_{i=1}^{n} \lambda_{i1} \cdot p_i(\theta_{i1})} \cdot e(g, g)^{\sum_{i=1}^{n} \lambda_{i2} \cdot p_i(\theta_{i2})} \cdot \ldots \cdot e(g, g)^{\sum_{i=1}^{n} \lambda_{im} \cdot p_i(\theta_{im})} \\
&= e(g, g)^{\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})} = e(g, g^{\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})}) \\
&= e(g, g^{\sum_{i=1}^{n} \lambda_{i1} \cdot p_i(\theta_{i1})} \cdot g^{\sum_{i=1}^{n} \lambda_{i2} \cdot p_i(\theta_{i2})} \cdot \ldots \cdot g^{\sum_{i=1}^{n} \lambda_{im} \cdot p_i(\theta_{im})}) \\
&= e(g, y^1 \cdot y^2 \cdot \ldots \cdot y^m) = e(g, y)
\end{aligned}$$

and

$$\begin{aligned}\prod_{i=1}^{n} \tau_i &= \tau^1 \cdot \ldots \cdot \tau^n = e(g, g^{\widetilde{x}_1}) \cdot e(g, g^{\widetilde{x}_2}) \cdot \ldots \cdot e(g, g^{\widetilde{x}_n}) \\ &= e(g,g)^{\widetilde{x}_1} \cdot e(g,g)^{\widetilde{x}_2} \cdot \ldots \cdot e(g,g)^{\widetilde{x}_n} = e(g,g)^{\sum_{i=1}^{n} \widetilde{x}_i} \qquad (1) \\ &= e(g, g^{\sum_{i=1}^{n} \widetilde{x}_i}) = e(g, g^{\widetilde{x}_1} \cdot g^{\widetilde{x}_2} \cdot \ldots \cdot g^{\widetilde{x}_n}) \\ &= e(g, x_1 \cdot x_2 \cdot \ldots \cdot x_n) = e(g, y)\end{aligned}$$

Combining the two results we obtain that $\prod_{i=1}^{n} \tau_i = \sigma$ and $\prod_{i=1}^{n} \tau_i = e(g, y)$ which imply that the algorithm **Verify** will output 1.

4.3 Verifiability of the Multiplicative VHSS

Theorem 2. *For any n secret inputs $x_1, x_2, \ldots, x_n \in \mathbb{F}$ and any set T of corrupted servers with $|T| \leqslant m$ in the multiplicative VHSS, it holds that any PPT adversary who modifies the shares of the secret inputs for any j such that $s_j \in T$, can cause a wrong value to be accepted as $x_1 \cdot x_2 \cdot \ldots \cdot x_n$ with negligible probability. It holds that*

$$\Pr[\boldsymbol{Exp}_{VHSS}^{Verif.}(x_1, x_2, \ldots, x_n, T, \mathcal{A}) = 1] \le \varepsilon.$$

Proof. Consider that $y' \neq f(x_1, x_2, \ldots, x_n)$ where $f(x_1, x_2, \ldots, x_n) = x_1 \cdot x_2 \cdot \ldots \cdot x_n = y$ and $\textbf{Verify}(\tau_1, \ldots, \tau_n, \sigma', y') = 1$.
Then:

$$\begin{aligned} &\textbf{Verify}(\tau_1, \ldots, \tau_n, \sigma', y') = 1 \\ \Longrightarrow\ &\prod_{i=1}^{n} \tau_i = \sigma' \text{ and } \prod_{i=1}^{n} \tau_i = e(g, y') \\ \Longrightarrow\ &\prod_{i=1}^{n} \tau_i = e(g, y') \ \text{ (see equation (1))} \\ \Longrightarrow\ &e(g, y) = e(g, y') \\ \Longrightarrow\ &e(g, g^{r_1}) = e(g, g^{r_2}) \ \text{ for some } \ r_1, r_2 \in \mathbb{F} \\ \Longrightarrow\ &e(g,g)^{r_1} = e(g,g)^{r_2} \ \text{ for some } \ r_1, r_2 \in \mathbb{F} \\ \Longrightarrow\ &r_1 = r_2 \ \text{ in } \ \mathbb{F} \\ \Longrightarrow\ &g^{r_1} = g^{r_2} \ \text{ in } \ \mathbb{G} \\ \Longrightarrow\ &y = y' \\ \Longrightarrow\ &f(x_1, x_2, \ldots, x_n) = y' \end{aligned}$$

which is a contradiction!

4.4 Security of the Multiplicative VHSS

Theorem 3. *For all $T \subset \{s_1, \ldots, s_m\}$ with $|T| \le m - 1$ and all PPT adversaries $\mathcal{A}$, the multiplicative VHSS scheme (**ShareSecret**, **PartialEval**, **PartialProof**, **FinalEval**, **FinalProof**, **Verify**) is $(m-1)$-secure. It holds that* $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \le \varepsilon(\lambda)$ *for some negligible $\varepsilon(\lambda)$.*

Proof. Let $|T| = m - 1$, that is, there are $m - 1$ corrupted servers.

