Practical SNARGs for Matrix Multiplications over Encrypted Data

Louis Tremblay Thibault[1,2], Michael Walter[1], Jiapeng Zhang[3]

[1]Zama, [2]École de technologie supérieure, [3]University of Southern California

Problem Statement

Motivation: SotA vFHE is too expensive. Can we build practical vHE for specific yet meaningful computations? Yes: matrix-vector multiplication!

Goal: Efficiently verifiable and actively secure outsourced computation of $\mathsf{y} = \mathsf{Ax}$

Setting: Plaintext space $\mathbb{Z}_p$, public matrix $\mathsf{A} \in \mathbb{Z}_p^{n \times m}$, private vector $\mathsf{x} \in \mathbb{Z}_p^m$

Key criterion for practicality:
Client time (Encryption + Verification + Decryption) < **Plaintext** O(mn) **computation**

Otherwise, there is no incentive to outsource!
Applications: Actively secure ML inference on sensitive user data.

Homomorphic Computation

Setting: prime ciphertext modulus $q > p$, $R_p = \mathbb{Z}_p[X]/\langle X^N + 1\rangle \subset R_q = \mathbb{F}_q[X]/\langle X^N + 1\rangle$

Reduction: Matrix-vector multiplication $\xrightarrow{(1)}$ Inner products $\xrightarrow{(2)}$ Ring multiplications $\xrightarrow{}$ GLWE-based homomorphic ring multiplications

Observation (folklore): For $\mathsf{x}, \mathsf{y} \in \mathbb{Z}_p^N$ and $x, y \in R_p$ their ring representation, the constant term of $z = x \cdot y$ can be written as

$$z_0 = \langle \mathsf{x}, (y_0, -y_{N-1}, \ldots, -y_1)\rangle$$

Define encoding $\tau : \mathbb{Z}_p^N \to R_q$ such that $\langle \mathsf{x}, \mathsf{y}\rangle = (x \cdot \tau(\mathsf{y}))_0$ over R_q. Then,

$$(\mathsf{Ax})_i = \langle \tau(\mathsf{A}_i), \mathsf{x}\rangle \quad (1)$$

GLWE-based encryption of the private vector $\mathsf{x} \in \mathbb{Z}_p^m$:
- Additively homomorphic;
- Allows for multiplication by small constants in R_p (encoded rows of A).

Protocol flow:
1. User encrypts $\mathsf{x} \in \mathbb{Z}_p^m$ into $t = m/N$ GLWE ciphertexts c;
2. Server computes $\mathsf{c}'_i = \langle \tau(\mathsf{A}_i), \mathsf{c}\rangle$ (multiplication by constants in R_p, sum of ciphertexts); (2)
3. User sample extracts and decrypts.

Optimizations:
- Ciphertext compression: PRNG used to derive ciphertext masks.
- Sample extraction: decryption takes $\mathcal{O}(kN)$ instead of $\mathcal{O}(kN \log N)$ per output component.

Adding Verifiability with Ring Switching (ePrint 2025/199)

Reduction: Statement over polynomial ring $R_q \xrightarrow{(1)}$ Statement over polynomial ring $\mathbb{F}_q[X] \xrightarrow{(2)}$ Statement over extension field $\mathbb{F}_{q^\gamma}$

(1): Ring Embedding
- Embed $\mathsf{Mx} = \mathsf{y}$ over R_q to $\mathbb{F}_q[X]$
- Introduce remainder vector r:

$$\mathsf{Mx} = \mathsf{y} + (X^N + 1) \cdot \mathsf{r}$$

(2): Ring Reduction
- Choose appropriate γ for soundness
- Map $\mathbb{F}_q[X]$ to $\mathbb{F}_{q^\gamma}$ via polynomial evaluation at random $\alpha \in \mathbb{F}_{q^\gamma}$
- Statement over $\mathbb{F}_{q^\gamma}$ may be proven with standard SNARG techniques (Sum-Check)

Key Takeaways

- First verifiable homomorphic encryption protocol where **client time < plaintext computation** at scale.
- SNARK-FHE paradigm ensure security **against malicious adversaries**.
- Ring switching allows to **bridge the algebraic structure gap** between Field-based SNARGs and Ring-based HE schemes.
- Open-source prototype implementation in Rust.
- Applications to privacy-preserving machine learning.

Full paper
ia.cr/2026/027

Repository

SNARK-FHE vs. FHE-SNARK

SNARK-FHE Paradigm: Prove correctness of FHE computation
FHE-SNARK Paradigm: Homomorphically compute SNARK proof

	Verifier	Security model
SNARK-FHE	Public	Active
FHE-SNARK	Designated	Passive

- FHE-SNARK exhibits better performances than SNARK-FHE but does not provide stronger privacy guarantees than plain FHE.
- Our technique yields practical performance in the stronger SNARK-FHE paradigm.

Implementation & Parameters

Stack: Rust, WHIR PCS (black-box), Fiat-Shamir with BLAKE3
Parameters: (≈100 bits of security via Lattice Estimator):

q	N	k	Standard deviation	p
$2^{64} - 2^{32} + 1$	2^{10}	1	2^{33}	2^4

Note: Protocol is PCS-agnostic — faster PCS implies better performances.

Experimental Results

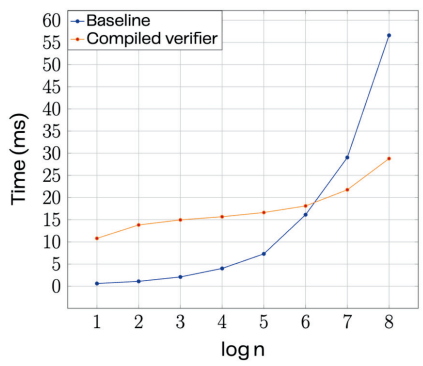
Prover time (seconds). PCS dominates (>50% of time):

log t \ log n		2	4	6	8
8	**w/ PCS**	0.57	1.80	6.69	24.96
	w/o PCS	0.24	0.64	2.31	9.40
10	**w/ PCS**	1.56	4.63	24.80	101.55
	w/o PCS	0.81	2.41	9.62	45.06

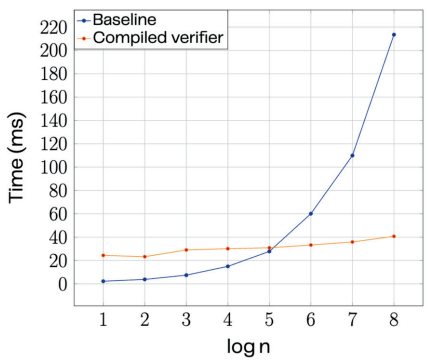
Proof size (kB). PCS dominates (>97%)

q	2	4	6	8
8	183	204	230	244
10	188	210	234	250

Key Result: Verifier beats plaintext computation.



Timing for $t = 2^8$ across n values



Timing for $t = 2^{10}$ across n values

Crossover at log t = 7, log n = 8 — prover only takes seconds in this regime.

Communication:
- Input ciphertext: 16 kB – 8 MB (linear in matrix width)
- Output ciphertext: 32 kB – 4 MB (linear in matrix height)
- Proof: 183 – 250 kB (logarithmic in matrix size)

Resources

zama.org/blog
github.com/zama-ai