Practical and Provably Secure Distributed Aggregation: Verifiable Additive Homomorphic Secret Sharing

Citation for the original published paper (version of record):
Tsaloli, G., Souza Banegas, G., Mitrokotsa, A. (2020). Practical and Provably Secure Distributed Aggregation: Verifiable Additive Homomorphic Secret
Sharing. Cryptography, 4(3). http://dx.doi.org/10.3390/cryptography4030025

cryptography

MDPI

Article

Practical and Provably Secure Distributed Aggregation: Verifiable Additive Homomorphic Secret Sharing †

Georgia Tsaloli *, Gustavo Banegas and Aikaterini Mitrokotsa

Department of Computer Science and Engineering, Chalmers University of Technology,
41296 Gothenburg, Sweden; gustavo@cryptme.in (G.B.); aikaterini.mitrokotsa@chalmers.se (A.M.)

* Correspondence: tsaloli@chalmers.se; Tel.: +46-73-3056904

† Information Security and Cryptology—ICISC 2019 Conference, Seoul, Korea, 4–6 December 2019.

Received: 29 July 2020; Accepted: 16 September 2020; Published: 21 September 2020

Abstract: Often clients (e.g., sensors, organizations) need to outsource joint computations that are based on some joint inputs to external untrusted servers. These computations often rely on the aggregation of data collected from multiple clients, while the 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. In this paper, we propose an approach, called verifiable additive homomorphic secret sharing (VAHSS), to achieve practical and provably secure aggregation of data, while allowing for the clients to protect their secret data and providing public verifiability i.e., everyone should be able to verify the correctness of the computed result. We propose three VAHSS constructions by combining an additive homomorphic secret sharing (HSS) scheme, for computing the sum of the clients' secret inputs, and three different methods for achieving public verifiability, namely: (i) homomorphic collision-resistant hash functions; (ii) linear homomorphic signatures; as well as (iii) a threshold RSA signature scheme. In all three constructions, we provide a detailed correctness, security, and verifiability analysis and detailed experimental evaluations. Our results demonstrate the efficiency of our proposed constructions, especially from the client side.

Keywords: function secret sharing; homomorphic secret sharing; verifiable computation; public verifiability

1. Introduction

The rise of communication technologies has formed multiple smart electronic devices (e.g., cell phones, sensors, wearables) with network connection, which produce a big amount of data every day. Remote, often untrusted, cloud servers are employed to store and process these data to be used by third parties, such as research institutions, hospitals, or electricity companies. Many applications (e.g., environmental monitoring, updating parameters in machine learning, statistics about electricity consumption) require joint computations on data coming from multiple clients. For example, using smart metering, data that are collected from sensors/clients can be used to compute statistics for the electricity consumption, while environmental sensors collect data that can be used to measure emissions and data collected from mobile phones can be aggregated and processed to appropriately update parameters in machine learning models for accurate user profiling.

Although decentralization has been a recent trend, we have witnessed a steady rise of massively distributed but not decentralized systems. When multiple clients outsource a joint computation using their joint inputs, multiple servers can be employed in order to avoid single points of failure and perform a reliable joint computations on the clients' inputs. Although this distributed cloud-assisted environment is very appealing and offers exceptional advantages, it is also followed by serious security and privacy challenges. Thus, in this work, we present the formal and practical analysis of verifiable additive homomorphic secret sharing (VAHSS), a novel cryptographic primitive introduced [1], which allows for multiple clients to outsource the joint addition of their inputs to multiple untrusted servers, providing guarantees that the clients' inputs remain secret as well as that the computed result is correct (i.e., verifiability property). More precisely, this paper is an extended version of the preliminary article [1].

We address the problem of cloud-assisted computing characterized by the following constraints: (i) multiple servers are recruited to perform joint additions on the inputs of n clients; (ii) the inputs of the clients need to remain secret; (iii) the servers are untrusted; (iv) no communication between the clients is possible; and, (v) anyone should be able to confirm that the computed result is correct (i.e., public verifiability property). More precisely, let us consider n clients (as illustrated in Figure 1), which hold n individual secret inputs $x_1, x_2, \dots x_n$, and they want to deploy the joint computation of the function $f(x_1, x_2, \dots, x_n) = x_1 + x_2 + \dots + x_n$ on their joint inputs by releasing shares of their inputs to multiple untrusted servers (the latter denoted by s_j for $j \in [m]$). We denote the share of a client c_i given to the server s_j by x_{ij}. Tsaloli et al. [2] 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 jointly perform the computation of a function $f(x_1, x_2, \dots, x_n) = y$, including no communication between the clients, and allowing anyone to get a proof π that the computed result is correct, i.e., providing to anyone a pair (y, π) which confirms the correctness of y (i.e., public verification). However, the possibility to achieve verifiable homomorphic secret sharing for other functions (e.g., addition) has been left open.

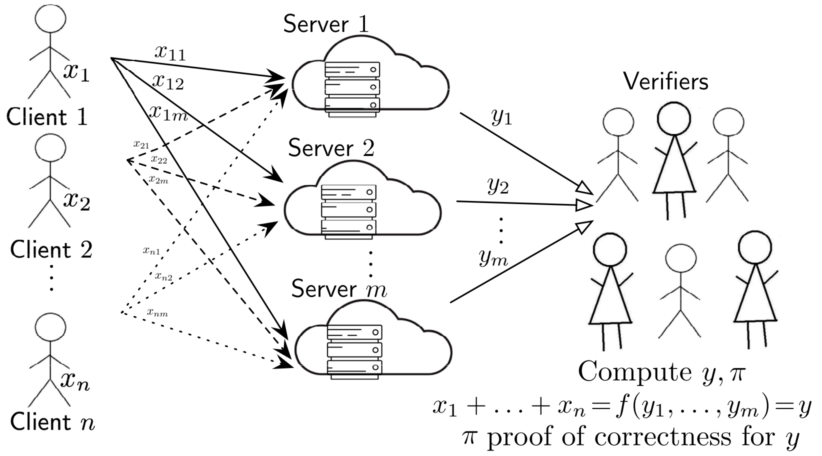


Figure 1. n clients outsourcing the joint addition of their joint inputs to m servers.

In this paper, we revisit the concept of verifiable homomorphic secret sharing (VHSS) and we explore the possibility to achieve verifiable additive homomorphic secret sharing. Our research shows that the latter is possible and we propose three concrete constructions that employ m servers to jointly compute the additions of n clients' inputs securely and privately, while, additionally, ensuring public verifiability. The proposed constructions can be utilized, for example, to compute statistics over electricity consumption when data are collected from multiple clients, in order to remotely monitor and determine a diagnosis for multiple patients according to their collected data, as well as to measure environmental conditions while using multiple sensors' data that come from environmental sensors (e.g., temperature, humidity). We have substantially extended the preliminary article [1] and added a detailed evaluation (both theoretical and experimental) of the three proposed VAHSS constructions. In the submitted paper, we present a detailed analysis for each of the constructions based on different conditions, providing both theoretical and experimental results.

