Sum It Up: Verifiable Additive Homomorphic Secret Sharing

Georgia Tsaloli(✉) and Aikaterini Mitrokotsa

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

Abstract. In many situations, clients (*e.g.,* researchers, companies, hospitals) need to outsource joint computations based on joint inputs to external cloud servers in order to provide useful results. Often clients want to guarantee that the results are *correct* and thus, an output that can be *publicly* verified is required. However, important security and privacy challenges are raised, since clients may hold sensitive information and the cloud servers can be untrusted. Our goal is to allow the clients to protect their secret data, while providing *public verifiability i.e.,* everyone should be able to verify the correctness of the computed result.

In this paper, we propose three concrete constructions of *verifiable additive homomorphic secret sharing* (VAHSS) to solve this problem. Our instantiations combine an *additive homomorphic secret sharing* (HSS) scheme, which relies on Shamir's secret sharing scheme over a finite field $\mathbb{F}$, for computing the sum of the clients' secret inputs, and three different methods for achieving *public verifiability.* More precisely, we employ: *(i)* homomorphic collision-resistant hash functions; *(ii)* linear homomorphic signatures; as well as *(iii)* a threshold RSA signature scheme. In all three cases we provide a detailed correctness, security and verifiability analysis and discuss their efficiency.

Keywords: Function secret sharing · Homomorphic secret sharing · Verifiable computation · Public verifiability

1 Introduction

The emergence of communication technologies is changing the way data are stored, processed and used. Data collected from multiple, often resource-constrained devices are stored and processed by remote, untrusted (cloud) servers and subsequently, used by third parties (*e.g.,* electricity companies, doctors, researchers). Furthermore, many applications involve joint computations on data collected from multiple clients (*e.g.,* compute statistics on electricity consumption via smart metering, measure emissions via environmental sensors or even e-voting systems). To avoid single points of failure, multiple servers can be recruited to perform joint computations for multiple clients. Although this distributed cloud-assisted environment is very attractive and has tremendous advantages, it is accompanied by serious *security* and *privacy* challenges.

J. H. Seo (Ed.): ICISC 2019, LNCS 11975, pp. 115–132, 2020.
https://doi.org/10.1007/978-3-030-40921-0_7

In such settings, it is often desirable to solve the cloud-assisted computing problem described by the following constraints: *(i)* n clients want to outsource their joint computations on their joint inputs to multiple servers; *(ii)* the clients want to keep their individual values secret; *(iii)* the servers are untrusted; *(iv)* the clients cannot communicate with each other; and *(v)* everyone should be able to verify the correctness of the computed result (*i.e., public verifiability*). Let us consider that n clients (as depicted in Fig. 1), with n individual secret inputs $x_1, x_2, \dots x_n$, want to outsource the joint computation of a function on their joint inputs $f(x_1, x_2, \dots, x_n)$. Tsaloli *et al.* [16] addressed the problem of computing the joint multiplications of n inputs corresponding to n clients and introduced the concept of *verifiable homomorphic secret sharing* (VHSS). More precisely, VHSS allows to split n secret inputs into m shares and perform the joint computation of a function $f(x_1, x_2, \dots, x_n) = y$, without any communication between the clients; while also providing a proof π that allows the *public verification* of the computed result, *i.e.,* having access to the pair (y, π) *anyone* can verify that the computed result is correct. However, the possibility to achieve verifiable homomorphic secret sharing for other functions has been left open.

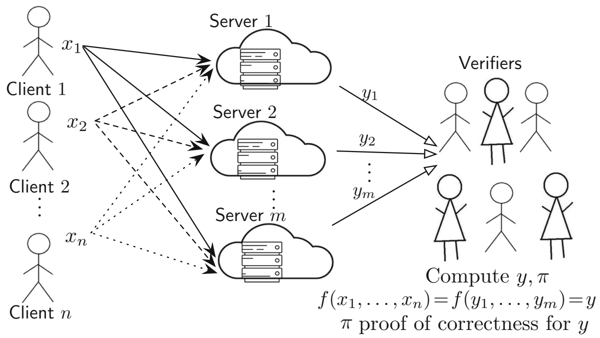

Fig. 1. n clients outsourcing the joint computation of their joint inputs to m servers.

In this paper, we revisit the concept of *verifiable homomorphic secret sharing* (VHSS) and we investigate whether it is possible to achieve *verifiable additive homomorphic secret sharing*. The answer is affirmative and we introduce three constructions that can be employed in order to compute securely and privately the joint addition of n inputs from n clients by employing m servers, while also providing *public verifiability*. These constructions can be useful, for instance, when statistics need to be computed about electricity consumption with data collected from multiple users, or when collecting data for remote monitoring and diagnosis from multiple patients, as well as when data from environmental sensors (*e.g.,* temperature, humidity) are collected from multiple sensors.

Our Contributions. We focus on the problem of outsourcing joint *additions*, while providing strong security and privacy guarantees when: *(i)* multiple clients outsource joint additions on their joint secret inputs; *(ii)* multiple untrusted servers are employed for the computation; and *(iii)* anyone can verify that the output of the computation is correct. We propose for the first time three different instantiations of *verifiable additive homomorphic secret sharing* (VAHSS).

We discriminate three different cases of VAHSS depending on the employed primitive (homomorphic hash functions, linearly homomorphic signatures and threshold signatures) as well as whether the partial proofs (used in order to check the correctness of the computed result) are computed by either the clients or the servers. Furthermore, we have modified the original VHSS definition in order to capture the different cases regarding the generation of the proofs; thus, allowing the employment of VHSS in multiple application settings.

Our constructions rely on casting Shamir's secret sharing scheme over a finite field $\mathbb{F}$ as an n-client, m-server, t-perfectly secure additive *homomorphic secret sharing* (HSS) for the function that sums n field elements. Such an additive HSS exists, if and only if $m > n{\cdot}t$. By employing the additive HSS in combination with homomorphic collision-resistant hash functions [13,17], we provide an instantiation, where the partial proofs are computed by the servers. Subsequently, we combine the additive HSS with a linearly homomorphic signature scheme [10], or a threshold RSA signature scheme [8] to obtain two different instantiations of VAHSS depending on whether the partial proofs are computed by the clients or by a subset of the servers correspondingly. In all three cases, we provide a detailed correctness, security and verifiability analysis.

1.1 Related Work

Homomorphic Secret Sharing. In threshold secret sharing schemes [15] a secret x is split into multiple shares (*e.g.,* $x_1, x_2, \ldots, x_m$) in such a way that by combining some subsets of the shares, it is possible to reconstruct the secret, while from smaller subsets of the shares, it is not possible to recover any information related to the secret. *Homomorphic secret sharing* (HSS) [7] can be seen as the secret sharing analogue of homomorphic encryption. More precisely, HSS allows the local evaluation of functions on shares on one or more secret inputs by relying on local computations on the shares of the secrets; while at the same time guaranteeing that the shares of the output are short. The first instance of additive HSS considered in the literature [3] is computed in some finite Abelian group. However, HSS does not provide any verifiability guarantees about the computed result.

