Poster: Fully Homomorphic Secret Sharing with Output Verifiability

Arup Mondal
Ashoka University
arup.mondal_phd19@ashoka.edu.in

Pratyush Ranjan Tiwari
Johns Hopkins University
pratyush@cs.jhu.edu

Debayan Gupta
Ashoka University
debayan.gupta@ashoka.edu.in

***Abstract*—We study, motivate, and construct the first fully homomorphic secret sharing scheme with public verifiability. In our scheme, n clients secret-share data to m servers to outsource joint computation on private inputs. The ability to evaluate a variety of functions due to the fully homomorphic nature of the scheme increases possible application areas, and verifiability allows any user to check output correctness. We note that previous claims of achieving the same properties had major defects (they assumed clients could solve discrete log).**

I. Introduction and Preliminaries

We show an important catch in Tsaloli et al.'s *verifiable homomorphic secret sharing* (VHSS) [1]: the verifiability algorithms proposed for the multiplicative part of the scheme are actually the ones that would work for making the additive part verifiable. Specifically, [1] assumes that each participant p_i has private input x_i but also $\tilde{x}_i$ such that $g^{\tilde{x}_i} = x_i$, **which implies that every party can solve discrete log, but the adversary cannot**. The additive part of their scheme works for the addition of secrets whereas the multiplicative part only works for the multiplication of secrets, unless we can solve discrete log. Any protocol in such a setting is not meaningful as no combination of the two operations is possible. We detail a fresh construction of the VHSS scheme which satisfies the: Verifiability and Homomorphism. An ideal scheme would also achieve efficiency through batching of proofs; our current system, while correct, is inefficient, especially in terms of size.

Definition 1.1 (Homomorphic Secret Sharing): This is defined by the following tuple of PPT algorithms:

- $\mathsf{ShareSecret}(1^\lambda, \mathsf{i}, \mathsf{x_i})$: Pick polynomial p_i of the form $p_i(X) = x_i + a_1X + a_2X^2 + ... + a_tX^t$ where the coefficients a_i are selected uniformly at random from $\mathbb{F}$. And the degree t follows $t.n < m$. Output shares $(x_{i1}, x_{i2}, ..., x_{im})$.
- $\mathsf{PartialEval}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: For the input j, we must be able to compute for all $i \in [n]$ the sum or the product, shares of which, y^j are then outputted.
- $\mathsf{FinalEval}(\mathsf{y}^1, \mathsf{y}^2, ..., \mathsf{y}^\mathsf{m})$: Takes as input the partial evaluations and outputs their sum or product for the final evaluation, $y = y^1 + y^2 + ... + y^m$ or $y = y^1.y^2.....y^m$.

Definition 1.2 (Verifiable Homomorphic Secret Sharing): We add the following algorithms for verifiability, constructions of which are detailed later:

- $\mathsf{PartialProof}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: Proof share for j.
- $\mathsf{FinalEval}(\mathsf{y}^1, \mathsf{y}^2, ..., \mathsf{y}^\mathsf{m})$: Takes as input the partial evaluations and outputs their sum or product.
- $\mathsf{FinalProof}(\sigma^1, \sigma^2, ..., \sigma^\mathsf{m})$: Takes as input the partial proofs and outputs the final proof, $\sigma = \sigma^1.\sigma^2.....\sigma^m$.
- $\mathsf{Verify}(\sigma, \mathsf{y})$: Check proof validity, return true/false.

Definition 1.3 (Self-Bilinear Maps): Self-Bilinear maps are bilinear maps such that the groups in the domain of the map are identical to the output group. $\mathbb{G}$ is a cyclic group of order p. The generator of $\mathbb{G}$ is g. A self-bilinear map is $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$. A self-bilinear map can be used to construct multilinear maps by recursion as follows: $e_{n+1}(X_1, \ldots, X_n, X_{n+1}) := e(e_n(X_1, \ldots, X_n), X_{n+1})$

Self-Bilinear Maps from inversion hard rings: Self-bilinear maps from inversion hard rings are constructed in the paper[2]. Yamakawa et al. [2] use additive notation to describe there bilinear map. However, we use the multiplicative notation in our construction (see [2]). This is not a problem as conversion between the two notations is easy. The main issue with this construction self-bilinear maps is that there exists no candidate ring with efficiently computable group operations that satisfy c-inversion. *We note that in this paper, we do not attempt to find or describe a candidate ring with efficiently computable group operations that satisfy c-inversion.*