Our Contribution. We address the problem of computing joint additions with privacy and security guarantees as the main requirements. More precisely, we treat the problem of verifiable multi-client aggregation that involves the following parties: (i) n clients, which hold secret inputs $x_1, x_2, \ldots, x_n$, respectively; (ii) m untrusted servers to whom the sum computation is outsourced; and, (iii) any verifier that would like to confirm that the computed sum is correct. We present, for the first time, three concrete constructions of verifiable additive homomorphic secret sharing (VAHSS).

We employ three different primitives (i.e., homomorphic hash functions, linearly homomorphic signatures, and threshold signatures) as the baseline for the generation of partial proofs (values that are used to confirm the correctness of the computed sum). The partial proofs are computed by either the servers or the clients. These characteristics lead to three different instantiations of VAHSS. Additionally, we have altered the original VHSS definition to capture the different scenarios on the proofs' generation; therefore, allowing for the employment of VHSS in several application settings.

Our constructions rely on casting Shamir's secret sharing scheme over a finite field $\mathbb{F}$ as an n-client, m-server, and t-perfectly secure additive homomorphic secret sharing (HSS) for the function that sums n field elements. Firstly, employing homomorphic collision-resistant hash functions [3,4] and incorporating them to the additive HSS, we design a construction, such that each server produces a partial proof. Next, a linearly homomorphic signature scheme [5] is combined with the additive HSS, which results in an instantiation where each client generates a partial proof. Ultimately, the employment of a threshold RSA signature scheme [6] in additive HSS allows a subset of servers to generate partial proofs that correspond to each client. In all three proposed constructions, we have provided detailed correctness, security, and verifiability analysis. Furthermore, we provide an evaluation of the three proposed constructions, in which we describe the cost of the required operations for each of the employed algorithms as well as present a detailed experimental evaluation. More precisely, we evaluate the performance of all three proposed VAHSS constructions and compare and illustrate how the employed algorithms perform, depending on the amount of the clients that participate during the computation and the required computation time for the verification process.

Organization. Section 2 gives an overview about the current state-of-the-art in homomorphic secret sharing and verifiable computation. Later, Section 3 provides the background on verifiable homomorphic secret sharing and definitions that are necessary for the rest of the paper. We provide our VHSS constructions in Section 4. Furthermore, in Section 5 we give a theoretical analysis of the costs for the constructions and provide details of our implementation results. We present a discussion about the constructions and their costs in Section 6. Finally, we provide our final considerations in Section 7.

2. Related Work

Homomorphic Secret Sharing. The key idea of threshold secret sharing schemes [7] is the ability to split a secret x into multiple shares (denoted by, for example, $x_1, x_2, \ldots, x_m$) while maintaining the following properties: (i) any subset greater than the threshold number of shares is enough to reconstruct the secret x, and (ii) any smaller subset allows no inference of information related to the secret x. Homomorphic secret sharing (HSS) [8] can be seen as the secret-sharing analog of homomorphic encryption. In particular, HSS is employed in order to locally evaluate functions on shares of secret inputs (or just one input), by appropriately combining locally computed values with the shares of the secret(s) as input. At the same time, an HSS system ensures that the shares of the output are short. The first instance of additive HSS that is considered in the literature [9] is computed in some finite Abelian group. Nevertheless, HSS gives no guarantee that the computed result is correct i.e., no verifiability is provided.

Verifiable Function Secret Sharing. To better realize function secret sharing (FSS) [10], one can consider it as a natural extension of distributed point functions (DPF). More precisely, FSS is a method to create shares for a function f, coming from a given function family $\mathcal{F}$, that are additively combined to give f. To visualize this, consider m functions $f_1, \ldots, f_m$, as described by the corresponding keys $k_1, \ldots, k_m$. These functions are the shares of f, such that $f(x) = f_1(x) + \ldots + f_m(x)$, for any input x. The notion of verifiable FSS (VFSS) [11] is introduced by Boyle et al. In particular, VFSS consists of interactive protocols that verify the consistency of some function $f \in \mathcal{F}$ with keys $(k_1^*, \ldots, k_m^*)$, which are generated by a potentially malicious user. However, VFSS [11] is applicable in the setting of multiple servers and one client. On the contrary, VHSS can be applied when multiple clients (multi-input) outsource the joint computation to multiple servers. Moreover, in VFSS, verification refers to confirming that the shares $f_1, \dots, f_m$ are consistent with some f; while, in VHSS, the goal of the verification is to ensure that the final result is correct.

Publicly Auditable Secure Multi-party Computation. Secure multi-party computation (MPC) protocols are linked with outsourced computations. In MPC protocols [12–14], non-interactive zero-knowledge (NIZK) proofs are generally used in order to achieve public verifiability. Baum et al. [15] introduced the notion of *publicly auditable* MPC protocols that are applicable when multiple clients and servers are involved. Given that publicly auditable MPC is based on the SPDZ protocol [12,13] and NIZK proofs, while it also employs Pedersen commitments (for enhancing each shared input x), it can be viewed as a generalization of the classic formalization of secure function evaluation. In the work of Baum et al. [15], anyone that has access to the published (in a bulletin board) transcript of the protocol can confirm that the computed result is correct (correctness property); while, the protocol provides privacy guarantees and requires at least one honest party. We should note that publicly auditable MPC protocols are very expressive regarding the class of functions being computed, but they 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 other hand, in VAHSS, we do not require any additional non-corruptible party as well as we do not employ expensive cryptographic primitives, such as NIZK proofs.

3. Preliminaries

Our concrete instantiations for the additive VHSS problem are based on the VHSS definition that was proposed in [2]. 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, which is reflected later in our instantiations. More precisely, depending on the construction, the execution of the **PartialProof** algorithm is performed by either the clients or the servers. We added the **Setup** algorithm to allow for 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 seven-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, a secret key sk, to be used by a client , and some public parameters pp .*
- $(\mathsf{share}_{i1}, \dots, \mathsf{share}_{im}, \tau_i) \leftarrow$ ***ShareSecret***$(1^\lambda, i, \boldsymbol{x}_i)$*: The algorithm takes as input 1^λ, $i \in \{1, \dots, n\}$ 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 (τ_i, when computed, can be included in the list of public parameters pp) related to the secret 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 $x_1, \dots, x_n$ 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. Note that k is a variable; thus, $k = i$ when **PartialProof** generates proofs per client or $k = j$ if it generates proofs per server.*
- $y \leftarrow$***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 compute, the algorithm **FinalEval** outputs y, the final result for $f(x_1, x_2, \dots, x_n)$.*

- $\sigma \leftarrow$***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. Note that* $|k| = n$, *if the partial proofs are computed per client or* $|k| = m$, *if they are computed per server.*
- $0/1 \leftarrow$***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**, and **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:

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

- **Verifiability**: let T be the set of corrupted servers with $|T| \leqslant m$ (note that, for $|T| = m$, the verifiabililty property holds; however, we do not have a secure system). 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. 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 $\sigma_k{}'$. Subsequently, for j, such that $s_j \notin T$, we set $y_j{}' =$ **Partial- Eval**$(j, (x_{1j}, \ldots, x_{nj}))$ and $\sigma_k{}' =$ **PartialProof**$(sk, pp, \mathrm{secret}_{\mathrm{values}}, k)$. 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:

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

- **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**$(1^\lambda, i, \hat{x}_i)$ where the secret input $\hat{x}_i = \begin{cases} x_i, \text{if } b = 0 \\ x_i', \text{otherwise} \end{cases}$.
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 $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) := \Pr[b = b'] - 1/2$ 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 $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$.