Verifiable Function Secret Sharing. *Function secret sharing* (FSS) [5] can be seen as a natural generalization of distributed point functions (DPF) and provides a method for additively secret sharing a function f from a given function family $\mathcal{F}$. In FSS 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 it holds $f(x) = f_1(x) + \ldots + f_m(x)$. Boyle *et al.* introduced the concept of *verifiable* FSS (VFSS) [6], which provides interactive protocols to verify that keys $(k_1^*, \ldots, k_m^*)$, obtained from a potentially malicious user, are consistent with some $f \in \mathcal{F}$. However, Boyle *et al.*'s VFSS applies in the setting of one client and multiple servers. On the contrary, VHSS can be applied when multiple clients (multi-input) outsource the joint computation to multiple servers. In addition, VFSS focuses on verifying that the shares $f_1, \ldots, f_m$ are consistent with f; while VHSS generates a proof that guarantees that the final result is correct.

Publicly Auditable Secure Multi-party Computation. Outsourcing computations is inherently connected to secure multi-party computation (MPC) protocols. In MPC [4,11,12], the public verifiability is traditionally achieved by employing non-interactive zero-knowledge (NIZK) proofs. Baum *et al.* [1] introduced the notion of *publicly auditable* MPC protocols that are suitable for the multi-client and multi-server setting. Publicly auditable MPC can be seen as an extension of the classic formalization of secure function evaluation; it relies on the SPDZ protocol [11,12] and NIZK proofs, while it enhances each shared input x with a Pedersen commitment. Baum *et al.* [1] require correctness and privacy, when there is at least one honest party, while everyone having access to the transcript of the protocol (published in a bulletin board) can verify the correctness of the computed result. We should note that *publicly auditable* MPC protocols are very expressive regarding the class of functions being computed, but often require heavy computations. To formalize auditable MPC an extra *non-corruptible* party is introduced in the standard MPC model, namely the *auditor*. On the contrary, in VAHSS, no additional non-corruptible party is required, while we avoid the employment of expensive cryptographic operations and primitives such as NIZK.

Organization. The paper is organized as follows. In Sect. 2, we provide the modified definition of verifiable homomorphic secret sharing (VHSS). In Sect. 3, we introduce three verifiable additive homomorphic secret sharing (VAHSS) constructions using homomorphic hash functions, linearly homomorphic signatures and a threshold signature scheme respectively. In all three proposed instantiations, we provide the corresponding correctness, security and verifiability proofs. Finally, Sect. 4 summarizes the paper.

2 Preliminaries

Our concrete instantiations for the additive VHSS problem are based on the VHSS definition proposed in [16]. However, we propose a slightly modified version of the VHSS definition to capture cases when partial proofs (used to verify the correctness of the final result) are computed either from the clients or the servers. We added the **Setup** algorithm to allow the generation of keys and we modified the **PartialProof** algorithm accordingly to allow the different scenarios.

Definition 1 (Verifiable Homomorphic Secret Sharing (VHSS)). *An n-client, m-server, t-secure verifiable homomorphic secret sharing scheme for a* *function $f : \mathcal{X} \mapsto \mathcal{Y}$, is a 7-tuple of PPT algorithms (**Setup**, **ShareSecret**, **PartialEval**, **PartialProof**, **FinalEval**, **FinalProof**, **Verify**) which are defined as follows:*

- $(pp, sk) \leftarrow$ ***Setup***(1^λ)*: On input 1^λ, where λ is the security parameter, the algorithm outputs a secret key sk and some public parameters pp.*
- $(\mathsf{share}_{i1}, \ldots, \mathsf{share}_{im}, \tau_i) \leftarrow$ ***ShareSecret***$(1^\lambda, i, \boldsymbol{x}_i)$*: The algorithm takes as input 1^λ, $i \in \{1, \ldots, n\}$ which is the index for the client c_i and $\boldsymbol{x}_i$ which denotes a vector of one (*i.e., *$x_i \in \mathcal{X}$) or more secret values that belong to each client and should be split into shares. The algorithm outputs m shares share_{ij} (denoted also by $x_{ij} \in \mathcal{X}$ when $\boldsymbol{x}_i = x_i$) for each server s_j, as well as, if necessary, a publicly available value τ_i*[1] *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 $x_1, \ldots, x_n$ that the server s_j has, the algorithm **PartialEval** outputs $y_j \in \mathcal{Y}$.*
- $\sigma_k \leftarrow$***PartialProof***$(sk, pp, secret_{values}, k)$*: On input the secret key sk, public parameters pp, secret values (based on which the partial proofs are generated), denoted by $secret_{values}$; and the corresponding index k (where k is either i or j), a partial proof σ_k is computed.*
- $y \leftarrow$***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 compute, the algorithm **FinalEval** outputs y, the final result for $f(x_1, x_2, \ldots, x_n)$.*
- $\sigma \leftarrow$***FinalProof***$(pp, \sigma_1, \ldots, \sigma_{|k|})$*: On input public parameters pp and the partial proofs $\sigma_1, \sigma_2, \ldots, \sigma_{|k|}$, the algorithm **FinalProof** outputs σ which is the proof that y is the correct value.*
- $0/1 \leftarrow$***Verify***(pp, σ, y)*: On input the final result y, the proof σ, and, when needed, public parameters pp, the algorithm **Verify** outputs either 0 or 1.*

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

- **Correctness:** For any secret input $x_1, \ldots, x_n$, for all m-tuples in the set $\{(\mathsf{share}_{i1}, \ldots, \mathsf{share}_{im}), \tau_i\}_{i=1}^n$ coming from **ShareSecret**, for all $y_1, \ldots, y_m$ computed by **PartialEval**, $\sigma_1, \ldots, \sigma_{|k|}$ computed from **PartialProof**, and for y and σ generated by **FinalEval** and **FinalProof** respectively, the scheme should satisfy the following correctness requirement:

$$\Pr\left[\mathbf{Verify}(pp, \sigma, y) = 1\right] = 1.$$

- **Verifiability:** Let T be the set of corrupted servers with $|T| \leqslant m$. Denote by $\mathcal{A}$ any PPT adversary and consider n secret inputs $x_1, \ldots, x_n \in \mathbb{F}$. Any PPT adversary $\mathcal{A}$ who controls the shares of the secret inputs for any j such that $s_j \in T$, can cause a wrong value to be accepted as $f(x_1, x_2, \ldots, x_n)$ with negligible probability. We define the following experiment $\mathbf{Exp}_{\mathrm{VHSS}}^{\mathrm{Verif.}}(x_1, \ldots, x_n, T, \mathcal{A})$:

[1] τ_i, when computed, can be included in the list of public parameters pp.

1. For all $i \in \{1, \ldots, n\}$, generate $(\mathsf{share}_{i1}, \ldots, \mathsf{share}_{im}, \tau_i) \leftarrow$ **ShareSecret**$(1^\lambda, i, \boldsymbol{x}_i)$ and publish τ_i.
2. For all j such that $s_j \in T$, give $\begin{pmatrix} \mathsf{share}_{1j} \\ \mathsf{share}_{2j} \\ \vdots \\ \mathsf{share}_{nj} \end{pmatrix}$ to the adversary.
3. For the corrupted servers $s_j \in T$, the adversary $\mathcal{A}$ outputs modified shares $y_j{}'$ and $\sigma_k{}'$. Then, for j such that $s_j \notin T$, we set $y_j{}' =$ **PartialEval**$(j, (x_{1j}, \ldots, x_{nj}))$ and $\sigma_k{}' =$ **PartialProof**$(sk, pp, \text{secret}_{\text{values}}, k)$. Note that we consider modified $\sigma_k{}'$ only when computed by the servers.
4. Compute the modified final value $y' =$ **FinalEval**$(y_1{}', y_2{}', \ldots, y_m{}')$ and the modified final proof $\sigma' =$ **FinalProof**$(pp, \sigma'_1, \ldots, \sigma'_{|k|})$.
5. If $y' \neq f(x_1, x_2, \ldots, x_n)$ and **Verify**$(pp, \sigma', y') = 1$, then output 1 else 0.