II. Verifiable Additive HSS

We start with a recent scheme [1] where multiple clients employ multiple servers to perform computation of a function on their joint input. There are n clients $\{c_1, c_2, ..., c_n\}$ who employ m servers $\{s_1, s_2, ..., s_m\}$ to calculate $f(x_1, x_2, ..., x_n)$ where $\{x_1, x_2, ..., x_n\}$ are the respective inputs of the n clients; corruption of upto t servers is tolerated. Such a scheme exists if and only if $m > n.t$. There are two main schemes: (1) an additive HSS scheme for adding n inputs and (2) a *Verifiable* Homomorphic Secret Sharing scheme that calculates f on n inputs; the function f calculates the product of the inputs. A VHSS scheme allows any user to confirm final output (rather than each share). However, **[1] assumes that each client c_i has private input x_i but also $\tilde{x}_i$ such that $g^{\tilde{x}_i} = x_i$**, which implies that every party can solve the discrete logarithm problem. The additive part of their scheme works for addition of secrets $\{x_1, x_2, ..., x_n\}$ whereas the multiplicative part **only** works for multiplication of secrets $\{g^{x_1}, g^{x_2}, ..., g^{x_n}\}$. Unless we can solve discrete log, no combination of the two operations is possible. The construction of our additive HSS scheme as follows:

For a finite field $\mathbb{F}$ with $|\mathbb{F}| > m$, security parameter λ, let $\theta_{i1}, ..., \theta_{im}$ be distinct field elements, for any $i \in \{n\}$ there exist $\lambda_{i1}, ..., \lambda_{im}$ such that over some univariate polynomial p_i of degree t over $\mathbb{F}$ we have $p_i(0) = \sum_{i=1}^{m} \lambda_{ij} p_i(\theta_{ij})$. Each client c_i distributes a share of their secret to each servers for the server to compute the partial sum y^i (for server s_i) following which any user can easily compute the final sum.

Definition 2.1 (Additive HSS): An additive HSS scheme is defined by the following tuple of PPT algorithms:

- $\mathsf{ShareSecret}(1^\lambda, \mathsf{i}, \mathsf{x_i})$: Pick polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + ... + a_t X^t$ where the coefficients a_i are selected uniformly at random from $\mathbb{F}$. And the degree t follows $t.n < m$.
  Output $(x_{i1}, x_{i2}, ..., x_{im}) = (\lambda_{i1}.p_i(\theta_{i1}), \lambda_{i2}.p_i(\theta_{i2}, ..., \lambda_{im}.p_i(\theta_{im}))$.
- $\mathsf{PartialEval}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: For the input j, compute for all $i \in [n]$ the sum $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$. Output $y^j = \sum_{i=1}^{n} \lambda_{ij} p_i(\theta_{ij})$.
- $\mathsf{FinalEval}(\mathsf{y^1}, \mathsf{y^2}, ..., \mathsf{y^m})$: Takes as input the partial evaluations and outputs their sum for the final evaluation, $y = y^1 + y^2 + ... + y^m$.

Definition 2.2 (Additive VHSS): An additive VHSS scheme is defined by the following tuple of PPT algorithms:

- $\mathsf{ShareSecret}(1^\lambda, \mathsf{i}, \mathsf{x_i})$: Pick polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + ... + a_t X^t$ where the coefficients a_i are selected uniformly at random from $\mathbb{F}$. And the degree t follows $t.n < m$. Output $(x_{i1}, x_{i2}, ..., x_{im}) = (\lambda_{i1}.p_i(\theta_{i1}), \lambda_{i2}.p_i(\theta_{i2}, ..., \lambda_{im}.p_i(\theta_{im}))$.
- $\mathsf{PartialEval}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: For the input j, compute for all $i \in [n]$ the sum $x_{ij} = \lambda_{ij} p_i(\theta_{ij})$. Output $y^j = \sum_{i=1}^{n} \lambda_{ij} p_i(\theta_{ij})$.
- $\mathsf{PartialProof}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: Share of proof for j; output $\sigma^j = e(g, g^{x_{1j}}.g^{x_{2j}}.....g^{x_{nj}}) = e(g, g^{y^j})$, where $e : \mathbb{G} \times \mathbb{G}. \to \mathbb{G}_k$
- $\mathsf{FinalEval}(\mathsf{y^1}, \mathsf{y^2}, ..., \mathsf{y^m})$: Takes partial evals and outputs their sum for the final eval, $y = y^1 + y^2 + ... + y^m$.
- $\mathsf{FinalProof}(\sigma^1, \sigma^2, ..., \sigma^m)$: Takes as input the partial proofs and outputs their product for the final proof, $\sigma = \sigma^1.\sigma^2.....\sigma^m$.
- $\mathsf{Verify}(\sigma, \mathsf{y})$: Check $\sigma = e(g, g^y)$; return 1(equal)/0(not).