Consider, without loss of generality, that the first $m-1$ servers are the corrupted ones. The adversary $\mathcal{A}$ has $(m-1)n$ shares from the corrupted servers and no additional information.

Then, if we denote the shares by $\hat{x}_{i1}, \ldots, \hat{x}_{im}$ with $i \in \{1, \ldots, n\}$, we know that $\prod_{j=1}^{m} \hat{x}_{ij} = \hat{x}_i$ for some i and it holds that:

$$\prod_{j=1}^{m-1} \hat{x}_{ij} \cdot \hat{x}_{im} = \hat{x}_i$$

$$\iff \hat{x}_{im} = (\prod_{j=1}^{m-1} \hat{x}_{ij})^{-1} \cdot \hat{x}_i$$

given the $(\prod_{j=1}^{m-1} \hat{x}_{ij})^{-1}$. Now, $\hat{x}_{im} \in \mathcal{Y}$ is just a value which gives nothing to the adversary regarding whether it is related to x_i or $x_i{}'$. Therefore, any PPT adversary has probability 1/2 to decide whether $\hat{x}_i$ is x_i or $x_i{}'$.

What is more, the adversary is also able to see the public encoded values $\tau_1 = e(g, g^{\widetilde{x}_1}), \ldots, \tau_n = e(g, g^{\widetilde{x}_n})$. It holds that the adversary cannot obtain neither $g^{\widetilde{x}_i}$ (Assumption 1) nor $\widetilde{x}_i$ (Observation 1) from τ_i. Therefore, the adversary gets no additional information from τ_i.

Finally, it holds that $\text{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

5 Conclusion

In this paper, we introduced the notion of *verifiable homomorphic secret sharing* (VHSS) for multi-input which is based on the general notion of homomorphic secret sharing (HSS). The VHSS scheme enables the clients (users) to locally generate shares of a proof which confirms that the combination of the shares is correct. We provided a detailed example for casting Shamir's secret sharing scheme [20] over a finite field $\mathbb{F}$ for the function f that sums n field elements. Such a scheme exists if and only if $m > n \cdot t$. Furthermore, we proposed an instantiation of the multiplicative verifiable homomorphic secret sharing (multiplicative VHSS) scheme for the function f that multiplies n elements under the hardness assumption of the fixed inversion problem in bilinear maps.

Acknowledgments. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and the VR grant PRECIS.

References

1. Ananth, P., Chandran, N., Goyal, V., Kanukurthi, B., Ostrovsky, R.: Achieving privacy in verifiable computation with multiple servers – without FHE and without pre-processing. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 149–166. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_9

2. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 863–874. ACM (2013)
3. Barkol, O., Ishai, Y., Weinreb, E.: On d-multiplicative secret sharing. J. cryptol. **23**(4), 580–593 (2010)
4. Benaloh, J.C.: Secret sharing homomorphisms: keeping shares of a secret secret (extended abstract). In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 251–260. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_19
5. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
6. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19
7. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1292–1303. ACM (2016)
8. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
9. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: LIPIcs-Leibniz International Proceedings in Informatics, vol. 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
10. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_26
11. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 844–855. ACM (2014)
12. Fiore, D., Mitrokotsa, A., Nizzardo, L., Pagnin, E.: Multi-key homomorphic authenticators. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 499–530. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_17
13. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25
14. Gentry, C.: A Fully Homomorphic Encryption Scheme. Stanford University, Stanford (2009)
15. Gordon, S.D., Katz, J., Liu, F.-H., Shi, E., Zhou, H.-S.: Multi-client verifiable computation with stronger security guarantees. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 144–168. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_6
16. Cheon, J.H., Lee, D.H.: A note on self-bilinear maps. Bull. Korean Math. Soc. **46**, 303–309 (2009)
17. Pagnin, E., Mitrokotsa, A., Tanaka, K.: Anonymous single-round server-aided verification of signatures. In: The 5th International Conference on Cryptology and Information Security (Latincrypt) (2017)

18. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: verifiable computation from attribute-based encryption. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_24
19. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Found. Secure Comput. **4**(11), 169–180 (1978)
20. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
21. Tang, C., Chen, Y.: Efficient non-interactive verifiable outsourced computation for arbitrary functions. IACR Cryptology ePrint Archive, 2014:439 (2014)
22. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: practical private queries on public data. In: NSDI, pp. 299–313 (2017)
23. Yoshida, M., Obana, S.: Verifiably multiplicative secret sharing. In: Shikata, J. (ed.) ICITS 2017. LNCS, vol. 10681, pp. 73–82. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72089-0_5