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

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

- **Security:** Let T be the set of the corrupted servers with $|T| < m$. Consider the following semantic security challenge experiment:
  1. The adversary $\mathcal{A}_1$ gives $(i, x_i, x'_i) \leftarrow \mathcal{A}_1(1^\lambda)$ 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 $(\widehat{\mathsf{share}}_{i1}, \ldots, \widehat{\mathsf{share}}_{im}, \widehat{\tau}_i) \leftarrow$ **ShareSecret**$(1^\lambda, i, \hat{\boldsymbol{x}}_i)$ where the secret input $\hat{\boldsymbol{x}}_i = \begin{cases} x_i, \text{if } b = 0 \\ x'_i, \text{otherwise} \end{cases}$.
  3. Given the shares from the corrupted servers T and $\widehat{\tau}_i$, the adversary distinguisher outputs a guess $b' \leftarrow \mathcal{D}((\widehat{\mathsf{share}}_{ij})_{j|s_j \in T}, \widehat{\tau}_i)$.

  Let $\text{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ be the advantage of $\mathcal{A} = \{\mathcal{A}_1, \mathcal{D}\}$ in guessing b in the above experiment, where the probability is taken over the randomness of the challenger and of $\mathcal{A}$. A 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 $\text{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

In our solution, we employ a simple variant of the (Strong) RSA based signature introduced by Catalano *et al.* [9], which can be seen as a linearly homomorphic signature scheme on $\mathbb{Z}_N$.

Definition 2 (Linearly Homomorphic Signature [10]**).** *A linearly homomorphic signature scheme is a tuple of PPT algorithms (**HKeyGen, HSign, HVerify, HEval**) defined as follows:*

– ***HKeyGen***$(1^\lambda, k)$ *takes as input the security parameter λ and an upper bound k for the number of messages that can be signed in each dataset. It outputs a secret signing key sk and a public key vk. The public key defines a message space M, a signature space $\mathcal{S}$, and a set $\mathcal{F}$ of admissible linear functions such that any $f : \mathcal{M}^n \mapsto \mathcal{M}$ is linear.*

– ***HSign***(sk, fid, m_i, i) *algorithm takes as input the secret key* sk*, a dataset identifier* fid*, and the* i*-th message* m_i *to be signed, and outputs a signature* σ_i*.*
– ***HVerify***(vk, fid, m, σ, f) *algorithm takes as input the verification key* vk*, a dataset identifier* fid*, a message* m*, a signature* σ *and a function* f*. It outputs either 1 if the signature corresponds to the message* m *or 0 otherwise.*
– ***HEval***$(vk, fid, f, \sigma_1, \ldots, \sigma_n)$ *algorithm takes as input the verification key* vk*, a dataset identifier* fid*, a function* $f \in \mathcal{F}$*, and a tuple of signatures* $\sigma_1, \ldots, \sigma_n$*. It outputs a new signature* σ*.*

We use homomorphic hash functions in order to achieve verifiability. Below, we provide the definition of such a function. More precisely, we employ a homomorphic hash function satisfying additive homomorphism [13].

Definition 3 (Homomorphic Hash Function [17]). *A homomorphic hash function* $h : \mathbb{F}_N \mapsto \mathbb{G}_q$*, where* $\mathbb{F}$ *is a finite field and* $\mathbb{G}$ *is a multiplicative group of prime order* q*, is defined as a collision-resistant hash function satisfying the homomorphism in addition to the properties of a universal hash function* $uh : (0,1)^* \mapsto (0,1)^l$*.*

1. *One-way: It is computationally hard to compute* $h^{-1}(x)$*.*
2. *Collision-free: It is computationally hard to find* $x, y \in \mathbb{F}^N (x \neq y)$ *such that* $h(x) = h(y)$*.*
3. *Homomorphism: For any* $x, y \in \mathbb{F}^N$*, it holds* $h(x \circ y) = h(x) \circ h(y)$ *where "*$\circ$*" is either "+" or "·".*

For completeness, we also provide the definition of a secure pseudorandom function PRF.

Definition 4 (Pseudorandom Function (PRF)). *Let* S *be a distribution over* $\{0,1\}^\ell$ *and* $F_s : \{0,1\}^m \rightarrow \{0,1\}^n$ *be a family of functions indexed by strings* s *in the support of* S*. We say* $\{F_s\}$ *is a pseudorandom function family if for every PPT adversary* D*, there exists a negligible function* ϵ *such that:*

$$|\Pr[D^{F_s}(\cdot) = 1] - \Pr[D^R(\cdot) = 1]| \leq \epsilon,$$

where s *is distributed according to* S*, and* R *is a function sampled uniformly at random from the set of all functions from* $\{0,1\}^m$ *to* $\{0,1\}^n$*.*

3 Verifiable Additive Homomorphic Secret Sharing

In this section, we present three different instantiations to achieve *verifiable additive homomorphic secret sharing* (VAHSS). More precisely, we consider n clients with their secret values $x_1, \ldots, x_n$ respectively, and m servers $s_1, \ldots, s_m$ that perform computations on shares of these secret values. Firstly, the clients split their secret values into shares, that reveal nothing about the secret value itself and then, they distribute the shares to each of the m servers. Each server performs some calculations in order to publish a value, which is related to the final result $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$. Then, depending on the instantiation proposed, partial proofs are generated in a different way. The partial proofs are values such that their combination results in a final proof, which confirms the correctness of the final computed value $f(x_1, \ldots, x_n)$.

3.1 Construction of VAHSS Using Homomorphic Hash Functions

In this section, we aim to compute the function value y, which corresponds to $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ as well as a proof σ that y is correct. We combine an additive HSS for the algorithms related to the value y and hash functions for the generation of the proof σ. Let $c_1, \ldots, c_n$ denote n clients and $x_1, \ldots, x_n$ their corresponding secret inputs. Let, for any $\{i\}_{i=1,\ldots,n}$, $\theta_{i1}, \ldots, \theta_{im}$ be distinct non-zero field elements and $\lambda_{i1}, \ldots, \lambda_{im}$ be field elements ("Lagrange coefficients") such that for any univariate polynomial p_i of degree t over a finite field $\mathbb{F} = \mathbb{F}_N$ we have:

$$p_i(0) = \sum_{j=1}^{m} \lambda_{ij} p_i(\theta_{ij}) \tag{1}$$

Each client c_i generates shares of the secret x_i, denoted by $x_{i1}, \ldots, x_{im}$ respectively, and gives the share x_{ij} to each server s_j. The servers, in turn, compute a partial sum, denoted by y_j, and publish it. Anyone can then compute $y = y_1 + \ldots + y_m$, which corresponds to the function value $y = f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$. We suggest that every client c_i uses a homomorphic collision-resistant function $H : x \mapsto g^x$ proposed by Krohn *et al.* [13] to generate a public value τ_i which reveals nothing about x_i (under the discrete logarithm assumption). Then, the servers compute values $\sigma_1, \ldots, \sigma_m$ which will be appropriately combined so that they give the proof σ that we are interested in. The value y comes from the combination of partial values y_j, which are computed by the m servers. More precisely, our solution is composed of the following algorithms:

1. **ShareSecret**$(1^\lambda, i, x_i, file_i)$**:** For elements $\{a_i\}_{i\in\{1,\ldots,t\}} \in \mathbb{F}$ selected uniformly at random, pick a t-degree polynomial p_i of the form $p_i(X) = x_i + a_1X + a_2X^2 + \ldots + a_tX^t$ with $t \cdot n < m$. Notice that the free coefficient of p_i is the secret input x_i. Let $H : x \mapsto g^x$ (with g a generator of the multiplicative group of $\mathbb{F}$) be a collision-resistant homomorphic hash function [17]. Let R_i be the output of a pseudorandom function (PRF) $F : \{0,1\}^{l_1} \times \{0,1\}^{l_2} \mapsto \mathbb{F}$ where $R_i = F_k(i, file_i)$ for a key $k \in \{0,1\}^{l_1}$ given to the clients and an input $file_i$ associated with client i such that $(i, file_i) \in \{0,1\}^{l_2}$. For $i = n$ we require $\mathbb{F} \ni R_n = \phi(N)\lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil - \sum_{i=1}^{n-1} R_i$. Then, compute $\tau_i = H(x_i + R_i)$, define $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$ (given thanks to the Eq. (1)) and output $(x_{i1}, x_{i2}, \ldots, x_{im}, \tau_i) = (\lambda_{i1} \cdot p_i(\theta_{i1}), \ldots, \lambda_{im} \cdot p_i(\theta_{im}), H(x_i + R_i))$.
2. **PartialEval**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$**:** Given the j-th shares of the secret inputs, compute the sum of all $x_{ij} = \lambda_{ij} \cdot p_i(\theta_{ij})$ for the given j and $i \in [n]$. Output y_j with $y_j = \lambda_{1j} \cdot p_1(\theta_{1j}) + \ldots + \lambda_{nj} \cdot p_n(\theta_{nj}) = \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})$.

3. **PartialProof**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$**:** Given the j-th shares of the secret inputs, compute and output the partial proof $\sigma_j = g^{\sum_{i=1}^{n} x_{ij}} = g^{y_j} = H(y_j)$.
4. **FinalEval**$(y_1, y_2, \ldots, y_m)$**:** Add the partial sums $y_1, \ldots, y_m$ together and output y (where $y = y_1 + \ldots + y_m$).
5. **FinalProof**$(\sigma_1, \ldots, \sigma_m)$**:** Given the partial proofs $\sigma_1, \sigma_2, \ldots, \sigma_m$, compute the final proof $\sigma = \prod_{j=1}^{m} \sigma_j$. Output σ.
6. **Verify**$(\tau_1, \ldots, \tau_n, \sigma, y)$**:** Check whether $\sigma = \prod_{i=1}^{n} \tau_i \wedge \prod_{i=1}^{n} \tau_i = H(y)$ holds. Output 1 if the check is satisfied or 0 otherwise.

Each client runs the **ShareSecret** algorithm to compute and distribute the shares of x_i to each of the m servers and a public value τ_i, which is needed for the verification. Then, each server s_j has the shares given from the n clients and runs the **PartialEval** algorithm to output the public values y_j related to the final function value. Furthermore, each server runs the **PartialProof** algorithm and produces the value σ_j. Finally, any user or verifier is able to run the **FinalEval** algorithm to get y and the **FinalProof** algorithm to get the proof σ. Lastly, **Verify** algorithm ensures that y and σ match and thus, $y = f(x_1, \ldots, x_n)$ is correct. Our construction is illustrated in the Table 1.

Table 1. VAHSS using homomorphic hash functions

Secret inputs (held by the clients)	Servers				Public values
	s_1	s_2	$\cdots$	s_m	
x_1	x_{11}	x_{12}	$\cdots$	x_{1m}	τ_1
x_2	x_{21}	x_{22}	$\cdots$	x_{2m}	τ_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_n	x_{n1}	x_{n2}	$\cdots$	x_{nm}	τ_n
Partial sums	y_1	y_2	$\cdots$	y_m	Total Sum: y
Partial proofs	σ_1	σ_2	$\cdots$	σ_m	Final Proof: σ

- **Correctness:** To prove the correctness of this construction, we need to prove that $\Pr\left[\textbf{Verify}(\tau_1, \ldots, \tau_n, \sigma, y) = 1\right] = 1$. By construction it holds that:

$$y = \sum_{j=1}^{m} y_j = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_{ij} \cdot p_i(\theta_{ij}) = \sum_{i=1}^{n} p_i(0) = \sum_{i=1}^{n} x_i \tag{2}$$

  Additionally, by construction, we have:

$$\sigma = \prod_{j=1}^{m} \sigma_j = \prod_{j=1}^{m} H(y_j) = \prod_{j=1}^{m} g^{y_j} = g^{\sum_{j=1}^{m} y_j} = g^y = H(y)$$

$$\text{and} \quad \prod_{i=1}^{n} \tau_i = \prod_{i=1}^{n} g^{x_i+R_i} = g^{\sum_{i=1}^{n} x_i} g^{\sum_{i=1}^{n} R_i} = g^{\sum_{i=1}^{n} x_i} g^{\sum_{i=1}^{n-1} R_i + R_n}$$
$$= g^{\sum_{i=1}^{n} x_i} g^{\phi(N) \lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil} = g^{\sum_{i=1}^{n} x_i} = g^{x_1+\ldots+x_n} \tag{3}$$
$$\stackrel{see\ eq.(2)}{=} g^y = H(y)$$

Combining the last two results we get that $\sigma = \prod_{i=1}^{n} \tau_i \wedge \prod_{i=1}^{n} \tau_i = H(y)$ holds. Therefore, the algorithm **Verify** outputs 1 with probability 1.

- **Security:** See [2] for a proof that the selected hash function H of our construction is a secure collision-resistant hash function under the discrete logarithm assumption.
  We will now prove that $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

Proof. ***Game 0:*** Consider $m-1$ corrupted servers. Then, $|T| = m-1$. Without loss of generality, let the first $m-1$ servers be the corrupted ones. Therefore, the adversary $\mathcal{A}$ has $(m-1)n$ shares from the corrupted servers and no additional information.

For any fixed i with $i \in \{1, \ldots, n\}$, it holds that $\sum_{j=1}^{m} \widehat{\mathsf{share}}_{ij} = \hat{x}_i$ and hence:

$$\sum_{j=1}^{m-1} \widehat{\mathsf{share}}_{ij} + \widehat{\mathsf{share}}_{im} = \hat{x}_i \iff \widehat{\mathsf{share}}_{im} = \hat{x}_i - \sum_{j=1}^{m-1} \widehat{\mathsf{share}}_{ij}$$

The adversary holds $\sum_{j=1}^{m-1} \widehat{\mathsf{share}}_{ij}$. Furthermore, the adversary holds the public value $\widehat{\tau_i} = g^{\hat{x}_i + R_i}$. Since R_i is the output of a PRF then $\widehat{\tau_i}$ is also a pseudorandom value.

Game 1: Consider that the adversary holds the same shares $\sum_{j=1}^{m-1} \widehat{\mathsf{share}}_{ij}$ and $\widehat{\tau_i}$ is now a truly random value.