In our solution, we employ a simple variant of the (Strong) RSA based signature that was introduced by Catalano et al. [16], which can be seen as a linearly homomorphic signature scheme on $\mathbb{Z}_N$.

Definition 2 (Linearly Homomorphic Signature [5])**.** *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 [4].

Definition 3 (Homomorphic Hash Function [3])**.** *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 that satisfies 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 "$\cdot$".*

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 that $\{F_s\}$ 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 $\{0,1\}^n$.

4. 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 $x_1, \dots, x_n$ respectively, and m servers $s_1, \dots, s_m$ that perform computations on shares of these secret values. Firstly, the clients split their secret values into shares, which 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 $f(x_1, \dots, x_n) = x_1 + \dots + x_n$. Subsequently, partial proofs are generated in a different way, depending on the instantiation proposed. The partial proofs are values, such that their combination results in a final proof, which confirms the correctness of the final computed value $f(x_1, \dots, x_n)$. Note that, for all of the proposed constructions, the clients do not need to communicate with each other, which often is the case in settings where the clients are wireless devices spread in different regions and are not in the communication range of each other (e.g., sensors measuring electricity consumption or environmental conditions). However, the clients could potentially collude with some of the servers, without compromising the security of our constructions as long as at least two clients remain honest.

4.1. Construction of VAHSS Using Homomorphic Hash Functions

In this section, we aim to compute the function value y, which corresponds to $f(x_1, \dots, x_n) = x_1 + \dots + x_n$ as well as a proof σ that y is correct. We combine an additive HSS for the algorithms that are related to the value y and hash functions for the generation of the proof σ. Let $c_1, \dots, c_n$ denote n clients and $x_1, \dots, x_n$ their corresponding secret inputs. Let, for any $\{i\}_{i=1,\dots,n}$, $\theta_{i1}, \dots, \theta_{im}$ be distinct non-zero field elements and $\lambda_{i1}, \dots, \lambda_{im}$ 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 $x_{i1}, \dots, x_{im}$, 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 $y = y_1 + \dots + y_m$, which corresponds to the function value $y = f(x_1, \dots, x_n) = x_1 + \dots + x_n$. We suggest that every client c_i uses a homomorphic collision-resistant function $H : x \mapsto g^x$ proposed by Krohn et al. [4] to generate a public value τ_i, which reveals nothing about x_i (under the discrete logarithm assumption). Afterwards, the servers compute values $\sigma_1, \dots, \sigma_m$, 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**$(1^\lambda, i, x_i)$: for elements $\{a_i\}_{i \in \{1,\dots,t\}} \in \mathbb{F}$ selected uniformly at random, pick a t-degree polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + \dots + a_t X^t$. 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 [3]. 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 a timestamp $file_i$ associated with client i such that $(i, file_i) \in \{0,1\}^{l_2}$. For $i = n$, we require $\mathbb{F} \ni R_n = \phi(N) \lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil - \sum_{i=1}^{n-1} R_i$. Subsequently, compute $\tau_i = H(x_i + R_i)$, define $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$ (given thanks to the Equation (1)) and output $(x_{i1}, x_{i2}, \dots, x_{im}, \tau_i) = (\lambda_{i1} \cdot p_i(\theta_{i1}), \dots, \lambda_{im} \cdot p_i(\theta_{im}), H(x_i + R_i))$.
2. **PartialEval** $(j, (x_{1j}, x_{2j}, \dots, x_{nj}))$: 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 $y_j = \lambda_{1j} \cdot p_1(\theta_{1j}) + \dots + \lambda_{nj} \cdot p_n(\theta_{nj}) = \sum_{i=1}^{n} \lambda_{ij} \cdot p_i(\theta_{ij})$.

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. Subsequently, 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 it produces the value σ_j. Finally, any user or verifier is able to run the **FinalEval** algorithm to obtain 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. Table 1 illustrates our construction.

Table 1. Verifiable additive homomorphic secret sharing (VAHSS) using homomorphic hash functions.

Secret Inputs	**Servers**				**Public**
(Held by the Clients)	s_1	s_2	$\cdots$	s_m	**Values**
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**: In order 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:

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

$$\begin{aligned} \text{and} \quad \prod_{i=1}^{n} \tau_i &= \prod_{i=1}^{n} g^{x_i + R_i} = g^{\sum_{i=1}^{n} x_i} g^{\sum_{i=1}^{n} R_i} = g^{\sum_{i=1}^{n} x_i} g^{\sum_{i=1}^{n-1} R_i + R_n} \\ &= g^{\sum_{i=1}^{n} x_i} g^{\phi(N) \lceil \frac{\sum_{i=1}^{n-1} R_i}{\phi(N)} \rceil} = g^{\sum_{i=1}^{n} x_i} = g^{x_1 + \ldots + x_n} \\ &\overset{\textit{see Equation (2)}}{=} g^{y} = H(y) \end{aligned} \tag{3}$$

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 [17] 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. Subsequently, $|T| = m-1$. Without a 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 $\widehat{\tau}_i = g^{\hat{x}_i + R_i}$. Because 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 the 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:

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

$$\Rightarrow \prod_{i=1}^{n} \tau_i = H(y') \quad \text{(see Equation (3))} \Rightarrow H(y) = H(y')$$

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

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

4.2. 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 while using additive HSS and we employ a linearly homomorphic signature scheme, presented in [5] as a simple variant of Catalano et al. [16] signature scheme, for the generation of the proof. All of the 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) that is useful for some calculations. Because 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 [18] 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 Section 4.1. The signatures, which are denoted by $\sigma_1, \ldots, \sigma_n$, are public and, when 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. Subsequently, 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 $p_i(X) = x_i + a_1 X + a_2 X^2 + \ldots + a_t X^t$. Notice that the free coefficient of p_i is the secret input x_i. Subsequently, 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** in order 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. Subsequently, 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 $y = x_1 + \ldots + x_n$. Table 2 reports an illustration of our solution.

Table 2. VAHSS using linear homomorphic signatures.

