SNARGs for Encrypted Matrix Multiplications and Connections to Verifiable PIR

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

[1] Zama, Paris, France
[2] École de technologie supérieure, Montréal, Canada
[3] University of Southern California, Los Angeles, USA

Abstract. Fully Homomorphic Encryption (FHE) enables computations to be performed directly on encrypted data, without ever requiring decryption. This capability is particularly crucial for privacy-preserving outsourced computation in sensitive fields such as healthcare and finance. While FHE ensures data confidentiality under the honest-but-curious adversarial model, achieving full malicious security, encompassing both integrity and privacy, requires an additional layer of verifiability. To address this, a growing body of research has explored combining FHE with techniques from verifiable computation, leading to the notion of verifiable FHE (vFHE). However, the integration of these two paradigms often results in substantial computational overhead, making existing approaches largely impractical for real-world deployment.

Although there are many approaches, there remain major challenges that make the efficiency of vFHE impractical. Among these barriers, one of the most severe challenges is the algebraic gap between cyclotomic rings, which are mainly used in FHE encryption, and finite fields, which are mainly used by SNARGs. In this paper, we rely on a novel solution called ring switching to bridge this gap, making significant progress toward efficient vFHE. To highlight the efficiency of ring switching, we provide an open-source implementation of its application to homomorphic matrix-vector multiplication. Experimental results demonstrate high efficiency.

Keywords: SNARGs · Verifiable Homomorphic Computation · Privacy-preserving delegation of computation · PIR

1 Introduction

Fully Homomorphic Encryption (FHE) is powered by its ability to perform computations directly on encrypted data, thereby eliminating the need for decryption during the computation. This property is highly desirable for privacy-preserving outsourced computation, especially over data containing sensitive information. While FHE ensures confidentiality against honest-but-curious adversaries, achieving full malicious security, which requires both integrity and privacy, demands an additional verifiability layer. Consequently, recent research has increasingly explored the combination of FHE with verifiable computation techniques, leading to the emergence of verifiable Fully Homomorphic Encryption (vFHE) [FG12, dCL24, TW25, ABPS24, HLMZ24, CCC+25, LLZ+25, ZLW+25].

The main framework of vFHE is that, in addition to returning the ciphertext of the computational outcome, the server is also required to generate a proof demonstrating that the ciphertext was obtained through an honest execution of the designated homomorphic computation.

E-mail: louis.tremblay.thibault@gmail.com (Louis Tremblay Thibault), michael.walter@zama.ai (Michael Walter), jiapengz@usc.edu (Jiapeng Zhang)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-11-09.

In current vFHE constructions, the main tool used to achieve efficient verifiability is the *Succinct Non-Interactive Argument* (SNARG). Although this problem has been extensively studied, integrating these two powerful paradigms still incurs significant computational overhead, rendering existing solutions largely impractical for real-world deployment.

To make a verifiable homomorphic computation practical, several factors are important to consider:

1. The server's running time, including both homomorphic computation and proof generation, should be efficient in practice.

2. The proof size should be compact.

3. Most importantly, the client's running time, including encryption, decryption and verification, should be less than running the computation over plaintext by themselves. Otherwise, there is no incentive to use FHE.

A large body of prior work has focused on optimizing the proving time of general FHE schemes (e.g., [TW25, ABPS24, CCC+25, LLZ+25]). However, we do not see many works that compare their verification time with the plaintext computation time. See also [KVMH24].

In fact, it is very challenging to achieve all three of these points for general fully homomorphic computation. Rather than aiming for a general but impractical vFHE scheme, we focus on the following question:

Can we realize verifiable homomorphic computation for some specific yet meaningful computation?

In this paper, we study a meaningful and widely used computation: matrix–vector multiplication, and present a new practical verifiable homomorphic encryption scheme for it. Our scheme allows to answer this question by the affirmative.

Matrix-vector multiplication. Let $\mathbf{A} \in \mathbb{Z}_l^{n \times m}$ and $\mathbf{x} \in \mathbb{Z}_l^m$, where $\mathbf{A}$ is public and $\mathbf{x}$ is private and $l \in \mathbb{N}$. The goal is to securely outsource the computation of $\mathbf{y} \leftarrow \mathbf{Ax}$. In this direction, we emphasize that matrix–vector multiplication is one of the most widely used arithmetic computations, with numerous applications, particularly in machine learning and private information retrieval. For example, many fundamental machine learning algorithms, including PCA, linear regression, and SVMs, rely heavily on repeated matrix–vector multiplications as their core computational primitive. Even in large language models (LLMs), matrix–vector multiplication plays a crucial role. As a consequence, privacy-preserving matrix–vector multiplication has also been extensively studied in prior works [LS18, LZ22, GQH+24, MMJG24, MK25]. Notably, our work is also inspired by [FBK+25], which introduces an interactive protocol for private LoRA (Low-Rank Adaption) fine-tuning of (public) LLMs. The authors remark that the protocol is also useful for inference on private query vectors. The key idea of their protocol is to outsource only the heavy linear computation of the fine-tuning and process the lighter non-linear part locally. The linear computation, i.e. the multiplication of a public matrix and a private vector, is performed using an additively homomorphic encryption scheme. A major weakness of the system is the amount of trust that must be put into the server. More precisely, the protocol needs to assume that the server is at most honest-but-curious, meaning that it follows the protocol honestly but may attempt to learn information from the protocol execution. Under this assumption, correctness is obvious and security of the interactive protocol follows from the work of [AGHV22], since it is *cleartext computable*. Adding verifiability to the outsourced matrix-vector multiplication protocol provides correctness and privacy in a stronger security model, where the server may be fully malicious.

1.1 Existing Approaches and Challenges

Homomorphic multiplication for matrices and vectors may appear to be a simple and solved problem at first glance, as it involves only linear operations, and there are several existing works on privacy-preserving matrix multiplication. However, to make it applicable to broader applications such as blockchains, public verifiability with sublinear verification time is crucial. Although verifiable FHE schemes have been extensively studied, several major efficiency bottlenecks remain. One of the key obstacles, as highlighted by [ZWL+25a] (the full version of [ZWL+25b]), is the algebraic structure gap barrier.

Algebraic structure gap. Typically, sublinear verification time has been achieved using standard, well-established approaches that generate SNARG proofs for homomorphic computations. We note that the majority of current SNARG techniques are built over finite fields.

However, to ensure efficient homomorphic computation, Ring-LWE is generally a more practical assumption than plain LWE. In this regime, all homomorphic computations are carried out over polynomial rings.

The algebraic structure gap refers to the fact that directly proving ring arithmetic using only finite-field techniques is usually very inefficient.

To build efficient proof systems for rings, there are two main existing approaches. The first direction mainly follows the Rinocchio paper [GNSV23]. In this direction, Rinocchio introduced Quadratic Ring Programs (QRPs), an adaptation of Quadratic Arithmetic Programs (QAPs) to the ring setting. It aims to prove the correctness of homomorphic computation directly over rings via QRPs. However, it has two main drawbacks:

- It requires a trusted setup, i.e., it is not publicly verifiable. Such a limitation is not favorable for some applications.
- Its concrete performance is poor. According to the benchmark presented by [SXZ+25], the proving time on the server could be 1000 times slower than the homomorphic computation itself.