Firstly, $\widehat{\mathsf{share}}_{im} \in \mathcal{Y}$ is just a value, which implies nothing to the adversary regarding whether it is related to x_i or ${x_i}'$. Moreover, ***Game 0*** and ***Game 1*** are computationally indistinguishable due to the security of the PRF. Thus, any PPT adversary has probability 1/2 to decide whether $\hat{x}_i$ is x_i or ${x_i}'$ and so, $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

- **Verifiability:** In this construction, for $y = x_1 + x_2 + \ldots + x_n$, if $y' \neq x_1 + \ldots + x_n$ and $\mathbf{Verify}(\tau_1, \ldots, \tau_n, \sigma', y') = 1$, then the verifiability follows:

$$\mathbf{Verify}(\tau_1, \ldots, \tau_n, \sigma', y') = 1 \Rightarrow \sigma' = \prod_{i=1}^{n} \tau_i \wedge \prod_{i=1}^{n} \tau_i = H(y')$$
$$\Rightarrow \prod_{i=1}^{n} \tau_i = H(y') \quad (\text{see Eq. 3}) \Rightarrow H(y) = H(y')$$

which is a contradiction since $y \neq y'$ and H is collision-resistant. Therefore,

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

3.2 Construction of VAHSS with Linear Homomorphic Signatures

Our goal is always to compute $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n = y$ as well as a proof σ that y is correct. We compute y using additive HSS and we employ a linearly homomorphic signature scheme, presented in [10] as a simple variant of Catalano *et al.'s* [9] signature scheme, for the generation of the proof. All clients hold the same signing and verification key. This could be the case if the clients are sensors of a company collecting information (*e.g., temperature, humidity*) useful for some calculations. Since the sensors/clients belong to the same company, sharing the same key might be necessary to facilitate configuration. In applications scenarios where clients should be set up with different keys, a multi-key scheme [14] could be used. However, in our construction, the clients can use the same signing key to sign their own secret value. In fact, they sign $x_{i,R}$ where $x_{i,R} = x_i + R_i$ with R_i chosen from each client as described in the Sect. 3.1. The signatures, denoted by $\sigma_1, \ldots, \sigma_n$ are public and combined they form a final signature σ, which verifies the correctness of y. Our instantiation constitutes of the following algorithms:

1. **Setup**$(1^k, N)$**:** Let N be the product of two safe primes each one of length $k'/2$. This algorithm chooses two random (safe) primes $\hat{p}, \hat{q}$ each one of length $k/2$ such that $\gcd(N, \phi(\hat{N})) = 1$ with $\hat{N} = \hat{p} \cdot \hat{q}$. Subsequently, the algorithm chooses $g, g_1, h_1, \ldots, h_n$ in $\mathbb{Z}^*_{\hat{N}}$ at random. Then, it chooses some (efficiently computable) injective function $H : \{0,1\}^* \mapsto \{0,1\}^l$ with $l < k'/2$. It outputs the public key $vk = (N, H, \hat{N}, g, g_1, h_1, \ldots, h_n)$ to be used by any verifier; and the secret key $sk = (\hat{p}, \hat{q})$ to be used for signing the secret values.
2. **ShareSecret**$(1^\lambda, i, x_i)$**:** For elements $\{a_i\}_{i \in \{1,\ldots,t\}} \in \mathbb{F}$ selected uniformly at random, pick a t-degree polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + \ldots + a_t X^t$ with $t \cdot n < m$. Notice that the free coefficient of p_i is the secret input x_i. Then, define $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$ (given using the equation (1)) and output $(x_{i1}, x_{i2}, \ldots, x_{im}) = \lambda_{i1} \cdot p_i(\theta_{i1}), \lambda_{i2} \cdot p_i(\theta_{i2}), \ldots, \lambda_{im} \cdot p_i(\theta_{im}))$.
3. **PartialEval**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$**:** Given the j-th shares of the secret inputs, compute the sum of all $x_{ij} = \lambda_{ij} \cdot p_i(\theta_{ij})$ for the given j and $i \in [n]$. Output y_j with $y_j = \lambda_{1j} \cdot p_1(\theta_{1j}) + \ldots + \lambda_{nj} \cdot p_n(\theta_{nj}) = \sum_{i=1}^n \lambda_{ij} \cdot p_i(\theta_{ij})$.
4. **PartialProof**$(sk, vk, fid, x_{i,R}, i)$**:** Parse the verification key vk to get $N, H, \hat{N}$, g, g_1 and $h_1, \ldots, h_n$. For the (efficiently computable) injective function H that is chosen from **Setup**, map fid to a prime: $H(fid) \mapsto e$. We denote the i-th vector of the canonical basis on $\mathbb{Z}^n$ by e_i. Choose random elements s_i and solve, using the knowledge for $\hat{p}$ and $\hat{q}$, the equation: $x^{eN} = g^{s_i} \prod_{j=1}^n {h_j}^{{f_j}^{(i)}} g_1^{x_{i,R}} \mod \hat{N}$ where ${f_j}^{(i)}$ denotes the j-th coordinate of the vector $f^{(i)}$. Notice that for our function e_i, the equation becomes $x^{eN} = g^{s_i} h_i g_1^{x_{i,R}} \mod \hat{N}$. Set $\widetilde{x_i} = x$. Output σ_i, where $\sigma_i = (e, s_i, fid, \widetilde{x_i})$ is the signature for x_i w.r.t. the function $f^{(i)} = e_i$.
5. **FinalEval**$(y_1, y_2, \ldots, y_m)$**:** Add the partial sums $y_1, \ldots, y_m$ together and output y (where $y = y_1 + \ldots + y_m$).
6. **FinalProof**$(vk, \hat{f}, \sigma_1, \sigma_2, \ldots, \sigma_n)$**:** Given the public verification key vk, the signatures $\sigma_1, \ldots, \sigma_n$, let $\hat{f} = (\alpha_1, \ldots, \alpha_n)$. Define $f' = (\sum_{i=1}^n \alpha_i f^{(i)} - f)/eN$ where $f = \sum_{i=1}^n \alpha_i f^{(i)} \mod eN$. Set $s = \sum_{i=1}^n \alpha_i s_i \mod eN$,

$s' = (\sum_{i=1}^{n} \alpha_i s_i - s)/eN$ and $\widetilde{x} = \frac{\prod_{i=1}^{n} \widetilde{x_i}^{\alpha_i}}{g^{s'} \prod_{j=1}^{n} h_j^{f'_j}} \mod \hat{N}$. For $\hat{f} = (1, \ldots, 1)$, compute $\widetilde{x} = \frac{\prod_{i=1}^{n} \widetilde{x_i}}{g^{s'} \prod_{j=1}^{n} h_j^{f'_j}} \mod \hat{N}$. Output σ where $\sigma = (e, s, fid, \widetilde{x})$.

7. **Verify**(vk, f, σ, y)**:** Compute $e = H(fid)$. Check that $y, s \in \mathbb{Z}_{eN}$ and $\widetilde{x}^{eN} = g^s \prod_{j=1}^{n} h_j^{f_j} g_1^y$ holds. Output: 1 if all checks are satisfied or 0 otherwise.