Notice that the partial proof computes $\sigma^j = e(g, g^{x_{1j}}.g^{x_{2j}}.....g^{x_{nj}}) = e(g, g^{\Sigma_i x_{ij}})$, and $\Sigma_i x_{ij}$ is simply y^j. When these partial proofs are multiplied, we get the following product: $e(g, g^{y^1}) \times e(g, g^{y^2}) \times \cdots \times e(g, g^{y^m})$. By the basic properties of bilinear pairings, this equals $e(g, g)^{y^1} \times e(g, g)^{y^2} \times \cdots \times e(g, g)^{y^m} = e(g, g)^{\Sigma_i y^i} = e(g, g^{\Sigma_i y^i}) = e(g, g^y)$, as required.

III. Verifiable Multiplicative HSS

We present the scheme as in [1]. We plan to leverage multiplicative Beaver triples and our additive scheme to produce a verifiable, multiplicative framework. The setup for the multiplicative VHSS scheme is similar and uses bilinear pairings. Every time a computation is performed, a proof of it has to be published, and can be checked using Verify.

Definition 3.1 (Multiplicative HSS): A multiplicative HSS scheme is defined by the following PPTA:

- $\mathsf{ShareSecret}(1^\lambda, \mathsf{i}, \mathsf{x_i})$: Pick polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + ... + a_t X^t$ where the coefficients a_i are selected uniformly at random from $\mathbb{F}$, degree t follows $t.n < m$. Output $(x_{i1}, x_{i2}, ..., x_{im})$ such that $x_{i1}.x_{i2}....x_{im} = x_i$
- $\mathsf{PartialEval}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: For the input j, compute the product $x_{1j}.x_{2j}.. .x_{nj} = y^j$.
- $\mathsf{FinalEval}(\mathsf{y^1}, \mathsf{y^2}, ..., \mathsf{y^m})$: Takes as input the partial evaluations and outputs their product for the final evaluation, $y = y^1.y^2.....y^m$.

Definition 3.2 (Multiplicative VHSS): A multiplicative VHSS scheme is defined by the following PPTA:

- $\mathsf{ShareSecret}(1^\lambda, \mathsf{i}, \mathsf{x_i})$: Pick polynomial p_i of the form $p_i(X) = x_i + a_1 X + a_2 X^2 + ... + a_t X^t$ where the coefficients a_i are selected uniformly at random from $\mathbb{F}$, degree t follows $t.n < m$. Output $(x_{i1}, x_{i2}, ..., x_{im})$ such that $x_{i1}.x_{i2}....x_{im} = x_i$
- $\mathsf{PartialEval}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: For the input j, compute the product $x_{1j}.x_{2j}.. .x_{nj} = y^j$.
- $\mathsf{PartialProof}(\mathsf{j}, (\mathsf{x_{1j}}, \mathsf{x_{2j}}, ..., \mathsf{x_{nj}}))$: Share of proof for j; output $\sigma^j = e_n(g^{x_{1j}}, \ldots, g^{x_{nj}})$, where e_n is the multinear map constructed from the self-bilinear map $e : \mathbb{G} \times \mathbb{G}. \to \mathbb{G}$.
- $\mathsf{FinalEval}(\mathsf{y^1}, \mathsf{y^2}, ..., \mathsf{y^m})$: Takes as input the partial evaluations and outputs their product for the final evaluation, $y = y^1.y^2.....y^m$.
- $\mathsf{FinalProof}(\sigma^1, \sigma^2, ..., \sigma^m)$: Takes as input the partial proofs and outputs their product for the final proof, $\sigma = e_m(\sigma^1.\sigma^2.....\sigma^m)$. where e_m is the multinear map constructed from the self-bilinear map $e : \mathbb{G} \times \mathbb{G}. \to \mathbb{G}$.
- $\mathsf{Verify}(\sigma, \mathsf{y})$: Check if $\sigma = e(g, g^y)$ return 1 if both are equal, else return 0.