The second approach to proving directly over lattices is the work of LaBRADOR [BS23] and LatticeFold+ [BC25]. Here, LaBRADOR is a lattice-based SNARG and LatticeFold+ is a folding scheme. This approach achieves efficient proving time. For example, LaBRADOR has been applied in many constructions such as polynomial commitment scheme [NS24], aggregate signatures [AAB+24] and threshold ring signatures [JAHT25]. However, it still has several limitations.

- The constructions are usually efficient only for small rings, such as polynomial rings of dimension 64. However, most homomorphic encryption schemes require computations over large rings in order to bridge efficiency and security. This gap makes them less applicable for generating SNARG proofs for homomorphic encryption.
- Both LaBRADOR and LatticeFold+ only natively support non-NTT-friendly prime moduli. These are not typical choices for the majority of lattice-based protocols, as they make the arithmetic much slower.
- Their verification time is not optimal. For example, it is still unknown how to make the verification time of LaBRADOR sublinear. Thus, it does not satisfy the efficiency criterion that we are interested in.

FHE-SNARK paradigm and its limitations. Other than designing efficient proof systems for ring arithmetic direction, there is another alternative way to avoid the algebraic

structural gap mentioned above. Recent work has explored a new approach known as "FHE-SNARK", which consists in performing integrity checks in the plaintext space rather than over ciphertexts [GGW24, ACGS24, CKP+23, GBK+25]. While some of these schemes demonstrate practical efficiency, this paradigm suffers from a fundamental limitation in that it necessitates decryption prior to verification. This design choice introduces two critical drawbacks as already highlighted in [TW25, ZWL+25b, PLFT25]. First, it precludes public verifiability, as only parties possessing the secret key can verify correctness. Although this may suffice for certain use cases, it excludes many applications where third-party or public verification is essential. Second, and more critically, it exposes the FHE scheme to active attacks. As is demonstrated in [MN24, TW25], ciphertext-level integrity checks enable stronger security guarantees which are not achieved by schemes following the FHE-SNARK paradigm. In fact, such schemes remain vulnerable to active attacks exposed in [CGG16]. This disparity is exemplified by the protocol from [FBK+25]. As mentioned above, applying our approach to the basic protocol strengthens the security model and achieves security against fully malicious servers. It can be verified that applying the FHE-SNARK paradigm fails to do so and still requires an honest-but-curious assumption about the server, which only marginally improves over the basic protocol presented in [FBK+25].

Connections to Publicly Verifiable Private Information Retrieval. Private Information Retrieval (PIR) [CGKS95] is a cryptographic primitive that allows a client to retrieve an element from a server-held database without revealing to the server which element was retrieved.

Suppose that a server holds a database represented as a matrix whose columns encode database entries. Using a linearly homomorphic encryption scheme, one can construct a PIR protocol between a client and a server as follows. The client encrypts the i-th basis vector $\mathbf{e}_i$ and sends the resulting ciphertext to the server. The server exploits the linear homomorphism of the encryption scheme to multiply the database matrix by the encrypted vector, thereby obtaining an encryption of the i-th column of the matrix. The server returns the resulting ciphertext, which the client decrypts to recover the requested database element in the clear. By the semantic security of the homomorphic encryption scheme, the server learns nothing about which element was queried by the client.

Under this approach, PIR reduces directly to the problem of matrix multiplication over encrypted data. However, such PIR solutions are typically secure only in the honest-but-curious model, since active adversaries may mount reaction attacks that compromise client privacy. To address this limitation, a recent line of work [dCL24, RLP26] has considered verifiable PIR as a means of achieving active security. However, those works introduce caveats: [dCL24] proposes a solution where a costly preprocessing phase must be run every time malicious server behavior is detected, and [RLP26] follows the FHE-SNARK paradigm and therefore inherit some of its limitations. Our SNARG-based solution addresses these limitations as the preprocessing is run only once and the verification procedure is independent of the user query, thereby thwarting reaction attacks. As such, we obtain a practical, one-time preprocessing, publicly verifiable PIR protocol. This is, to the best of our knowledge, the first instantiation of a PIR protocol with such strong security guarantees. We overview the performance of our protocol in the PIR setting in Section 5.

1.2 Our New Solution

As previously discussed, the main challenge here is the mathematical gap between polynomial rings $\mathfrak{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ and finite fields. To address this gap, we adopt a new approach called the ring-switching idea introduced by Huang, Mao and Zhang [HMZ25]. The core idea of ring switching is to replace the ring $\mathfrak{R}_q$ over which the statement to prove

is defined with an appropriately chosen extension field of $\mathbb{Z}_q$ of smaller size. At a high-level, ring switching proceeds in two steps:

1. Ring embedding. Firstly, an appropriate extension field $\mathbb{F}_q$ of $\mathbb{Z}_q$ is chosen. The size of $\mathbb{F}_q$ is related to the security parameter of the SNARG proof. For example, q is a 64-bit prime in our implementation, we then choose an extension degree of 2 to achieve 100-bit security. Once the extension field has been fixed, we then embed the statement $\mathbf{Mx} = \mathbf{y}$ over $\mathfrak{R}_q$ into a statement over $\mathbb{F}_q[X]$.

2. Ring reduction. The second step of ring switching is to reduce the statement, via a polynomial commitment scheme over the extension field, from $\mathbb{F}_q[X]$ to $\mathbb{F}_q$. This ring reduction step runs in linear time, after which we can perform a standard SNARG proof over the extension field. We also note that the statement size becomes much smaller after ring reduction, since each $\mathfrak{R}_q$ element is compressed into an $\mathbb{F}_q$ element.

In this paper, we use a ring-based encryption scheme that is friendly to the ring switching idea. We also implement our protocol, and the experimental result demonstrates that our scheme is highly efficient. From the experiment, we clearly see that our client time, including both decryption and verification time, is much faster than plaintext computation for larger scale input (i.e., matrices having at least 2^{27} entries over the plaintext space). We are not aware of any other vFHE scheme that achieves this.

Our Contributions Our contributions may be summarized as follows.

1. We devise a concretely efficient, publicly verifiable plaintext matrix–encrypted vector multiplication protocol.

2. We show how this protocol yields one-time preprocessing, publicly verifiable PIR.

3. We evaluate the performance of our protocols and provide an open-source implementation.

2 Preliminaries and Notations

Let $\mathbb{Z}$ denote the set of integers, and let $\mathbb{Z}_q$ denote the finite field of size q, where q is a prime. For a distribution D, we write $x \leftarrow D$ to denote sampling an element from D. We use $\mathfrak{R}_q$ to denote the polynomial ring $\mathbb{Z}_q[X]/(X^N + 1)$.

2.1 GLWE and Additively Homomorphic Encryption

Definition 1. Let $N, \kappa, q \in \mathbb{Z}$ with N a power of 2 and χ_s and χ_e be two distributions over $\mathfrak{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$. For $\mathbf{s} \in \mathfrak{R}_q^\kappa$, the *General LWE (GLWE) distribution* is defined as $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathfrak{R}_q^{\kappa+1}$, where $\mathbf{e} \leftarrow \chi_e$ and $\mathbf{a} \leftarrow \mathcal{U}(\mathfrak{R}_q^\kappa)$. The *GLWE assumption* states that it is hard to distinguish the GLWE distribution from $\mathcal{U}(\mathfrak{R}_q^{\kappa+1})$ when $\mathbf{s} \leftarrow \chi_s^\kappa$.