All n clients get the secret key sk from **Setup** and hold their secret value $x_1, \ldots, x_n$ respectively. Each client runs **ShareSecret** to split its secret value x_i into m shares and **PartialProof** to produce the partial signature (for the secret x_i) σ_i. The values σ_i's are not generated by the servers; since in that case, malicious compromised servers would not be detected. Then, each client distributes the shares to each of the m servers and publishes σ_i. Each server s_j computes and publishes the partial function value y_j by running **PartialEval**. Any verifier is able to get the function value $y = f(x_1, \ldots, x_n)$ from the **FinalEval** and the proof σ from the **FinalProof**. The **Verify** algorithm outputs 1 if and only if $y = x_1 + \ldots + x_n$. An illustration of our solution is reported in the Table 2.

Table 2. VAHSS using linear homomorphic signatures

Secret inputs (held by the clients)	Servers				Public values
	s_1	s_2	$\cdots$	s_m	vk
$x_1,\ sk$	x_{11}	x_{12}	$\cdots$	x_{1m}	σ_1
$x_2,\ sk$	x_{21}	x_{22}	$\cdots$	x_{2m}	σ_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_n,\ sk$	x_{n1}	x_{n2}	$\cdots$	x_{nm}	σ_n
Partial sums (public)	y_1	y_2	$\cdots$	y_m	Final proof (public)
Total sum (public)	y				σ

- **Correctness:** To prove the correctness of our construction we need to prove that $\Pr\left[\mathbf{Verify}(vk, f, \sigma, y) = 1\right] = 1$. It holds that:

$$\begin{aligned}
\widetilde{x}^{eN} &= \Big(\frac{\prod_{i=1}^{n} \widetilde{x_i}}{g^{s'} \prod_{i=1}^{n} h_j^{f'_j}}\Big)^{eN} = \frac{\prod_{i=1}^{n} \widetilde{x_i}^{eN}}{g^{s' eN} \prod_{i=1}^{n} h_j^{f'_j eN}} = \frac{\prod_{i=1}^{n} (g^{s_i} \prod_{j=1}^{n} h_j^{f_j^{(i)}} g_1^{x_{i,R}})}{g^{s' eN} \prod_{i=1}^{n} h_j^{f'_j eN}} \\
&= \frac{g^{\sum_{i=1}^{n} s_i}}{g^{s' eN}} \cdot \frac{\prod_{i=1}^{n} \prod_{j=1}^{n} h_j^{f_j^{(i)}}}{\prod_{i=1}^{n} h_j^{f'_j eN}} \cdot g_1^{\sum_{i=1}^{n} x_{i,R}} \\
&= \frac{g^{\sum_{i=1}^{n} s_i}}{g^{s' eN}} \cdot \frac{\prod_{i=1}^{n} \prod_{j=1}^{n} h_j^{f_j^{(i)}}}{\prod_{i=1}^{n} h_j^{f'_j eN}} \cdot g_1^{\sum_{i=1}^{n} x_i} \cdot g_1^{\sum_{i=1}^{n} R_i} \\
&\overset{see\ eq.(3)}{=} g^{\sum_{i=1}^{n} s_i - s' eN} \prod_{j=1}^{n} h_j^{\sum_{i=1}^{n} f_j^{(i)} - f'_j eN} g_1^{\sum_{i=1}^{n} x_i} = g^s \prod_{j=1}^{n} h_j^{f_j} g_1^{\sum_{i=1}^{n} x_i}
\end{aligned} \tag{4}$$

Thanks to the equation (2), it also holds that $y = \sum_{i=1}^{n} x_i$. Then $\widetilde{x}^{eN} = g^s \cdot \prod_{j=1}^{n} {h_j}^{f_j} \cdot {g_1}^y$ and thus, **Verify**$(vk, \sigma, y, f) = 1$ with probability 1.

- **Security:** The security of the signatures results easily from the original signature scheme proposed by Catalano *et al.* [9]. Moreover, $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$ as we have proven in the Sect. 3.1. We should note that, since in this construction no τ_i values are incorporated, the arguments related to the pseudorandomness of τ_i are not necessary.
- **Verifiability:** Verifiability is by construction straightforward since the final signature $\sigma \leftarrow \mathbf{FinalProof}(vk, \hat{f}, \sigma_1, \ldots, \sigma_n)$ is obtained using the correctly computed (by the clients) $\sigma_1, \ldots, \sigma_n$ and thus, $\sigma' = \sigma$ in this case. Therefore, if $y' \neq x_1 + \ldots + x_n$ while $y = x_1 + \ldots + x_n$ and $\mathbf{Verify}(vk, \sigma', y', f) = 1$ then:

$$\mathbf{Verify}(vk, \sigma', y', f) = 1 \Rightarrow \mathbf{Verify}(vk, \sigma, y', f) = 1$$

$$\Rightarrow \widetilde{x}^{eN} = g^s \prod_{j=1}^{n} {h_j}^{f_j} g_1^{y'} \text{(see equation (4))}$$

$$\Rightarrow g^s \prod_{j=1}^{n} {h_j}^{f_j} g_1^{\sum_{i=1}^{n} x_i} = g^s \prod_{j=1}^{n} {h_j}^{f_j} g_1^{y'} \Rightarrow \sum_{i=1}^{n} x_i = y'$$

which is a contradiction!
Therefore, $\Pr[\mathbf{Exp}_{\mathrm{VHSS}}^{\mathrm{Verif.}}(x_1, \ldots, x_n, T, \mathcal{A}) = 1] \leq \varepsilon$.

3.3 Construction of VAHSS with Threshold Signature Sharing

We propose a scheme where the clients generate and distribute shares of their secret values to the m servers and the servers mutually produce shares of the final value y similarly to the previous constructions. However, in order to generate the proof σ that confirms the correctness of y, our scheme employs the (t, n)-threshold RSA signature scheme proposed in [8] so that a signature σ is successfully generated even if $\mathsf{t} - 1$ servers are corrupted. Our proposed scheme (illustrated in the Table 3) acts in accordance with the following algorithms:

1. **Setup** $(1^k, N)$**:** Let $N = p \cdot q$ be the RSA modulus such that $p = 2p' + 1$ and $q = q' + 1$, where p', q' are large primes. Choose the public RSA key e_i such that $e_i \gg \binom{n}{\mathsf{t}}$ and then, pick the private RSA key d_i so that $e_i d_i \equiv 1 \mod (p'q')$. Output the public key e_i and the private key d_i.
2. **ShareSecret**$(1^\lambda, i, x_i, d_i, file_i)$**:** For elements $\{a_i\}_{i \in \{1,\ldots,t\}} \in \mathbb{F}$ selected uniformly at random, pick a t-degree polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + \ldots + a_t X^t$ with $t \cdot n < m$. Notice that the free coefficient of p_i is the secret input x_i. Then, define $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$ (given thanks to the Eq. (1)). Let $\mathcal{A}_i$ be an $m \times \mathsf{t}$ full-rank public matrix with elements from $\mathbb{F} = {\mathbb{Z}_r}^*$ for a prime r. Let $\boldsymbol{d} = (d_i, r_2, \ldots, r_\mathsf{t})^\intercal$ be a secret vector from $\mathbb{F}^\mathsf{t}$, where d_i is the private RSA key and $r_2, \ldots, r_\mathsf{t} \in \mathbb{F}$ are randomly chosen. Let a_{ij} be the entry at the i-th row and j-th column of the matrix $\mathcal{A}_i$. For all $j \in [m]$, set $\omega_{ij} = \mathsf{a}_{j1} d_i + \mathsf{a}_{j2} r_2 + \ldots + \mathsf{a}_{j\mathsf{t}} r_\mathsf{t} \in \mathbb{F}$ to be the share generated from the client c_i for the server s_j. It is now formed an $m \times \mathsf{t}$ system $\mathcal{A}_i \boldsymbol{d} = \boldsymbol{\omega_i}$.