A summary of the above scheme is that secret x is split as $\prod_{i=1}^{n} x_i$ then these x_i's are split over m different server's as $x_i = x_{i1}.x_{i2}.....x_{im}$. As a result what we get is this matrix $m \times n$ matrix where the ij'th element is x_{ij} such that $x = \prod_{j=1}^{m} \prod_{i=1}^{n} x_{ij} = \prod_{i=1}^{n} \prod_{j=1}^{m} x_{ij}$. These products are checked in the exponent using the multilinear map construction via self-bilinear maps.

References

[1] G. Tsaloli, B. Liang, and A. Mitrokotsa, "Verifiable homomorphic secret sharing," in *Provable Security - 12th International Conference, ProvSec 2018, Jeju, South Korea, October 25-28, 2018, Proceedings*, 2018, pp. 40–55.

[2] T. Yamakawa, S. Yamada, G. Hanaoka, and N. Kunihiro, "Generic hardness of inversion on ring and its relation to self-bilinear map," *Theor. Comput. Sci.*, vol. 820, pp. 60–84, 2020.

Poster: Fully Homomorphic Secret Sharing with Output Verifiability

Arup Mondal[1], Pratyush Ranjan Tiwari[2], Debayan Gupta[1]

[1]Ashoka University, [2]Johns Hopkins University

arup.mondal_phd19@ashoka.edu.in, pratyush@cs.jhu.edu, debayan.gupta@ashoka.edu.in

Objectives

We study, motivate, and construct the first fully homomorphic secret sharing scheme with public verifiability. Figure 1 present a schematic overview of the problem of *outsourcing computations*, where n participants secret-share data to m servers to outsource joint computation on joint inputs with output *verification*.

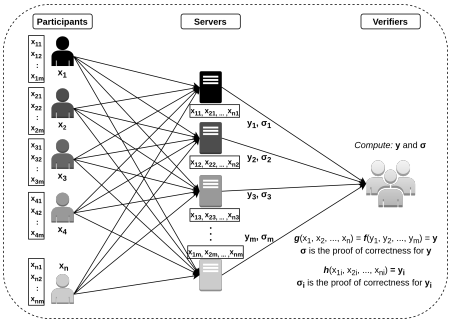

Figure 1: n participants compute the joint function g (addition or multiplication) of their joint inputs to m servers with publicly output verifiable.

We detail a fresh construction of the verifiable homomorphic secret sharing scheme which actually satisfies the requirements, discussing two essential properties::

- **Homomorphism:** Arbitrary computations on shared inputs based on local computations on their shares.
- **Verifiability:** Ensure any participants can efficiently confirm and verify the final output (rather than each share) is correct.

Self-Bilinear Maps

Self-Bilinear maps are bilinear maps such that the groups in the domain of the map are identical to the output group. $\mathbb{G}$ is a cyclic group of order p. The generator of $\mathbb{G}$ is g. A self-bilinear map is $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$. A self-bilinear map can be used to construct multilinear maps by recursion as follows: $e_m(X_1, \ldots, X_{m-1}, X_m) := e(e_{m-1}(X_1, \ldots, X_{m-1}), X_m)$.

Additive Verifiable Homomorphic Secret Sharing

Participants with Secrets	Servers S_1	S_2	$\ldots$	S_m
x_1	x_{11}	x_{12}	$\ldots$	x_{1m}
x_2	x_{21}	x_{22}	$\ldots$	x_{2m}
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
x_n	x_{n1}	x_{n2}	$\ldots$	x_{nm}
Partial Eval (Addition)	$y^1 = \sum_{i=1}^{n} x_{i1}$	$y^2 = \sum_{i=1}^{n} x_{i2}$	$\ldots$	$y^m = \sum_{i=1}^{n} x_{im}$
Partial Proof	$\sigma^1 = e(g, g^{y^1})$	$\sigma^2 = e(g, g^{y^2})$	$\ldots$	$\sigma^m = e(g, g^{y^m})$
Final Eval (Addition)	$y = \sum_{i=1}^{m} y^i$			
Final Proof	$\sigma = \sum_{i=1}^{m} \sigma^i$			
Verify	$\sigma = e(g, g^{\sigma})$			

Table 1: Additive Verifiable Homomorphic Secret Sharing Construction and Functionalities.