The hardness of the GLWE assumption with varying parameters can be estimated concretely using [APS15]. Let $p < q$ and $\Delta = \lfloor q/p \rceil$. Based on the GLWE assumption, we can define the following secret key encryption scheme with message space $\mathfrak{R}_p$, which goes back to [Reg05].

The scheme in Algorithm 1 is secure based on the GLWE assumption. The scheme is correct, as long as $\|e/\Delta\|_\infty \leq 1/2$. It is easy to see that the scheme is additively homomorphic, i.e.

$$\mathrm{Dec}_{\mathbf{s}}\left(\sum_i \mathrm{Enc}_{\mathbf{s}}(m_i)\right) = \sum_i m_i$$

Algorithm 1: A GLWE-based Encryption Scheme. The parameters N, κ, q, Δ, χ_s, χ_e are public and available to all algorithms.

Gen :
  $\mathbf{s} \leftarrow \chi_s^\kappa$
  return $\mathbf{s}$
$\mathrm{Enc}_{\mathbf{s}}(m)$:
  $\mathbf{a} \leftarrow \mathcal{U}(\mathfrak{R}_q^\kappa)$
  $e \leftarrow \chi_e$
  return $\mathbf{c} = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + \Delta m)$
$\mathrm{Dec}_{\mathbf{s}}(\mathbf{c} = (\mathbf{a}, b))$:
  $m \leftarrow \lfloor (b - \langle \mathbf{a}, \mathbf{s} \rangle)/\Delta \rceil$
  return m

as long as parameters are such that $\|(\sum_i e_i)/\Delta\|_\infty \leq 1/2$. Ciphertexts can be multiplied with small polynomials, i.e.

$$\mathrm{Dec}_{\mathbf{s}}\left(m_1 \cdot \mathrm{Enc}_{\mathbf{s}}(m_2)\right) = m_1 \cdot m_2$$

as long as parameters are such that $\|(m_1 \cdot e_2)/\Delta\|_\infty \leq 1/2$.

2.2 Polynomial Commitment Scheme

Let us recall the definition of a Polynomial Commitment Scheme (PCS). A PCS over a field $\mathbb{F}$ is a quadruple of algorithms $(\mathrm{Gen}, \mathrm{Com}, \mathrm{Eval}, \mathrm{Verify})$ such that:

- $\mathrm{Gen}(1^\lambda) \mapsto \mathsf{pp}$ takes as input a security parameter and outputs public parameters pp.
- $\mathrm{Com}(\mathsf{pp}, f) \mapsto c$ takes as input the public parameters and a polynomial $f \in \mathbb{F}^{\leq d}[X]$ and outputs a commitment c to f.
- $\mathrm{Eval}(\mathsf{pp}, f, c, x, y) \mapsto \pi$ takes as input the public parameters pp, the polynomial f and its commitment c as well as an evaluation point x and a claimed evaluation y of f at x. The algorithm outputs a proof π that $f(x) = y$.
- $\mathrm{Verify}(\mathsf{pp}, c, x, y, \pi) \mapsto \{\mathsf{acc}/\mathsf{rej}\}$ takes in the public parameters pp, a commitment c, an evaluation point x, a claimed evaluation y and an evaluation proof π and either accepts or rejects.

A PCS must observe the following properties:

- **Completeness.** Let $\mathsf{pp} \leftarrow \mathrm{Gen}(1^\lambda)$. For any polynomial $f \in \mathbb{F}^{\leq d}[X]$ and all domain elements x, let $\mathsf{pp} \leftarrow \mathrm{Gen}(1^\lambda)$, $c \leftarrow \mathrm{Com}(\mathsf{pp}, f)$, $y = f(x)$ and $\pi \leftarrow \mathrm{Eval}(\mathsf{pp}, f, c, x, y)$. The Verify algorithm must accept with probability 1.
- **Evaluation binding.** Let $\mathsf{pp} \leftarrow \mathrm{Gen}(1^\lambda)$. For all f and for any $y \neq y'$ and π, π', Verify accepts both $(\mathsf{pp}, c, x, y, \pi)$ and $(\mathsf{pp}, x, c, y', \pi')$ with negligible probability.

2.3 Succinct Non-Interactive Argument

Now we recall the definition of Succint Non-Interactive Argument (SNARG).

Definition 2 (SNARG)**.** A succinct non-interactive argument $\Pi = (\mathrm{Gen}, \mathrm{Prove}, \mathrm{Verify})$ for an NP relation R is a tuple of algorithms with the following syntax.

- $\mathrm{Gen}(1^\lambda) \mapsto \mathsf{crs}$: Takes as input a security parameter $\lambda \in \mathbb{N}$ and outputs a common reference string crs.
- $\mathrm{Prove}(\mathsf{crs}, x, w) \mapsto \pi$: Takes as input common reference string crs, a statement x, and a witness w, and outputs a proof π when $(x, w) \in R$.
- $\mathrm{Verify}(\mathsf{crs}, x, \pi) \mapsto \{\mathsf{acc}, \mathsf{rej}\}$: Takes as input common reference string crs, a statement x, and a proof π, and outputs either acc (accept) or rej (reject).

The following security properties are required.

- **Completeness.** For all input $(x, w) \in R$, let $\mathsf{crs} \leftarrow \mathrm{Gen}(1^\lambda)$ and $\pi \leftarrow \mathrm{Prove}(\mathsf{crs}, x, w)$. The verifier accepts (crs, x, π) with probability 1.
- **Soundness.** For all $(x, w) \notin R$, and for any (crs, π) generated by a (possibly malicious) polynomial-time prover, the verifier accepts (crs, x, π) only with negligible probability.
- **Succinctness.** We say that the SNARG is succinct if the running time of Verify and the proof size are in $\mathtt{poly}(\lambda, |x|, \log |w|)$.

2.4 Public Verifiable Homomorphic Matrix Multiplication

Now we explain our setting for publicly verifiable homomorphic matrix multiplication. It involves several steps.

1. **Preprocessing stage**:
   (a) The client first performs some preprocessing for a publicly known matrix $\mathbf{A}$ and generates the encryption keys.
2. **Computation stage**:
   (a) For a plaintext vector $\mathbf{x}$, the client encrypts it as $\mathbf{c}$ and sends $\mathbf{c}$ to the server.
   (b) The server homomorphically computes $\mathbf{Ax}$ and returns $\mathbf{c}'$ together with a succinct proof π.
   (c) The client verifies the proof and decrypts $\mathbf{c}'$.

Hereafter, we require several properties from the protocol:

- **Correctness.** If the server executes the whole protocol honestly, then $\mathrm{Dec}(\mathbf{c}') = \mathbf{Ax}$ and the client accepts the proof π.
- **Soundness.** If the server deviates from the prescribed protocol, then the client rejects the proof with overwhelming probability.