Secret Inputs	**Servers**				**Public Values**
(Held by the Clients)	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 $\Pr\Big[\ \textbf{Verify}(vk, f, \sigma, y) = 1\ \Big] = 1$. It holds that:

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

$$\overset{\textit{see Equation } (3)}{=} g^{\sum_{i=1}^{n} s_i - s'eN} \prod_{j=1}^{n} h_j^{\sum_{i=1}^{n} f_j^{(i)} - f'_j eN} {g_1}^{\sum_{i=1}^{n} x_i} = g^s \prod_{j=1}^{n} h_j^{f_j} {g_1}^{\sum_{i=1}^{n} x_i} \tag{4}$$

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

- **Security**: The security of the signatures easily results from the original signature scheme that was proposed by Catalano et al. [16]. Moreover, $\mathrm{Adv}(1^\lambda, \mathcal{A}, T) \leq \varepsilon(\lambda)$ for some negligible $\varepsilon(\lambda)$ as we have proven in the Section 4.1. 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 $\sigma \leftarrow \textbf{FinalProof}(vk, \hat{f}, \sigma_1, \ldots, \sigma_n)$ 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 $\textbf{Verify}(vk, \sigma', y', f) = 1$, then:

$$\begin{aligned}
&\textbf{Verify}(vk, \sigma', y', f) = 1 \Rightarrow \textbf{Verify}(vk, \sigma, y', f) = 1 \\
&\Rightarrow \widetilde{x}^{eN} = g^s \prod_{j=1}^{n} h_j^{f_j} g_1^{y'} \quad \text{(see Equation (4))} \\
&\Rightarrow g^s \prod_{j=1}^{n} h_j^{f_j} g_1^{\sum_{i=1}^{n} x_i} = g^s \prod_{j=1}^{n} h_j^{f_j} g_1^{y'} \Rightarrow \sum_{i=1}^{n} x_i = y'
\end{aligned}$$

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

4.3. 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 [6], so that a signature σ is successfully generated, even if $\mathrm{t} - 1$ 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 $q = 2q' + 1$, where p', q' are large primes. Choose the public RSA key e_i, such that $e_i \gg \binom{n}{\mathrm{t}}$ 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**$(1^\lambda, i, x_i, d_i,)$: 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 $p_i(X) = x_i + a_1 X + a_2 X^2 + \ldots + a_t X^t$. Notice that the free coefficient of p_i is the secret input x_i. Subsequently, define $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$ (given thanks

to the Equation (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 that is 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 [3]. Let R_i be randomly selected values, as described in the Section 4.1. 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 $S = \{s_1, s_2, \ldots, s_\mathsf{t}\}$ be the coalition of t servers ($\mathsf{t} < m$) (w.l.o.g. take the first t), forming the system $\mathcal{A}_{iS}\boldsymbol{d} = \boldsymbol{\omega_{iS}}$. Let the $\mathsf{t} \times \mathsf{t}$ adjugate matrix of $\mathcal{A}_{iS}$ be:

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

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

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

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

   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 whether $\sigma = \prod_{i=1}^{n} H(x_i + R_i) \wedge H(y) = \prod_{i=1}^{n} H(x_i + R_i)$ 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. Subsequently, 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 obtains 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	**Public**	**Servers**				
(Held by the Clients)	**Values**	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}, ω_{11}	x_{12}, ω_{12}	$\cdots$	x_{1m}, ω_{1m}	$\boldsymbol{\sigma_1}$
x_2, d_2	$H(x_2+R_2), e_2, \mathcal{A}_2$	x_{21}, ω_{21}	x_{22}, ω_{22}	$\cdots$	x_{2m}, ω_{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}, ω_{n1}	x_{n2}, ω_{n2}	$\cdots$	x_{nm}, ω_{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 $\Pr\Big[\ \textbf{Verify}(H(x_1+R_1), \ldots, H(x_n+R_n), \sigma, y) = 1\ \Big] = 1$. For convenience, here, denote $H(x_i + R_i)$ by H_i. By construction:

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

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

  Therefore, **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 $|T| \leq \mathrm{t} - 1$, under the static adversary model given that the standard RSA signature scheme is secure [6]. 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 Section 4.1. Therefore, our construction is secure for $|T| \leq \min\{\mathrm{t} - 1, m - 1\}$.

- **Verifiability**: for **Verify**$(H(x_1+R_1), \ldots, H(x_n+R_n), \sigma', y') = 1$ and $y' \neq y$ we have:

$$\begin{aligned}&\textbf{Verify}(H(x_1+R_1), \ldots, H(x_n+R_n)), \sigma', y') = 1 \\ &\Rightarrow \sigma' = \prod_{i=1}^{n} H(x_i + R_i) \wedge H(y') = \prod_{i=1}^{n} H(x_i + R_i) \\ &\Rightarrow H(y') = \prod_{i=1}^{n} H(x_i + R_i) \quad \text{(see Equation (7))} \Rightarrow H(y') = H(y)\end{aligned}$$

  which is a contradiction! Thus,

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

5. Evaluation

In this section, we provide the evaluation of our proposed constructions. First, we perform a theoretical analysis and present the amount of operations required to implement each construction. Next, we provide a detailed experimental evaluation; we describe the experimental setup and provide the required computational time for each of the presented transactions for different conditions, from both the client and the server side.

5.1. Theoretical Analysis

We now present the basic operations that are required in our constructions. Recall that we denote the number of clients by n and the number of servers by m. The threshold for generating shares of the secret clients' inputs is denoted by t, while the threshold used in the proposed third VAHSS construction i.e., based on threshold signature sharing, for generating shares of the private RSA key, is denoted by t. All of the operations are reported per algorithm and, thus, considering the cost for each client or server, respectively. This means that if, for instance, $n-1$ additions correspond to the **PartialEval** algorithm in the table, this number represents the amount of operations that are required from each server that executes the **PartialEval** algorithm.

In our constructions, in the **Sharesecret** algorithm, each client generates m shares that are related to its secret inputs. Below, we present in detail how we calculate the cost for the client to compute these m shares. Each client needs m polynomial evaluations (denoted by P_{eval}) as well as the computation of the Lagrange coefficients (denoted by L_{coeff}), as shown in Equation (1). We consider Horner's method [19] to calculate P_{eval}, which gives t multiplications and t additions for a polynomial of degree t, as in our constructions. Furthermore, L_{coeff} requires 2 multiplications, 1 computation of inverse, and 1 addition (in $\mathbb{F}$) for each factor out of $m-1$. One additional multiplication is needed in order to form the right hand side of Equation (1). Therefore, the cost for generating m secret shares can be demonstrated, as follows:

$$\begin{aligned} Shares_{cost} =& mP_{eval} + L_{coeff} + 1_{mul} \\ =& m(t_{add} + t_{mul}) + (m-2)(1_{mul} + 1_{inv} + 1_{add}) + 1_{mul} \\ =& mt_{add} + mt_{mul} + (m-2)_{mul} + (m-2)_{inv} + (m-2)_{add} + 1_{mul} \\ =& (mt+m-2)_{add} + (mt+m-1)_{mul} + (m-2)_{inv} \end{aligned} \tag{8}$$

This amount of operations is added in the **Sharesecret** algorithm costs. Let us now present the tables that summarize the cost of our constructions. In parentheses, we display by whom the algorithm is executed. Whenever not specified, the algorithm can be run from any verifier. Table 4 illustrates the costs for the VAHSS construction while using homomorphic hash functions (also found as VAHSS-HSS). Observe that there is no **Setup** algorithm in this construction. Table 5 shows the number of operations required in the VAHSS construction while using linear homomorphic signatures (also found as VAHSS-LHS). Finally, Table 6 gives the costs of the VAHSS construction while using threshold signature sharing (found also as VAHSS-TSS). Observe that the same variables are used to show the theoretical results. However, note that t appears in the VAHSS-TSS construction for the first time, denoting a different number than the threshold t that is used for generating shares of the secret inputs. Moreover, the algorithms are executed from either a client or server, depending on the construction presented.