Notice that the partial proof (in Table 1) computes $\sigma^j = e(g, g^{x_{1j}}.g^{x_{2j}}.....g^{x_{nj}}) = e(g, g^{\Sigma_i x_{ij}})$, and $\Sigma_i x_{ij}$ is simply y^j. When these partial proofs are multiplied, we get the following product: $e(g, g^{y^1}) \times e(g, g^{y^2}) \times \cdots \times e(g, g^{y^m})$. By the basic properties of bilinear pairings, this equals $e(g, g)^{y^1} \times e(g, g)^{y^2} \times \cdots \times e(g, g)^{y^m} = e(g, g)^{\Sigma_i y^i} = e(g, g^{\Sigma_i y^i}) = e(g, g^y)$, as required.

Multiplicative Verifiable Homomorphic Secret Sharing

Participants with Secrets	Servers S_1	S_2	$\ldots$	S_m
x_1	x_{11}	x_{12}	$\ldots$	x_{1m}
x_2	x_{21}	x_{22}	$\ldots$	x_{2m}
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
x_n	x_{n1}	x_{n2}	$\ldots$	x_{nm}
Partial Eval (Multiplication)	$y^1 = \prod_{i=1}^{n} x_{i1}$	$y^2 = \prod_{i=1}^{n} x_{i2}$	$\ldots$	$y^m = \prod_{i=1}^{n} x_{im}$
Partial Proof	σ^1	σ^2	$\ldots$	σ^m
Final Eval (Multiplication)	$y = \prod_{i=1}^{m} y^i$			
Final Proof	$\sigma = e_m(\sigma^1, \ldots, \sigma^m)$			
Verify	$\sigma = e(g, g^{\sigma})$			

Table 2: Multiplicative Verifiable Homomorphic Secret Sharing Construction and Functionalities.

A summary of the above scheme (in Table 2) is that secret x is split as $\prod_{i=1}^{n} x_i$ then these x_i's are split over m different server's as $x_i = x_{i1}.x_{i2}.\ldots.x_{im}$. As a result what we get is this matrix $m \times n$ matrix where the ij'th element is x_{ij} such that $x = \prod_{j=1}^{m} \prod_{i=1}^{n} x_{ij} = \prod_{i=1}^{n} \prod_{j=1}^{m} x_{ij}$. These products are checked in the exponent using the multilinear map construction via self-bilinear maps.

Applications with Asymptotic Complexity Comparison

Privacy-Preserving Machine Learning: The proposed primitives provide the following three features over existing schemes:

- Privacy and correctness against the semi-honest corruption of any subset of servers and participants.
- Privacy against any t-malicious server (t is the threshold of a threshold secret sharing scheme).
- Detect the manipulation of the inference results with short evidence (using output verifiability).

Paper	Threat Model		No. of Server(s)	Privacy Guarantee		Comms.	Output Verify
	Participants	Servers		Computation	Output		
[2]	dishonest	1-server dishonest	3	yes	yes	2 round	no
This Work	dishonest	t-server dishonest	m	yes	yes	1 round	yes

Table 3: Comparison between a secret-shared privacy-preserving neural network training framework and what would happen if VHSS was used instead.

Privacy-Preserving Federated Learning: The proposed primitives can enable an improved federated learning (FL) framework by introducing multiple FL aggregators to design a distributed network of FL aggregators $\mathcal{A}$, where each $\mathcal{A}$ aggregates each participant's secret-shared model parameters and produces a proof of the computation. This reduces the risk of a malicious aggregator by distributing the task.

Paper	Threat Model		Decentralized FL Server(s)	Privacy Guarantee		Comms.	Output Verify
	Participants	FL servers		Compute	Output		
[1]	dishonest	semi-honest	No	yes	yes	3 round	no
This Work	dishonest	t-server dishonest	Yes	yes	yes	1 round	yes

Table 4: Comparison between a secret-shared privacy-preserving federated learning framework and what would happen if VHSS was used instead.

References

[1] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[2] Prashanthi Ramachandran, Shivam Agarwal, Arup Mondal, Aastha Shah, and Debayan Gupta. S++: A fast and deployable secure-computation framework for privacy-preserving neural network training. *arXiv preprint arXiv:2101.12078*, 2021.

[3] Georgia Tsaloli, Bei Liang, and Aikaterini Mitrokotsa. Verifiable homomorphic secret sharing. In *International Conference on Provable Security*, pages 40–55. Springer, 2018.