Additionally, our goal for **succinctness** is to ensure that the client's online time, i.e. encryption, proof verification and decryption, is faster than computing $\mathbf{Ax}$ directly in the plaintext space.

3 Our Protocol

Now we give a formal description of our protocol in this section. Similar to standard vFHE, there are two parts in our protocol. The first part leverages a homomorphic encryption scheme to perform the computation over encrypted data. The second part is to generate a succinct proof to prove that the prescribed homomorphic computation has been correctly executed.

3.1 Homomorphic Computation

Let $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ and $\mathbf{x} \in \mathbb{Z}_p^m$. We now present a way to leverage the scheme from Algorithm 1 to perform the computation homomorphically on the encryption of $\mathbf{x}$. Crucially, the evaluation algorithm will only use arithmetic operations in $\mathfrak{R}_q$ and thus be suitable to be combined with a SNARG for operations in $\mathfrak{R}_q$, as demonstrated in Section 3.2.

The algorithm is a sub-algorithm from the one presented in [FBK+25], from which we strip all non-arithmetic operations to enable efficient verifiability. The key observation (already in [HLHD22]) is that for two polynomials $x = \sum_i x_i X^i \in \mathfrak{R}_p$ and $y = \sum_i y_i X^i \in \mathfrak{R}_p$ the constant term of $z = x \cdot y$ can be written as

$$\begin{aligned} z_0 &= x_0 \cdot y_0 - \sum_{i,j:i+j=N} x_i \cdot y_j \\ &= \langle \mathbf{x}, (y_0, -y_{N-1}, -y_{N-2}, \ldots, -y_1) \rangle \end{aligned}$$

So let $\tau : \mathbb{Z}_p^N \mapsto \mathfrak{R}_q$ be the encoding $\tau(\mathbf{y}) = y_0 - \sum_{i=1}^{N-1} y_{N-i} X^i$. Then $\langle \mathbf{y}, \mathbf{x} \rangle = (\tau(\mathbf{y}) \cdot x)_0$. Let $m > N$ and assume that $N \mid m$ (otherwise simply pad with zeros) and denote $t = m/N$. We extend $\tau : \mathbb{Z}_p^m \mapsto \mathfrak{R}_p^t$ to vectors of size m by applying it to blocks of length N and concatenating the result into a vector of size t over $\mathfrak{R}_p$. By also encoding the message $\mathbf{x} \in \mathbb{Z}_p^m$ into t elements $\mathbf{x} = (x_1, x_2, \ldots, x_t) \in \mathfrak{R}_p^t$ in the obvious way, we can extend the inner product computation to

$$\langle \mathbf{y}, \mathbf{x} \rangle = \langle \tau(\mathbf{y}), \mathbf{x} \rangle_0 \quad .$$

The algorithm now follows naturally and is given in Algorithm 2. We assume the user has already run Gen to generate a secret key $\mathbf{s}$.

Algorithm 2: An Algorithm for Homomorphic Matrix-Vector Multiplication

User on input $\mathbf{x} \in \mathbb{Z}_p^m$:
$\mathbf{y} \leftarrow (\sum_{j=0}^{N} x_{i \cdot N + j} \cdot X^j)_{i=0}^{t-1} \in \mathfrak{R}_q^t$
foreach $y_i \in \mathbf{y}$ **do**
  $\mathbf{c}_i \leftarrow \mathrm{Enc}_{\mathbf{s}}(y_i)$
end
send $\mathbf{c} = (\mathbf{c}_i)_{i=0}^{t-1}$ to the server

Server on input $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ and ciphertext vector $\mathbf{c}$:
foreach *row* $\mathbf{a}_i \in \mathbb{Z}_p^m$ *of* $\mathbf{A}$ **do**
  $\mathbf{c}'_i \leftarrow \langle \tau(\mathbf{a}_i), \mathbf{c} \rangle$
end
send $\mathbf{c}' = (\mathbf{c}'_i)_{i=0}^{n-1}$ to the user

User on input $\mathbf{c}'$:
foreach $\mathbf{c}'_i$ **do**
  $z_i \leftarrow \mathrm{Dec}_{\mathbf{s}}(\mathbf{c}'_i)_0$
end
return $\mathbf{z} = (z_1, \ldots, z_n)$

Correctness Let $\mathbf{c}'_i = (\mathbf{a}'_i, b'_i) \in \mathfrak{R}_q^{\kappa+1}$ and $e'_i = b'_i - \langle \mathbf{a}'_i, \mathbf{s} \rangle \in \mathfrak{R}_q$ be the corresponding noise polynomial. Note that we have $e'_i = \langle \tau(\mathbf{a}_i), \mathbf{e} \rangle$, where $\mathbf{e} \leftarrow \chi_e^t$ is the vector of noise polynomials in $\mathbf{c}$. For correctness, we require $\|e'_i\|_\infty = \|\langle \tau(\mathbf{a}_i), \mathbf{e} \rangle\|_\infty \leq \Delta/2$ for all i. Let

σ^2 be the variance of χ_e. Then, the variance of each coordinate of $\langle \tau(\mathbf{a}_i), \mathbf{e} \rangle$ is $\leq m \cdot p^2 \cdot \sigma^2$. Heuristically assuming that each coordinate of $\langle \tau(\mathbf{a}_i), \mathbf{e} \rangle$ follows a Gaussian, we have that

$$\Pr[\|\langle \tau(\mathbf{a}_i), \mathbf{e} \rangle\|_\infty \geq r \cdot \sqrt{m} \cdot p \cdot \sigma] \leq 2\exp(-r^2/2)/\sqrt{2\pi r^2}$$

which is less than 2^{-130} for $r > 13.2$. So, if we set parameters such that $r \cdot \sqrt{m} \cdot p \cdot \sigma \leq \Delta/2$ for reasonable r, the protocol is correct with overwhelming probability. Equivalently, we require $q/\sigma \gtrsim 2r\sqrt{m}p^2$. We can use the lattice estimator [APS15] to find suitable k and N for any desired security level.

Optimizations When sending many (G)LWE-based ciphertexts, a common technique to compress them is to not choose the $\mathbf{a}$ elements of the ciphertexts uniformly at random. Instead, a seed is chosen and the values $\mathbf{a}$ are all derived using a PRNG. Clearly, this is applicable here for the ciphertexts sent by the user (as pointed out in [FBK+25]).

Note that the result of the multiplication is stored in the constant coefficients of the plaintexts encrypted in $\mathbf{c}'_i$. So, the user does not need to decrypt the entire plaintext. Partially decrypting the constant coefficient can be done by first performing *sample extraction*, which is a technique that is folklore in the homomorphic encryption literature. Specifically, decryption requires computing $\langle \mathbf{a}, \mathbf{s} \rangle$, while all other operations are componentwise. It can be verified that $\langle \mathbf{a}, \mathbf{s} \rangle_0 = \langle \tau^{-1}(\mathbf{a}), \bar{\mathbf{s}} \rangle$, where $\bar{\mathbf{s}} \in \mathbb{Z}_q^{\kappa N}$ is the coefficient vector of $\mathbf{s}$. Accordingly, the constant coefficient can be decrypted as $z_i \leftarrow \lfloor (b'_0 - \langle \tau^{-1}(\mathbf{a}'), \bar{\mathbf{s}} \rangle)/\Delta \rceil$ where $\mathbf{c}'_i = (\mathbf{a}', b')$. This reduces decryption time from $O(\kappa N \log N)$ to $O(\kappa N)$ per coordinate of $\mathbf{z}$.