Looking at Tables 4–6, we observe some differences regarding the costs that are expected in each algorithm. More precisely, we can see that the **ShareSecret** algorithm is always run by the clients. In this respect, the VAHSS-HSS and VAHSS-LHS constructions slightly differ, since a client needs to produce some additional public values in the VAHSS-HSS case. The VAHSS-TSS is more expensive, since the client also generates shares of its private RSA key. Furthermore, as we have mentioned, VAHSS-HSS requires no **Setup**, thus, it is computationally less expensive than the other two constructions from the client's side. Subsequently, we can observe that the **PartialEval** and **FinalEval** algorithms are always run by the servers and are expected to have the same computational cost. The **PartialProof** algorithm is run either by the servers or the clients. In the VAHSS-HSS construction, the computation is low and is made by the servers, while, in the VAHSS-LHS, the execution is made by the clients and it is also low-cost. In the VAHSS-TSS solution, **PartialProof** is run by a coalition of servers and the cost depends on the amount of the servers that can be considered in the computation. Next, **FinalProof** is quite practical in all cases, but it is not particularly comparable since there are several parameters that can affect this cost. Finally, the verification process (**Verify**) has the same computational cost for the first and third construction, while it is slightly heavier for the VAHSS-LHS construction.

Table 4. Number of operations of the VAHSS based on homomorphic hash functions (VAHSS-HSS) construction.

Operation **Algorithm**	**Addition**	**Multiplication**	**Exponentiation**	**Random Sampling**
ShareSecret (client)	$mt + m - 1$	–	$mt + m$	$m - 1$
PartialEval (server)	$n - 1$	–	–	–
PartialProof (server)	$n - 1$	–	1	–
FinalEval	$m - 1$	–	–	–
FinalProof	–	$m - 1$	–	–
Verify	–	$n - 1$	1	–

Table 5. Number of operations of the VAHSS based on linear homomorphic signatures (VAHSS-LHS) construction.

Operation **Algorithm**	**Addition**	**Multiplication**	**Exponentiation**	**Inverse Computation**	**Random Sampling**
Setup (client)	–	–	–	–	$n + 2$
ShareSecret (client)	$mt + m - 2$	$mt + m - 1$	$m - 2$	–	–
PartialEval (server)	$n - 1$	–	–	–	–
PartialProof (client)	–	3	4	1	1
FinalEval	$m - 1$	–	–	–	–
FinalProof	n	$n + 1$	1	–	–
Verify	–	$n + 1$	3	–	–

Table 6. Number of operations of the VAHSS based on threshold signature sharing (VAHSS-TSS) construction.

Operation **Algorithm**	**Addition**	Multiplication	**Exponentiation**	**Inverse Computation**	**Random Sampling**
Setup (client)	–	–	–	1	1
ShareSecret (client)	$mt + m\mathrm{t} - 1$	$mt + m\mathrm{t} + m - 1$	$m - 1$	–	1
PartialEval (server)	$n - 1$	–	–	–	–
PartialProof (server)	–	2t	t	–	–
FinalEval	$m - 1$	–	–	–	–
FinalProof (server)		$\mathrm{t} + n - 2$	$n + 2$	–	–
Verify	–	$n - 1$	1	–	–

5.2. Prototype Analysis

In this section, we present our results from the experimental analysis regarding the performance of the three proposed VAHSS constructions. More precisely, we have implemented the following constructions: VAHSS based on homomorphic hash functions (VAHSS-HSS), VAHSS based on linear homomorphic signatures (VAHSS-LHS), and VAHSS based on threshold signature sharing (VAHSS-TSS) and compare them regarding their performance.

In our implementations, we used the programming language "C++" and the GMP Library (https://gmplib.org/) for handling big numbers and their arithmetic operations. Furthermore, we ran the experiments on Arch Linux Kernel 5.7.7 over a Dell Latitude 5300 with processor Intel i5-8365U CPU @ 1.60 GHz (micro architecture codename Whisky Lake), with 16 GB RAM, 32KiB L1d cache, 32KiB L1i cache, 256KiB L2 cache, and 6MiB L3 cache. In order to perform a fair comparison, we have selected the following common parameters for all of the implementations: group generator, number of clients, number of servers, and the finite field for the Shamir's secret sharing.

We have used a benchmarking dataset for the experimental evaluation. More precisely, we have used the individual household electric power consumption dataset by the UC Irvine machine learning repository (https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption#). The values represent electricity consumption provided by smartmeters. We note that, since the values in the dataset are float numbers, we need to preprocess them in order to employ them in our constructions by multiplying all of the input values by 100. Below, we list the required computation cost (in time) for all different algorithms of the proposed constructions.

We test our constructions for 500 clients, 3 servers, and a 64-bit prime number for forming the finite field $\mathbb{F}$ that is used for the secret inputs' shares generation. Additionally, the primes that are used for the VAHSS-LHS and VAHSS-TSS constructions are randomly generated primes of 128 bits. The timing is measured and shown in microseconds. The **Sharesecret**, **PartialEval**, and **PartialProof** costs are represented by their median values. For instance, for the **Sharesecret** algorithm, each client generates a random polynomial to be used for the shares' generation and, since we consider 500 clients, we need to take into account their different costs. Thus, we get all of the timings and sort them in order to obtain the 250th element of the list. Similarly, we also obtain the median values for **PartialEval** and **PartialProof**. Table 7 illustrates the time in microseconds for 500 clients for the three different constructions.

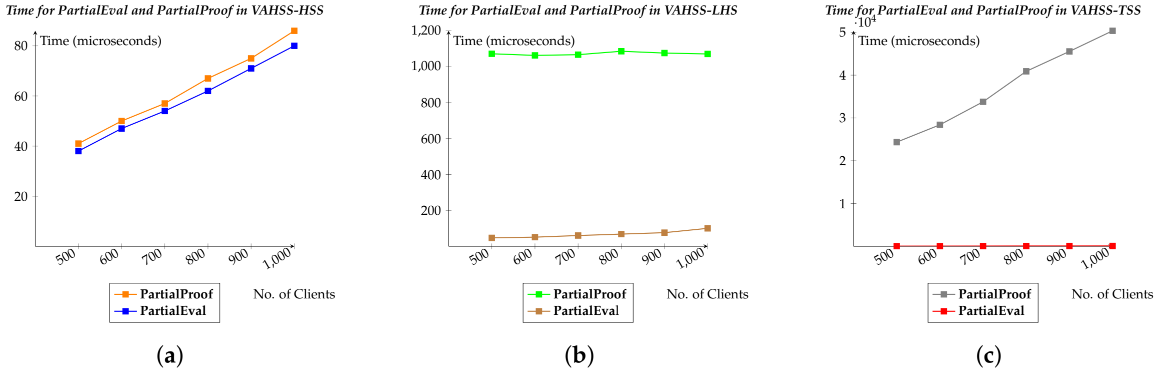
Table 7. Timing (in microseconds) for each of our constructions.

Construction Algorithm	VAHSS-HSS	VAHSS-LHS	VAHSS-TSS
Setup	0 [1]	2540	310
Sharesecret	300	298	299
PartialEval	58	47	76
PartialProof	49	1072	24,293
FinalEval [2]	479	979	882
Final Proof	550 [3]	537	17,192
Verify	147	9091	294

[1] It does not require key generation; [2] Timing in nanoseconds; [3] Timing in nanoseconds.