Let $H : x_i \mapsto g^{x_i}$ (with g a generator of the multiplicative group of $\mathbb{F}$) be a collision-resistant homomorphic hash function [17]. Let R_i be randomly selected values as described in the Sect. 3.1. Output the public matrix $\mathcal{A}_i$, the (x_i's) shares $(x_{i1}, x_{i2}, \ldots, x_{im}) = \lambda_{i1} \cdot p_i(\theta_{i1}), \lambda_{i2} \cdot p_i(\theta_{i2}), \ldots, \lambda_{im} \cdot p_i(\theta_{im}))$, the shares of the private key $\boldsymbol{\omega_i} = (\omega_{i1}, \ldots, \omega_{im})$ and $H(x_i + R_i)$.

3. **PartialEval**$(j, (x_{1j}, x_{2j}, \ldots, x_{nj}))$**:** Given the j-th shares of the secret inputs, compute the sum of all $x_{ij} = \lambda_{ij} \cdot p_i(\theta_{ij})$ for the given j and $i \in [n]$. Output y_j with $y_j = \lambda_{1j} \cdot p_1(\theta_{1j}) + \ldots + \lambda_{nj} \cdot p_n(\theta_{nj}) = \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})$.
4. **PartialProof**$(\boldsymbol{\omega_1}, \ldots, \boldsymbol{\omega_n}, H(x_1 + R_1), \ldots, H(x_n + R_n), \mathcal{A}_1, \ldots, \mathcal{A}_n, N)$**:** For all $i \in [n]$ run the algorithm **PartialProof**$_i(\boldsymbol{\omega_i}, H(x_i + R_i), \mathcal{A}_i, i, N)$ where:

 PartialProof$_i(\boldsymbol{\omega_i}, H(x_i + R_i), \mathcal{A}_i, i, N)$**:** Let $S = \{s_1, s_2, \ldots, s_{\mathfrak{t}}\}$ be the coalition of $\mathfrak{t}$ servers ($\mathfrak{t} < m$) (w.l.o.g. take the first $\mathfrak{t}$), forming the system $\mathcal{A}_{iS}\boldsymbol{d} = \boldsymbol{\omega_{iS}}$. Let the $\mathfrak{t} \times \mathfrak{t}$ adjugate matrix of $\mathcal{A}_{iS}$ be:

$$\mathcal{C}_{iS} = \begin{bmatrix} c_{11} & c_{21} & \ldots & c_{\mathfrak{t}1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1\mathfrak{t}} & c_{2\mathfrak{t}} & \ldots & c_{\mathfrak{t}\mathfrak{t}} \end{bmatrix}$$

 Denote the determinant of $\mathcal{A}_{iS}$ by Δ_{iS}. It holds that:

$$\mathcal{A}_{iS}\mathcal{C}_{iS} = \mathcal{C}_{iS}\mathcal{A}_{iS} = \Delta_{iS}\mathbb{I}_{\mathfrak{t}} \tag{5}$$

 where $\mathbb{I}_{\mathfrak{t}}$ stands for the $\mathfrak{t} \times \mathfrak{t}$ identity matrix. Compute the partial signature of x_i: $\sigma_{ij} = H(x_i + R_i)^{2c_{j1}\omega_{ij}} \mod N$. Output $\boldsymbol{\sigma_i} = (\sigma_{i1}, \ldots, \sigma_{i\mathfrak{t}})$.

 PartialProof outputs $\boldsymbol{\sigma_1}, \ldots, \boldsymbol{\sigma_n}$.
5. **FinalEval**$(y_1, y_2, \ldots, y_m)$**:** Add the partial sums $y_1, \ldots, y_m$ together and output y (where $y = y_1 + \ldots + y_m$).
6. **FinalProof**$(e_1, \ldots, e_n, H(x_1 + R_1), \ldots, H(x_n + R_n), \boldsymbol{\sigma_1}, \ldots, \boldsymbol{\sigma_n}, N)$**:** For all $i \in \{1, \ldots, n\}$ run the algorithm **FinalProof**$_i(e_i, H(x_i + R_i), \boldsymbol{\sigma_i}, N)$ where:

 FinalProof$_i(e_i, H(x_i + R_i), \boldsymbol{\sigma_i}, N)$**:** Combine the partial signatures by computing $\overline{\sigma_i} = \prod_{j \in S} \sigma_{ij} \mod N$. Compute $\sigma_i = \overline{\sigma_i}^{\alpha_i} H(x_i + R_i)^{\beta_i} \mod N$ with α_i, β_i integers such that

$$2\Delta_{iS}\alpha_i + e_i\beta_i = 1. \tag{6}$$

 Output σ_i, *i.e.,* the signature that corresponds to the secret x_i.

 FinalProof outputs $\sigma = \prod_{i=1}^{n} {\sigma_i}^{e_i}$.
7. **Verify**$(H(x_1 + R_1), \ldots, H(x_n + R_n), \sigma, y)$**:** Check if $\sigma = \prod_{i=1}^{n} H(x_i + R_i) \wedge H(y) = \prod_{i=1}^{n} H(x_i + R_i)$ holds. Output 1 if the check is satisfied or 0 otherwise.

After the initialization with the **Setup**, each client c_i gets its public and private RSA keys, e_i and d_i respectively. Then, each c_i runs **ShareSecret** to compute and distribute the shares of x_i to each of the m servers, and form a public matrix

$\mathcal{A}_i$, shares of the private key $(\omega_{i1}, \ldots, \omega_{im})$ and the hash of the secret input and a randomly chosen value, $H(x_i + R_i)$, to be used for the signatures' generation. $H(x_i + R_i)$ is a publicly available value. Subsequently, each server runs **PartialEval** to generate public values y_j related to the final function value. A set of a coalition of the servers runs **PartialProof** and get the partial signatures. For instance, $\boldsymbol{\sigma_1}$ is the vector that contains the partial signatures of x_1, $\boldsymbol{\sigma_2}$ is the vector that contains the partial signatures of x_2 and so on. Anyone is able to run **FinalEval** to get y and **FinalProof** to get σ, which is the final signature that corresponds to the secret inputs $x_1, \ldots, x_n$. Finally, the **Verify** algorithm succeeds if and only if the final value y is correct.