Performance The user performs $t = m/N$ encryptions, each of cost $O(\kappa N \log N)$ for a total cost of $O(mk \log N)$, and n decryptions for a total cost of $O(n\kappa N)$ (using the optimization described above). Thus, for $\kappa \log N = o(n)$ and $\kappa N = o(m)$, this yields a non-trivial solution for the user.

The server performs $m \cdot t \cdot \kappa$ polynomial multiplications in the ring $\mathfrak{R}_q$ for a total cost of $O(mn\kappa \log N)$.

The user sends t ciphertexts containing N elements of $\mathbb{Z}_q$ each (using the compression technique outlined above), for a total number of m elements in $\mathbb{Z}_q$ and a short seed. The server sends n ciphertexts for a total number of $n \cdot (\kappa + 1) \cdot N$ element in $\mathbb{Z}_q$.

We note that some space is wasted when the server returns ciphertexts of which the client only needs to extract the constant coefficient. The work of [FBK+25] performs sample extraction on the server side and then re-pack the resulting LWE ciphertexts into RLWE ciphertexts, compressing them to n/N RLWE ciphertexts. However, this re-packing involves operations that are not purely arithmetic over $\mathfrak{R}_q$ and thus are hard to reconcile with our SNARG for $\mathfrak{R}_q$ in Section 3.2.

3.2 Efficient SNARG Proofs for Ring Operations

The Protocol For simplicity, we describe the protocol for a matrix-vector multiplication of the form

$$\mathbf{y} \leftarrow \mathbf{M}\mathbf{x} \in \mathfrak{R}_q^n$$

where $\mathbf{M}$ is the encoding of the matrix $\mathbf{A}$. We later show how to extend this to the setting described in Section 3.1 where the input to the multiplication is a vector of GLWE ciphertexts.

In order to prove integrity, the prover is essentially required to generate a proof for the statement $\mathbf{M}\mathbf{x} = \mathbf{y}$. In this regard, SNARGs appear to be a natural solution to this problem. However, as mentioned before, the main challenge is that $\mathbf{M}\mathbf{x} = \mathbf{y}$ is computed over the polynomial ring $\mathfrak{R}_q$, which is usually large (thus involving more expensive arithmetic

operations) and does not have a large exceptional set,[1] preventing standard SNARG techniques from being applied.

We now explain how ring switching can address this challenge. The high-level idea of ring switching is to transfer the statement from $\mathfrak{R}_q$ to an extension field of $\mathbb{Z}_q$, which we denote by $\mathbb{F}_q$. Ring switching consists of two steps. The first step lifts the statement over $\mathfrak{R}_q$ to a statement over $\mathbb{F}_q[X]$. To achieve this, we first observe that $\mathbf{M}, \mathbf{x}, \mathbf{y}$ can be naturally considered as elements of $(\mathbb{Z}_q[X])^{n\times m}, (\mathbb{Z}_q[X])^m$, and $(\mathbb{Z}_q[X])^n$, respectively; that is, each entry of $\mathbf{M}, \mathbf{x}, \mathbf{y}$ is a polynomial in $\mathbb{Z}_q[X]$. Furthermore, we can assume that the degree of each polynomial is at most $N-1$, since $\mathfrak{R}_q = \mathbb{Z}_q[X]/(X^N+1)$. Based on this embedding, we know that $\mathbf{Mx} = \mathbf{y}$ over $\mathfrak{R}_q$ if there exists a vector $\mathbf{r} \in \mathbb{Z}_q[X]$ with each entry being a polynomial of degree at most N such that

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

Now we have converted the arithmetic structure from $\mathfrak{R}_q$ to $\mathbb{Z}_q[X]$. Next, we choose an extension field of $\mathbb{Z}_q$, denoted by $\mathbb{F}_q$, and embed the statements into $\mathbb{F}_q[X]$. The purpose of this embedding is to enable the application of sumcheck [LFKN92] protocols later. In lattice-based constructions, the native field $\mathbb{Z}_q$ is often too small to preserve soundness in sumcheck protocols. To conclude the ring embedding step, it is sufficient for the prover to prove that:

- $\mathbf{M}, \mathbf{x}, \mathbf{y}, \mathbf{r}$ are over $\mathbb{Z}_q[X]$, and
- $\mathbf{Mx} = \mathbf{y} + (X^N + 1) \cdot \mathbf{r}$ over $\mathbb{F}_q[X]$

Here, the prover has to additionally prove the first item in order to avoid soundness errors caused by the ring embedding. However, this part can actually be done by the verifier itself, since the verifier knows $\mathbf{x}$, and $\mathbf{y}$, and has a commitment to $\mathbf{M}$ that guarantees that $\mathbf{M}$ is over $\mathbb{Z}_q$. If the prover sends any $\mathbf{r}$ that is not in $\mathbb{Z}_q[X]$, then the second condition will not be satisfied. After this ring embedding step, we can use a multilinear extension to represent the statements, enabling a SNARG-style proof. Here, the multilinear extension is a commonly used step in constructing PIOP proofs; we refer the reader to [HMZ25] for details.

We recall that $\mathbf{M}$ is now a matrix in $(\mathbb{F}_q[X])^{n\times m}$. Hence, each entry $\mathbf{M}_{i,j}$ of $\mathbf{M}$ has the form,

$$\mathbf{M}_{i,j} = \sum_{k<N} (\mathbf{M}_{i,j})_k \cdot X^k$$

where $(\mathbf{M}_{i,j})_k$ are the coefficients of $M_{i,j}$. Now let $\widetilde{\mathbf{M}}$ be the multilinear polynomial such that $\widetilde{\mathbf{M}}(i,j,k) = (\mathbf{M}_{i,j})_k$ for all $i \in [n], j \in [m]$ and $k \in [N]$. Here we slightly abuse the notation, as the variables of $\widetilde{\mathbf{M}}$ actually represent the binary representation of i, j, and k. Similarly, we can define $\widetilde{\mathbf{x}}$, $\widetilde{\mathbf{y}}$, and $\widetilde{\mathbf{r}}$. Now it remains for the prover to show that for every $i \in [n]$,

$$\begin{aligned}&\sum_j \left(\sum_k \widetilde{\mathbf{M}}(i,j,k)X^k\right)\left(\sum_k \widetilde{\mathbf{x}}(j,k)X^k\right)\\ =&\sum_k \widetilde{\mathbf{y}}(i,k)X^k + \sum_k \widetilde{\mathbf{r}}(i,k)X^k \cdot (X^N+1)\end{aligned}$$

For each i, we define a degree-$2(N-1)$ univariate polynomial $F_i(X)$ over $\mathbb{F}_q$ by

[1] Large exceptional sets are necessary for soundness requirements.