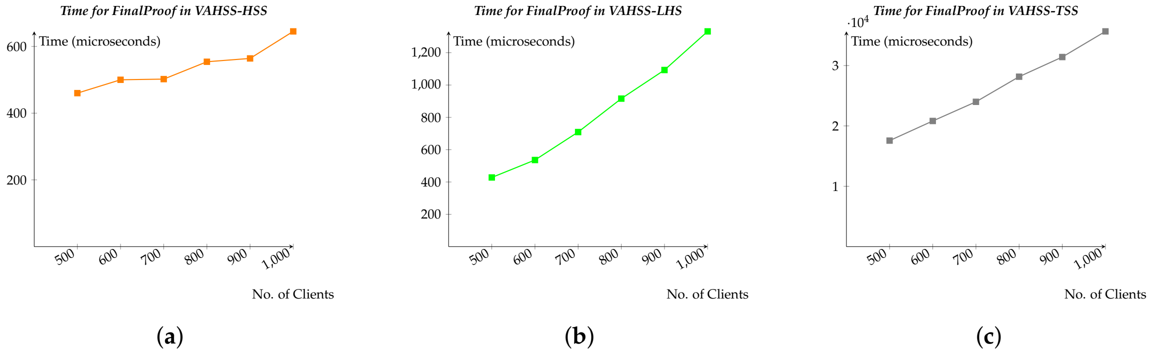
Our tests were extended to different amounts of clients, ranging between 500 and 1000, while we fixed the number of servers to 3. Moreover, we should note that we ran our experiments for several prime numbers of various sizes and no significant change was noticed; therefore, these results are omitted. Below, we provide the results for the algorithms, which performed noticeably different for different parameters, as illustrated by figures. Figure 2 shows the timing for executing the **PartialEval** and **PartialProof** algorithms in each of our constructions. More precisely, Figure 2a shows the VAHSS-HSS case, Figure 2b demonstrates the VAHSS-LHS case, while the VAHSS-TSS case is depicted in Figure 2c. The graphs show how the timing changes depending on the number of clients participating in the computation. Next, Figure 3 shows the time that is required for computing the **Finalproof** in each of the constructions, representing again how the performance varies according to a different amount of clients. Finally, Figure 4 shows the timing for executing the **Verify** algorithm given the outputs from each server and from the clients. We remind the reader that anyone may run the **Verify** algorithm in order to check the correctness of the resulted y value and obtain y itself.

For further details, the code is available in a github repository (https://github.com/tsaloligeorgia/AddVHSS).



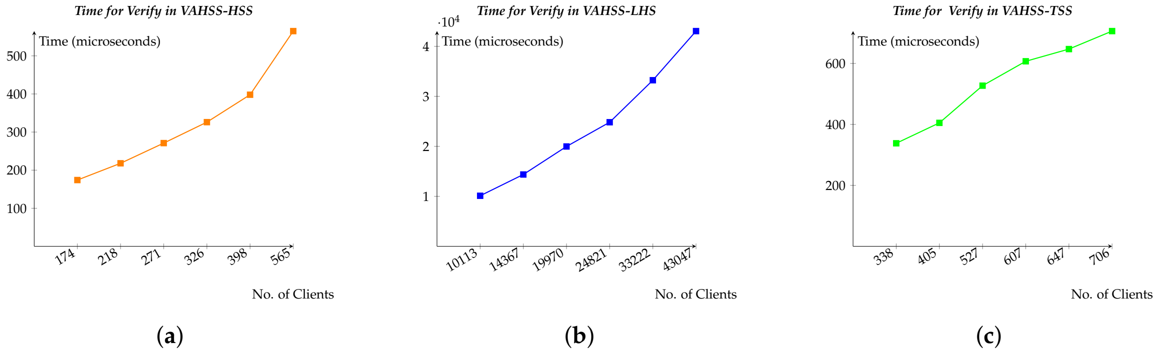
(**a**) (**b**) (**c**)

Figure 2. Time for **PartialEval** and **PartialProof** in our constructions. (**a**) Time for **PartialEval** and **PartialProof** in VAHSS-HSS. (**b**) Time for **PartialEval** and **PartialProof** in VAHSS-LHS. (**c**) Time for **PartialEval** and **PartialProof** in VAHSS-TSS.



(**a**) (**b**) (**c**)

Figure 3. Time for **FinalProof** in our constructions. (**a**) Time for the **FinalProof** algorithm in VAHSS-HSS. (**b**) Time for the **FinalProof** algorithm in VAHSS-LHS. (**c**) Time for the **FinalProof** algorithm in VAHSS-TSS.



(**a**) (**b**) (**c**)

Figure 4. Time for running **Verify** for each of our constructions. (**a**) Time for the **Verify** algorithm in VAHSS-HSS. (**b**) Time for the **Verify** algorithm in VAHSS-LHS. (**c**) Time for the **Verify** algorithm in VAHSS-TSS.

6. Discussion

We have presented three verifiable additive homomorphic secret sharing (VAHSS) constructions and, then, provided their theoretical evaluation. Furthermore, we developed a prototype and compared the performance (required computation time) of the proposed constructions for each used algorithm, given different parameters and conditions, as shown in Section 5.2. To the best of our knowledge, no existing scheme achieves privacy-preserving distributed verifiable aggregation based on homomorphic operations.

As mentioned earlier, each construction relies on a different mathematical component and might be used in different application scenarios. More precisely, the constructions differ on how the partial evaluations (**PartialEval**) and the partial proofs (**PartialProof**) are generated and who performs each computation. For instance, in the VAHSS-HSS construction, clients are only needed to execute the **Sharesecret** algorithm for generating m shares, and the rest of the computations (required to produce the sum y and the proof σ) are performed by the servers. Nevertheless, the VAHSS-LHS construction requires that each client deals with the **Setup**, **ShareSecret**, and **PartialProof** algorithms. In fact, in this case, the clients are the ones that generate the partial proofs instead of the servers, utilizing their private RSA key. Moreover, in the VAHSS-TSS construction, the client runs the **Setup** and **ShareSecret**, while the servers deal with the execution of the other algorithms. Additionally, the VAHSS-TSS solution is based on a threshold signature sharing scheme and, as a result, a coalition of servers is required to perform the **PartialProof** algorithm; not all of the m servers are needed.

The computation cost required by each construction is different, as demonstrated in Section 5. However, this does not necessarily imply that one construction is better than another. Actually, it shows that there is a trade-off to choose from. For example, if the employed devices to function as clients have power/process constraints, then the best option could be the construction VAHSS-HSS, since it requires fewer computations and, as a consequence, it is less expensive regarding power consumption. However, if the employed clients are devices with significant power resources and the application requires that the clients produce the partial proofs, then the best option would be the VAHSS-LHS construction.