Table 3. VAHSS with threshold signature sharing

Secret inputs (held by the clients)	Public values	Servers				
		s_1	s_2	$\cdots$	s_m	$\{s_{j_1}, \ldots, s_{j_t}\}$
$x_1,\ d_1$	$H(x_1 + R_1), e_1, \mathcal{A}_1$	$x_{11},\ \omega_{11}$	$x_{12},\ \omega_{12}$	$\cdots$	$x_{1m},\ \omega_{1m}$	$\boldsymbol{\sigma_1}$
$x_2,\ d_2$	$H(x_2 + R_2), e_2, \mathcal{A}_2$	$x_{21},\ \omega_{21}$	$x_{22},\ \omega_{22}$	$\cdots$	$x_{2m},\ \omega_{2m}$	$\boldsymbol{\sigma_2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_n,\ d_n$	$H(x_n + R_n), e_n, \mathcal{A}_n$	$x_{n1},\ \omega_{n1}$	$x_{n2},\ \omega_{n2}$	$\cdots$	$x_{nm},\ \omega_{nm}$	$\boldsymbol{\sigma_n}$
Partial sums (public)		y_1	y_2	$\cdots$	y_m	Final proof (public)
Total sum (public)		y				σ

- **Correctness:** To prove the correctness of our construction we need to prove that $\Pr\left[\mathbf{Verify}(H(x_1 + R_1), \ldots, H(x_n + R_n), \sigma, y) = 1\right] = 1$. For convenience, here, denote $H(x_i + R_i)$ by H_i. By construction:

$$\sigma = \prod_{i=1}^{n} {\sigma_i}^{e_i} = \prod_{i=1}^{n} (\overline{\sigma_i}^{\alpha_i} H_i^{\beta_i})^{e_i} = \prod_{i=1}^{n} (\prod_{j \in S} {\sigma_{ij}}^{\alpha_i} H_i^{\beta_i})^{e_i}$$
$$= \prod_{i=1}^{n} (H_i^{\beta_i} \prod_{j \in S} H_i^{2c_{j1}\omega_{ij}\alpha_i})^{e_i} = \prod_{i=1}^{n} H_i^{\beta_i e_i} H_i^{\sum_{j\in S} 2c_{j1}\omega_{ij}\alpha_i e_i}$$
$$\overset{see\ eq.(5)}{=} \prod_{i=1}^{n} H_i^{\beta_i e_i} H_i^{2\Delta_{iS} d_i \alpha_i e_i} = \prod_{i=1}^{n} H_i^{2\Delta_{iS}\alpha_i + \beta_i e_i} (\mod N)$$
$$\overset{see\ eq.(6)}{=} \prod_{i=1}^{n} H_i = \prod_{i=1}^{n} H(x_i + R_i) \text{ and also,}$$

$$\prod_{i=1}^{n} H(x_i + R_i) = \prod_{i=1}^{n} g^{x_i + R_i} \overset{see\ eq.(3)}{=} H(y) \tag{7}$$

Therefore, $\mathbf{Verify}(H(x_1 + R_1), \ldots, H(x_n + R_n), \sigma, y) = 1$ with probability 1, as desired.

- **Security:** The security of the signatures follows from the fact that the threshold signature scheme, which is employed in our construction, is secure, for $|T| \leq \mathfrak{t} - 1$, under the static adversary model given that the standard RSA signature scheme is secure [8]. Additionally, for $|T| \leq m - 1$, $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$ as we have proven in the Sect. 3.1. Therefore, our construction is secure for $|T| \leq \min\{\mathfrak{t} - 1, m - 1\}$.
- **Verifiability:** For $\mathbf{Verify}(H(x_1+R_1), \dots, H(x_n+R_n), \sigma', y') = 1$ and $y' \neq y$ we have:

$$\mathbf{Verify}(H(x_1 + R_1), \dots, H(x_n + R_n)), \sigma', y') = 1$$
$$\Rightarrow \sigma' = \prod_{i=1}^{n} H(x_i + R_i) \wedge H(y') = \prod_{i=1}^{n} H(x_i + R_i)$$
$$\Rightarrow H(y') = \prod_{i=1}^{n} H(x_i + R_i) \text{(see equation (7))} \Rightarrow H(y') = H(y)$$

which is a contradiction! Thus,

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

Table 4. Summary and comparison between the VAHSS proposed constructions.

Proposed construction	Cooperation between servers	Computations on client*
VAHSS with homomorphic hash functions	No	$(+)^{**}: 2m^2 + 3m + 1$, $(\times): 2m^2 + 2m$ (Exp.): 1
VAHSS with linear homomorphic signatures	No	$(+)^{**}: 2m^2 + 3m + 1$, $(\times): 2m^2 + 2m + 2$ (Exp.): 3
VAHSS with threshold signature sharing	Yes	$(+)^{**}: 2m^2 + 2m + m\mathfrak{t} + 1$, $(\times): 2m^2 + 2m + m\mathfrak{t}$ (Exp.): 1

*(+), (×), (Exp.) denote the number of additions, multiplications and exponentiations corresp.
**client n needs to perform $n - 1$ additional additions

4 Conclusion

In this paper, we addressed the problem of outsourcing joint additions, such that multiple clients give shares of their secret inputs to multiple untrusted servers. The latter perform the computations and then, anyone is able to ensure that the final output is correct (*i.e., public verifiability*). We instantiated three concrete constructions for the *verifiable additive homomorphic secret sharing* (VAHSS) problem by employing different cryptographic primitives and allowing the generation of the partial proofs by either the clients or the servers. In all three constructions, we achieved the property of public verifiability *i.e.,* anyone is able to confirm that the final result y is indeed the sum of the n secret inputs. In

Table 4, we provide a comparison of the proposed VAHSS constructions in terms of the employed primitives, the need for collaboration between the servers as well as the computation requirements on the client side. In all cases the computational cost required on the client side is rather similar *i.e.,* the computational complexity in all cases is $O(m^2)$ (where m denotes the number of servers) similarly to the complexity of a simple secret sharing scheme, while the one based on homomorphic hash functions seems to be slightly more lightweight. Our work is complementary to the multiplicative VHSS solution proposed by Tsaloli *et al.* [16]. The technique introduced in our constructions in order to randomize the τ_i values can also be incorporated in the multiplicative VHSS construction and, thus, provide better security guarantees.

Acknowledgement. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We would also like to thank Daniel Slamanig and Bei Liang for the helpful comments and discussions.

References

1. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 175–196. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_11
2. Bellare, M., Goldreich, O., Goldwasser, S.: Incremental cryptography: the case of hashing and signing. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 216–233. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_22
3. 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
4. Boyle, E., Garg, S., Jain, A., Kalai, Y.T., Sahai, A.: Secure computation against adaptive auxiliary information. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 316–334. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_18
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.: Function secret sharing: improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1292–1303. ACM (2016)
7. 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
8. Bozkurt, İ.N., Kaya, K., Selçuk, A.A.: Practical threshold signatures with linear secret sharing schemes. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 167–178. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02384-2_11
9. Catalano, D., Fiore, D., Warinschi, B.: Efficient network coding signatures in the standard model. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012.

LNCS, vol. 7293, pp. 680–696. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_40
10. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: new homomorphic primitives and applications. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 193–212. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_11
11. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_1
12. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
13. Krohn, M., Freedman, M., Mazieres, D.: On-the-fly verification of rateless erasure codes for efficient content distribution. In: 2004 Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, pp. 226–240 (2004)
14. Schabhüser, L., Butin, D., Buchmann, J.: Context hiding multi-key linearly homomorphic authenticators. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 493–513. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12612-4_25
15. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
16. Tsaloli, G., Liang, B., Mitrokotsa, A.: Verifiable homomorphic secret sharing. In: Baek, J., Susilo, W., Kim, J. (eds.) ProvSec 2018. LNCS, vol. 11192, pp. 40–55. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01446-9_3
17. Yao, H., Wang, C., Hai, B., Zhu, S.: Homomorphic hash and blockchain based authentication key exchange protocol for strangers. In: International Conference on Advanced Cloud and Big Data (CBD), Lanzhou, pp. 243–248 (2018)