$$F_i(X) := \sum_j \left(\sum_k \widetilde{\mathbf{M}}(i,j,k) X^k \right) \left(\sum_k \widetilde{\mathbf{x}}(j,k) X^k \right) - \sum_k \widetilde{\mathbf{y}}(i,k) X^k + \sum_k \widetilde{\mathbf{r}}(i,k) X^k \cdot (X^N + 1)$$

Then it is essentially equivalent to proving that $F_i(X) \equiv 0$. Here, we can use a standard polynomial zero test: the verifier randomly sends a point $\alpha \in \mathbb{F}_q$, and the prover then proves that for all $i \in [n]$, $F_i(\alpha) = 0$. The soundness error is upper bounded by $2N/|\mathbb{F}_q| = O(N/\mathbb{F}_q)$. Notice that there are n constraints here, as the prover needs to prove that $F_i(\alpha) = 0$ for all $i \in [n]$. We then use a standard batching technique to combine them. Define a $(\log n)$-variable polynomial $f(t)$ to batch all the constraints.

$$f(t) = \sum_i \widetilde{\mathbf{eq}}(t,i) F_i(\alpha)$$

This is the unique multilinear polynomial that encodes the n evaluations of $F_i(\alpha)$. As such, in order to prove that $F_i(\alpha) = 0$ for all $i \in [n]$, it suffices to prove that $f(t)$ is the zero polynomial. Note that $f(t)$ is a polynomial over $\mathbb{F}_q$, and the ring-switching process has now been completed. By the Schwartz-Zippel lemma, for a random $\tau \xleftarrow{\$} \mathbb{F}_q^{\log n}$, $f(\tau) = 0$ implies with high probability that $f(t)$ is the zero polynomial. The verifier can thus sample such a random τ, send it to the prover who can then use the standard sumcheck protocol over the finite field to prove that

$$f(\tau) = 0. \tag{1}$$

To do so, we first expand the components of $f(\tau)$. We define $\widetilde{\ell}(\mathbf{x})$ to be the multilinear extension of α^k for all $k \in [N]$. Let

$$\begin{aligned}
\zeta_1 &:= \sum_i \widetilde{\mathbf{eq}}(\tau,i) \sum_j \left(\sum_k \widetilde{\mathbf{M}}(i,j,k) \widetilde{\ell}(k) \right) \widetilde{\mathbf{x}}_\alpha(j) \\
\zeta_2 &:= \sum_i \widetilde{\mathbf{eq}}(\tau,i) \sum_k \widetilde{\mathbf{y}}_\alpha(i) \\
\zeta_3 &:= \sum_i \widetilde{\mathbf{eq}}(\tau,i) \sum_k \widetilde{\mathbf{r}}(i,k) \widetilde{\ell}(k)
\end{aligned}$$

The prover computes ζ_1 and ζ_3 and sends these values to the verifier, who computes ζ_2 by itself and checks that

$$\zeta_1 - \zeta_2 - \zeta_3 \cdot (\alpha^N + 1) = 0$$

which implies that Equation (1) holds, provided the values sent by the prover are well-formed. Let

$$\begin{aligned}
g(i,j,k) &:= \widetilde{\mathbf{eq}}(\tau,i) \cdot \widetilde{\mathbf{M}}(i,j,k) \cdot \widetilde{\mathbf{x}}_\alpha(j) \cdot \widetilde{\ell}(k) \\
h(i,k) &:= \widetilde{\mathbf{eq}}(\tau,i) \cdot \widetilde{\mathbf{r}}(i) \cdot \widetilde{\ell}(k)
\end{aligned}$$

To convince the verifier that ζ_1 and ζ_3 are well-formed, the prover and verifier run the sumcheck protocol twice, once for

$$\sum_{i,j,k} g(i,j,k) = \zeta_1 \tag{2}$$

and once for

$$\sum_{i,k} h(i,k) = \zeta_3. \tag{3}$$

The sumcheck protocol reduces the validity of Equations (2) and (3) to checking that $g(\rho_1) = v_1$ and $h(\rho_2) = v_2$ where ρ_1, ρ_2 are vectors of random values sampled from the sumcheck challenge space by the verifier throughout the execution of the sumcheck protocol and v_1, v_2 are values determined by the execution of the protocol. For ζ_1, the verifier can perform this check efficiently by computing by itself

$$\begin{aligned}\nu_1 &\coloneqq \widetilde{\mathbf{eq}}(\tau, \rho_1[..\log n])\\ \nu_2 &\coloneqq \widetilde{\mathbf{x}}_\alpha(\rho_1[\log n..\log nm])\\ \nu_3 &\coloneqq \widetilde{\ell}(\rho_1[\log nm..])\end{aligned}$$

and checking that

$$v_1 = \nu_1 \cdot \mu \cdot \nu_2 \cdot \nu_3$$

holds, where μ denotes the evaluation of $\widetilde{\mathbf{M}}(\rho_1)$ provided by the prover using a polynomial commitment scheme. If this last check holds, then the verifier is convinced that the value ζ_1 provided by the prover is well-formed. The prover and verifier proceed analogously to ensure that Equation (3) holds, this time with $\widetilde{\mathbf{r}}(\rho_2)$ being provided by the PCS.

It is worth noting that the value ζ_2 is not subject to this technique because the verifier needs to know $\mathbf{y}$ in full for decryption. This is not the case for $\mathbf{r}$, so the overall protocol proof size benefits from including only a commitment and evaluation proof for $\widetilde{\mathbf{r}}$, instead of the full vector $\mathbf{r}$. The verifier can compute ζ_2 efficiently by itself.

Algorithm 3: An Efficient SNARG over Rings

Server and **User** preprocess the matrix $\mathbf{M}$;
Server commits to the polynomial $\widetilde{\mathbf{M}}$;

User sends $\mathbf{x}$ to the server;

Server sends $\mathbf{y}$ and a commitment to $\widetilde{\mathbf{r}}$ to the user;

Server and **User** run the sumcheck protocol to prove that $f(\tau) = 0$;

User makes one query to $\widetilde{\mathbf{M}}$ and one query to $\widetilde{\mathbf{r}}$ and accepts or rejects.

Optimizations The above protocol is defined for a matrix-vector product $\mathbf{y} = \mathbf{Mx}$ over $\mathfrak{R}_q$. However, the homomorphic matrix-vector multiplication algorithm defined in Algorithm 2 multiplies a plaintext matrix with a vector of GLWE ciphertexts encrypting $\mathbf{x}$. As described in Section 2.1, such ciphertexts are vectors of dimension $\kappa + 1$ over $\mathfrak{R}_q$. As such, the protocol described above only allows us to prove the correct execution of matrix-vector multiplication for a *single* component of the resulting GLWE ciphertexts. A naive way to extend the above protocol is to simply repeat it $\kappa + 1$ times, once for each component of the GLWE ciphertexts. A more efficient manner of extending the protocol to the desired functionality is to require the verifier to sample a supplementary random challenge β and have the prover and verifier run the above protocol for

$$\begin{aligned}&\sum_j \left(\sum_k \widetilde{\mathbf{M}}(i,j,k)X^k\right)\left(\sum_{\iota=1}^{\kappa+1}\beta^\iota\left(\sum_k \widetilde{\mathbf{x}}_\iota(j,k)X^k\right)\right)\\ =&\sum_{\iota=1}^{\kappa+1}\beta^\iota\left(\sum_k \widetilde{\mathbf{y}}_\iota(i,k)X^k + \sum_k \widetilde{\mathbf{r}}_\iota(i,k)X^k\cdot(X^N+1)\right)\end{aligned}$$

for all $i \in [n]$.