Furthermore, the metric of the number of operations could be used to establish which flavor of VAHSS is more appropriate, depending on the application setting and, in this case, the VAHSS-HSS construction requires fewer operations on the client-side. The prototype implementation and evaluation reinforces the fact that the VAHSS-HSS construction presents better timing for most of the required operations. As an additional metric for comparison, we employ the communication overhead (required bandwidth) for each of the proposed constructions. More precisely, Figures 5–7 show the required bandwidth usage for each construction. The figures give the number of bytes received and sent per client and per server. Figure 5 shows that the VAHSS-HSS construction uses fewer bytes per client than the other constructions, since it only needs to send the shares. In the VAHSS-LHS construction, the client needs to receive the secret key and verification key, and the clients generate and share the proofs. In the VAHSS-TSS case, there is a significant reduction in the required transferred data. However, it is still higher than the required communication overhead in the VAHSS-HSS construction, since it needs to output a matrix and receive two big prime numbers.

Application scenarios: Mobile phones, wearables, and other Internet-of-Things (IoT) devices are all connected to distributed network systems. These devices generate a lot of data that often need to be aggregated to compute statistics, or even employed for user modeling and personalization of clients/users. For instance, a direct application of our constructions could be the measurement of electricity consumption (as well as water or gas consumption) in a specific region. More precisely, if we consider that each household in a specific region has an electricity consumption that is equal to x_i, then, by employing our proposed constructions, the electricity production company could check what is the required energy consumption (by computing the sum $\sum_{i=1}^{n} x_i$, where n denotes the number of households) in that specific region and adjust the electricity production accordingly. Furthermore, by employing the verifiability property, the electricity consumption company can detect possible leaks of faults in how the electrical energy is handled. In our prototype, we use the data on the consumption of electricity and, more precisely, the UC Irvine machine learning repository. Because this is one of the most representative settings, all three constructions are suitable to be employed in this application, depending on the resources of the employed sensors/clients in different households. If we consider that multiple companies are collaborating in this aggregation process, then the collected data could be aggregated by multiple servers that collaborate or not.

Another scenario, where our constructions could be suitable are health monitoring settings, where a health provider may want to measure the average physical activity levels or patient conditions (e.g., temperature) in specific regions in order to draw conclusions about the health conditions of parts of the population and accordingly adjust services. For instance, health insurance companies could offer discounts to families or neighborhoods, depending on their physical activity, while the verifiability property provides transparency and fairness guarantees regarding the provided services from the health insurance company (https://www.generaliglobalhealth.com/news/global-insights/insurtech/digital-technology-transforming-global-healthcare.html). Similarly, the proposed constructions would facilitate the aggregation process and collaboration between multiple hospitals that store confidential patient records and need to aggregate data in order to decide on the diagnosis and treatment of patients.

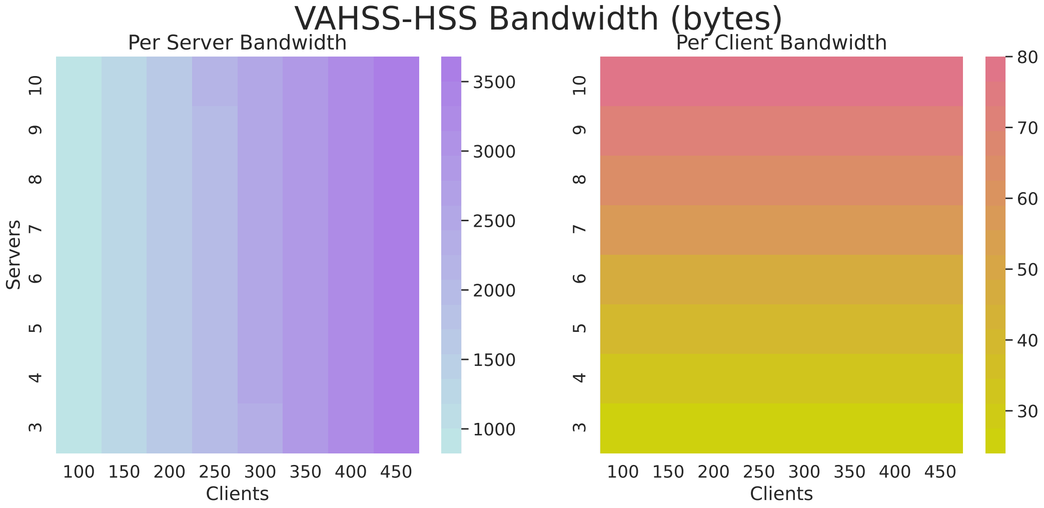


Figure 5. Bandwidth per serve and per client in bytes using VAHSS-HSS.

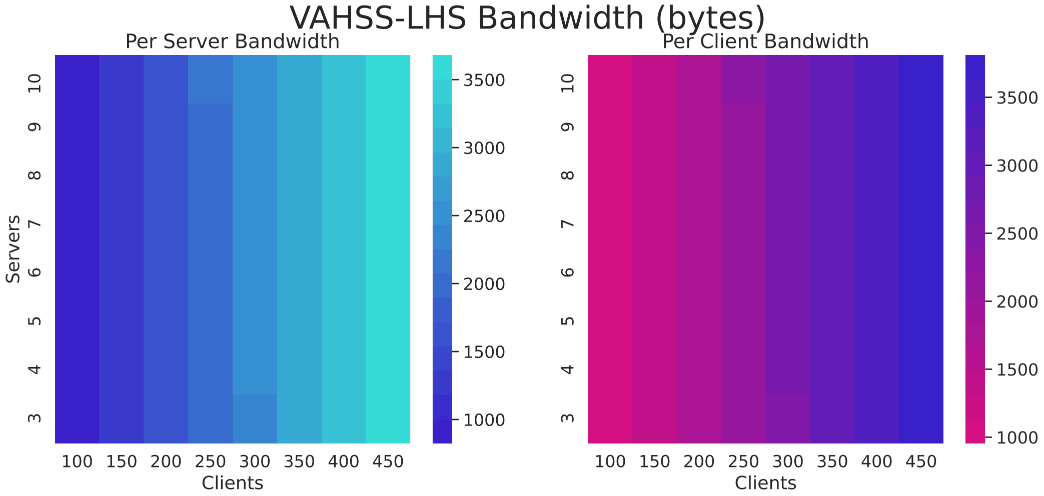


Figure 6. Bandwidth per serve and per client in bytes using VAHSS-LHS.

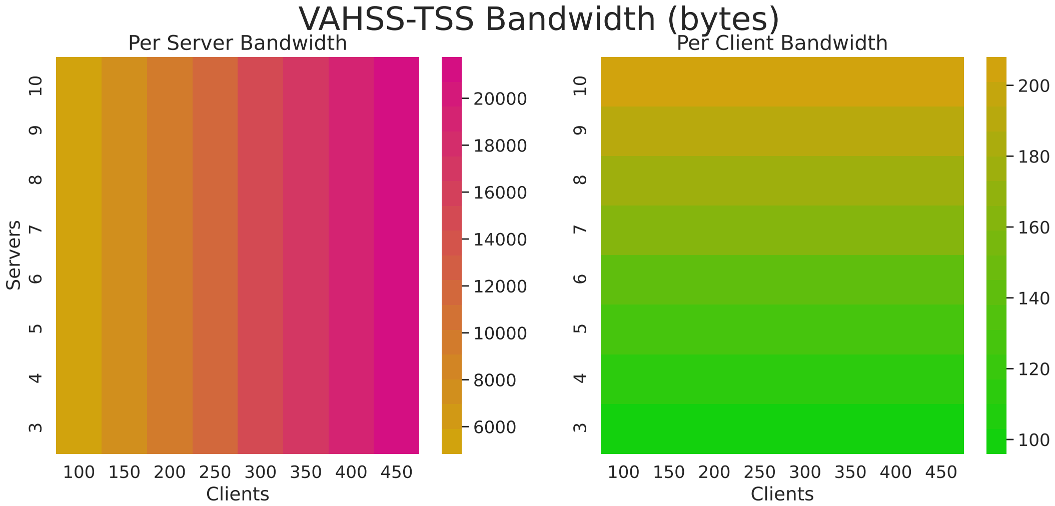


Figure 7. Bandwidth required per server and per client in bytes using VAHSS-TSS.