Analysis of the SNARG Proof We now analyze both the running time and the security of the SNARG resulting from the standard compilation of our PIOP and the WHIR PCS using Fiat-Shamir [BFS20].

The prover's running time consists of a PIOP (polynomial interactive oracle proof) and a polynomial commitment evaluation. The PIOP component mainly involves a sumcheck protocol, which runs in linear time, while the polynomial commitment incurs a quasi-linear time cost. Note that the matrix has size $O(nm)$, the PCS running time is $O(nm \log(nm))$. For the verification part, PCS has logarithmic running time but the PIOP has time linear in the size of the input and output. Therefore,

- The prover's running time is $O(nm \log(nm))$;
- The verifier's running time is $O(m + nN)$;
- The proof size $O(\log(nm))$.

We note that both our PIOP and most PCS protocols have perfect completeness, and hence the SNARG also has perfect completeness (if combined with a perfectly complete PCS). For soundness, the PIOP mainly relies on polynomial testing, which by the Schwartz-Zippel lemma yields a soundness error of

$$O(N/|\mathbb{F}_q| + \log(n)/|\mathbb{F}_q| + \kappa/|\mathbb{F}_q|).$$

Overall, by choosing a proper extension field size, we can achieve a negligible soundness error. In our implementation, we choose q to be a 64-bit prime and $\mathbb{F}_q$ is a degree-2 extension field. Overall, this gives 100-bit security.

4 Experimental analysis

We implement the above protocol in Rust to measure its concrete performance and open-source the implementation[2]. Our prototype confirms the hypothesis of Section 3.1 according to which our overall protocol's verifier is concretely faster than the standard $O(mn)$ matrix-vector multiplication algorithm over $\mathbb{Z}_p$.

4.1 Implementation description

Our prototype implements the protocol as described Section 3.2, including all optimizations discussed in this work. We render the protocol non-interactive via the Fiat-Shamir transform using BLAKE3 as the hash function [OANWO21]. The protocol relies on a polynomial commitment scheme in a black-box manner. In other words, it can be instantiated with any PCS. For performance and ease of use purposes, we choose in our implementation to instantiate the PCS with the WHIR polynomial commitment scheme [ACFY25] set to target 100 bits of security. Naturally, this means that our implementation directly inherits WHIR's performance. It is worth noting that our protocol can readily be adapted to forthcoming polynomial commitment schemes that perform better than WHIR.

We use the parameters from Table 1 which allow us to claim approximately 100 bits of security for the encryption scheme given in Section 2.1 according to the well-known Lattice Estimator [APS15], all the while maintaining correctness of the results. Letting $\mathbb{Z}_q$ be our base prime field and $\mathbb{F}_q$ be the quadratic extension field of $\mathbb{Z}_q$, we are able to claim 100 bits of security for the SNARG according to the discussion in the previous section.

Table 1: Parameters

q	N	κ	$\log_2 \sigma$	p
$2^{64} - 2^{32} + 1$	1024	1	33	2^4

4.2 Experiments

For each experiment, we run the process at least 10 times and take the median measurement as the representative. The machine used to run the experiments is an AWS EC2 instance of type `Hpc7a.24xlarge` equipped with 48 CPU cores and 768 GB of memory. To model a computationally powerful prover and a computationally bounded verifier, we design the prover algorithm to be parallel and the verifier to be single-threaded.

Prover time Table 2 provides the prover time as a function of the matrix parameters $t := m/N$ and n. The prover times are measured both with and without the PCS computation in order to clearly present the performance of our SNARG. The time taken by the prover with the PCS disabled is nearly always less than half that taken by the prover with the PCS enabled. This indicates that our protocol's prover time is actually dominated by the PCS prover time. Hence, given a PCS exhibiting better performances than WHIR, we can expect our protocol's online computation time to significantly decrease.

Verifier time The overall protocol verifier time is taken as the sum of:

- Time taken to encrypt $\mathbf{x}$;
- Time taken to verify the SNARG proof;
- Time taken to decrypt $\mathbf{c}'$.

We call this value "compiled verifier time". To model a computationally bounded verifier, we restrict the verifier algorithms to be single-threaded, i.e. no parallel computation takes place. Figures 1 to 3 provide comparisons between the compiled verifier time and the baseline $O(mn)$ plaintext matrix-vector multiplication algorithm over $\mathbb{Z}_p$, again for matrix parameters $t := m/N$ and n. To provide a fair comparison, the baseline plaintext matrix-vector multiplication is also computed sequentially. There is no clear bottleneck for the verifier time, as different steps contribute differently to the compiled verifier time depending on the parameters. Our experiments show that our verifier starts to outperform the baseline $O(mn)$ plaintext matrix-vector multiplication algorithm when $m = 2^{18}$ and $n = 2^8$. The SNARG prover takes less than 24 seconds to execute in this parameter regime, which demonstrates the practicality of our protocol.

Communication costs The SNARG proof size is taken as the sum of:

- The sumcheck proof sizes for the ζ_1 and ζ_3 claims;
- The WHIR evaluation proof sizes for $\widetilde{\mathbf{M}}(\rho_1)$ and $\widetilde{\mathbf{r}}(\rho_2)$.

Table 3 presents the proof sizes as a function of t and n. Our results exhibit the very succinct proof sizes resulting from our protocol. Furthermore, the WHIR evaluation proofs account for more than 97% of the total proof size. Given a PCS with shorter proofs, one can expect the proof sizes of our overall protocol to decrease drastically.

Recent work [DG25, CS25] suggests that conjectures invoked for performance purposes in WHIR may not hold. If these results are correct, WHIR proof sizes may be subject to a $\approx 4\times$ blowup.

[2]The code is available at:
https://anonymous.4open.science/r/matrx-vector-multiplication

Table 2: Prover times (in seconds). The rows with a ✓indicate measurements with the PCS enabled, while the rows with a ✗indicate measurements without the PCS.