Besides, an important characteristic of our constructions is that the clients (from where the data are collected) do not need to communicate with each other. Thus, they could be easily employed in order to achieve reliable and verifiable environmental monitoring, when environmental sensors collecting appropriate measurements (e.g., temperature, humidity, CO_2, ozone, etc.) are spread in large regions and the sensors are not in the communication range of each other. More precisely, consider the setting where monitoring the air quality of specific neighborhoods in a city is needed. By employing sensors that are spread in large regions, data can be collected and then use the VAHSS-HSS construction to sum the measurement of CO_2. In this case, we do not rely only on one server, but on several servers to aggregate the measurements, while the verifiability property can be employed to guarantee the integrity of the aggregation process.

7. Conclusions

Major security and privacy challenges exist in the context of joint computations that are outsourced to untrusted cloud servers. Sensitive information of individuals might be leaked or malicious cloud servers might attempt to alter the aggregation results. In this work, we presented three concrete constructions for the verifiable additive homomorphic secret sharing (VAHSS) problem. We provided a solution based on homomorphic hash functions, a solution that uses linear homomorphic signatures and a construction based on a threshold signature sharing scheme. We proved all three constructions correct, secure, and verifiable. These constructions allow for any verifier to obtain the value y that is the sum of the clients' secret inputs and confirm its correctness; without compromising the clients' privacy or relying on trusted servers and without requiring any communication between the clients. We demonstrated the theoretical analysis of our work, showing the amount of computational cost that is required for each construction both from the clients' and servers' side. Subsequently, our experimental results illustrated how the different operations correspond to the required (computation) time with respect to the algorithms that are executed. Thus, the appropriate construction may be employed, depending on the available resources, requirements, and assumptions (e.g., communication between servers or no communication). We believe that our proposed VAHSS constructions can be employed in various applications that require the secure aggregation of data that were collected from multiple clients (e.g., smart metering, environmental monitoring, and health databases) and provide a practical and provably secure distributed solution, while avoiding single point of failures and any leakage of sensitive information.

Author Contributions: Conceptualization, G.T. and A.M.; implementations, G.B. and G.T.; writing—original draft preparation, G.T.; writing—review and editing, G.T., G.B. and A.M.; supervision, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Gustavo Banegas is funded by WASP expedition project Massive, Secure, and Low-Latency Connectivity for IoT Applications.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tsaloli, G.; Mitrokotsa, A. Sum It Up: Verifiable Additive Homomorphic Secret Sharing. In *Information Security and Cryptology—ICISC 2019*; Seo, J.H., Ed.; Springer International Publishing: Cham, Switzerland, 2020; pp. 115–132.
2. Tsaloli, G.; Liang, B.; Mitrokotsa, A. Verifiable Homomorphic Secret Sharing. In Proceedings of the 12th International Conference on Provable Security, ProvSec 2018, Jeju, Korea, 25–28 October 2018; pp. 40–55. [CrossRef]
3. Yao, H.; Wang, C.; Hai, B.; Zhu, S. Homomorphic Hash and Blockchain Based Authentication Key Exchange Protocol for Strangers. In Proceedings of the International Conference on Advanced Cloud and Big Data (CBD), Lanzhou, China, 12–15 August 2018; pp. 243–248. [CrossRef]
4. Krohn, M.; Freedman, M.; Mazieres, D. On-the-fly verification of rateless erasure codes for efficient content distribution. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 12 May 2004; pp. 226–240. [CrossRef]
5. Catalano, D.; Marcedone, A.; Puglisi, O. Authenticating Computation on Groups: New Homomorphic Primitives and Applications. In *Advances in Cryptology—ASIACRYPT 2014*; Sarkar, P., Iwata, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 193–212.
6. Bozkurt, İ.N.; Kaya, K.; Selçuk, A.A. Practical Threshold Signatures with Linear Secret Sharing Schemes. In *Progress in Cryptology—AFRICACRYPT 2009*; Preneel, B., Ed.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 167–178.
7. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
8. Boyle, E.; Gilboa, N.; Ishai, Y. Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation. In *Advances in Cryptology—EUROCRYPT 2017*; Springer International Publishing: Cham, Switzerland, 2017; Volume 10211, pp. 163–193._6. [CrossRef]
9. Benaloh, J.C. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Conference on the Theory and Application of Cryptographic Techniques*; Springer: Berlin, Germany, 1987.
10. Boyle, E.; Gilboa, N.; Ishai, Y. Function Secret Sharing. In *Advances in Cryptology—EUROCRYPT 2015*; Springer: Berlin, Germany, 2015; Volume 9057, pp. 337–367. [CrossRef]
11. 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—CCS'16, Vienna, Austria, 24–28 October 2016; pp. 1292–1303. [CrossRef]
12. Damgård, I.; Pastro, V.; Smart, N.; Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology—CRYPTO 2012*; Safavi-Naini, R., Canetti, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 643–662.
13. 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 *Computer Security—ESORICS 2013*; Crampton, J., Jajodia, S., Mayes, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 1–18.
14. Boyle, E.; Garg, S.; Jain, A.; Kalai, Y.T.; Sahai, A. Secure Computation against Adaptive Auxiliary Information. In *Advances in Cryptology—CRYPTO 2013*; Canetti, R., Garay, J.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 316–334.
15. Baum, C.; Damgård, I.; Orlandi, C. Publicly Auditable Secure Multi-Party Computation. In *Security and Cryptography for Networks*; Abdalla, M., De Prisco, R., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 175–196.

16. Catalano, D.; Fiore, D.; Warinschi, B. Efficient Network Coding Signatures in the Standard Model. In *Public Key Cryptography—PKC 2012*; Fischlin, M., Buchmann, J., Manulis, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 680–696.
17. Bellare, M.; Goldreich, O.; Goldwasser, S. Incremental Cryptography: The Case of Hashing and Signing. In *Advances in Cryptology—CRYPTO '94*; Desmedt, Y.G., Ed.; Springer: Berlin/Heidelberg, Germany, 1994; pp. 216–233.
18. Schabhüser, L.; Butin, D.; Buchmann, J. Context Hiding Multi-key Linearly Homomorphic Authenticators. In *Topics in Cryptology—CT-RSA 2019*; Matsui, M., Ed.; Springer International Publishing: Cham, Switzerland, 2019; pp. 493–513.
19. Dorn, W.S. Generalizations of Horner's rule for polynomial evaluation. *IBM J. Res. Dev.* **1962**, *6*, 239–245. [CrossRef]