$\log n$ \ $\log t$		1	2	3	4	5	6	7	8	9	10
1	✓	0.04	0.05	0.06	0.08	0.12	0.17	0.24	0.37	0.59	1.07
	✗	0.01	0.02	0.02	0.03	0.04	0.07	0.10	0.16	0.29	0.55
2	✓	0.06	0.07	0.09	0.12	0.17	0.24	0.35	0.52	0.95	1.53
	✗	0.02	0.02	0.03	0.04	0.06	0.09	0.13	0.24	0.42	0.82
3	✓	0.08	0.10	0.13	0.18	0.25	0.35	0.52	0.88	1.43	2.68
	✗	0.02	0.03	0.04	0.06	0.08	0.13	0.20	0.37	0.69	1.38
4	✓	0.11	0.14	0.20	0.25	0.36	0.50	0.86	1.38	2.52	4.52
	✗	0.03	0.04	0.06	0.08	0.13	0.20	0.34	0.61	1.23	2.55
5	✓	0.17	0.22	0.27	0.38	0.52	0.86	1.35	2.49	4.38	8.74
	✗	0.05	0.06	0.08	0.13	0.20	0.33	0.61	1.19	2.45	4.53
6	✓	0.27	0.33	0.43	0.58	0.94	1.43	3.28	4.60	9.55	17.03
	✗	0.07	0.09	0.13	0.20	0.34	0.61	1.18	2.39	4.40	9.04
7	✓	0.42	0.51	0.64	1.11	1.78	3.35	6.41	11.88	21.42	45.46
	✗	0.11	0.14	0.21	0.35	0.62	1.20	2.36	4.33	8.89	17.84
8	✓	0.63	0.78	1.28	2.03	3.80	6.45	13.54	23.88	50.78	96.72
	✗	0.16	0.24	0.39	0.66	1.25	2.44	4.48	8.84	17.71	42.60
9	✓	0.97	1.46	2.20	4.04	6.70	13.62	24.39	51.53	97.50	210.19
	✗	0.28	0.43	0.69	1.27	2.49	4.56	8.86	17.40	42.93	91.43
10	✓	1.76	2.49	4.35	6.98	13.80	24.61	51.97	97.72	207.47	397.12
	✗	0.49	0.74	1.32	2.52	4.53	8.87	17.91	42.33	92.55	184.21

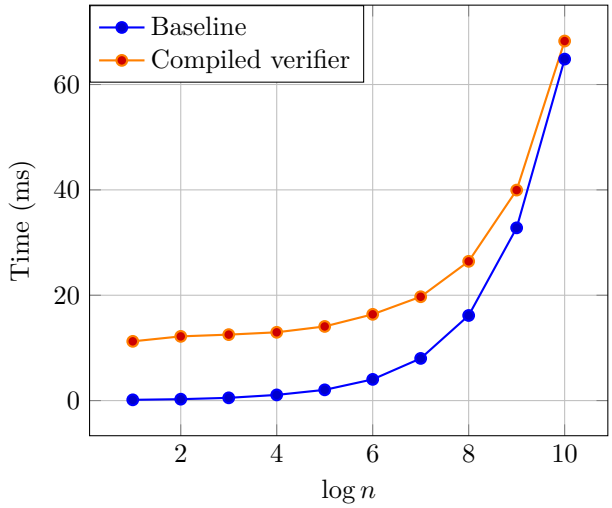


Figure 1: Timing for $t = 2^6$ across n values.

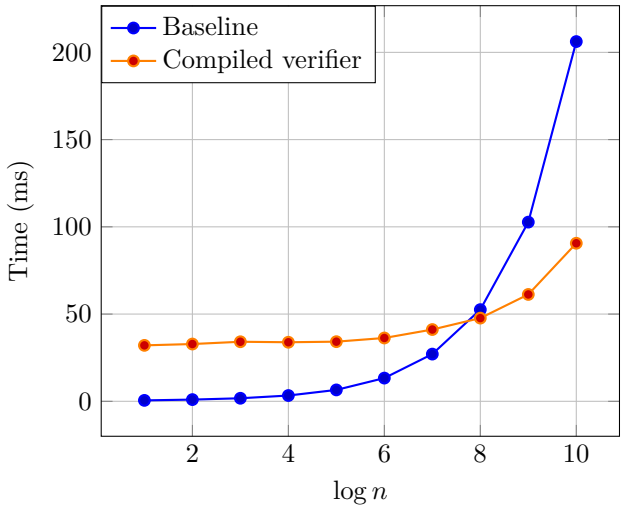


Figure 2: Timing for $t = 2^8$ across n values.

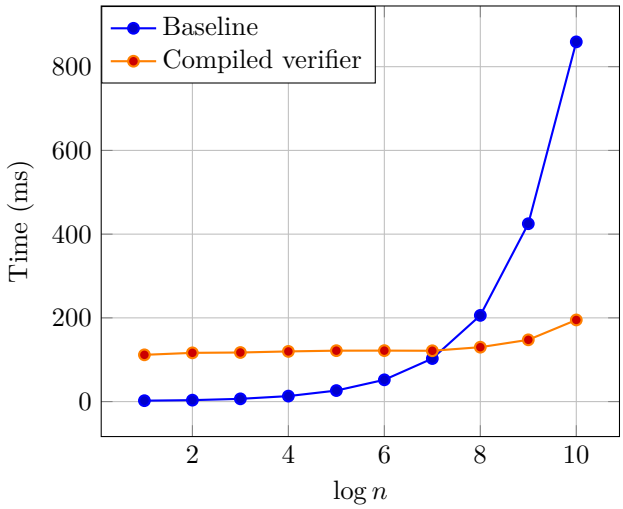


Figure 3: Timing for $t = 2^{10}$ across n values.

Table 3: Proof sizes (in kB).

$\log n$ \ $\log t$	1	2	3	4	5	6	7	8	9	10
1	131	135	141	140	145	150	153	156	164	166
2	155	159	161	167	171	173	178	182	184	187
3	167	171	175	180	183	188	192	193	195	200
4	177	183	185	190	191	199	203	201	205	211
5	190	192	195	199	206	209	210	213	217	219
6	206	210	213	217	222	222	224	230	232	234
7	216	219	224	227	230	228	236	238	241	242
8	223	228	232	235	235	240	242	243	246	252
9	234	238	239	243	247	248	250	253	258	260
10	249	251	253	257	259	263	264	268	272	272

Table 4: Output ciphertext vector size, using parameters from Table 1 (in kB).

$\log n$	1	2	3	4	5	6	7	8	9	10
$\lvert\mathbf{c}'\rvert$	32	64	128	256	512	1024	2048	4096	8192	16384

Table 5: Input compressed ciphertext vector size, using parameters from Table 1 (in kB).

$\log t$	1	2	3	4	5	6	7	8	9	10
$\lvert\mathbf{c}\rvert$	16	32	64	128	256	512	1024	2048	4096	8192

In addition to the SNARG proof, the prover also sends the ciphertext vector $\mathbf{c}'$ resulting from the protocol (c.f. Algorithm 2). This vector has bit size $n(\kappa+1)N\log q$, which amounts to between 32 kB for our smallest experiments and 16 MB for our largest experiments. This measure grows linearly in the number of rows of $\mathbf{A}$ and is represented in Table 4. After decryption, the output vector ($\mathbf{z}$ in Algorithm 2) has size $n\log p$, which ranges between 1 B and 512 B.

The plaintext input vector ($\mathbf{x}$ in Algorithm 2) has bit size $m\log p$, which ranges from 512 B for our smaller experiments to 512 kB for our larger experiments. Encrypting this vector using the compression technique discussed in Section 3.1 yields an input ciphertext vector ($\mathbf{c}$ in Algorithm 2) of size $tN\log q$, which ranges from 16 kB to 8 MB. This measure grows linearly with t the number of columns in the matrix $\mathbf{M}$ and is given in Table 5. The compression method follows prior work [FBK+25] and uses a 64-bit seed.

5 Performance of publicly verifiable PIR

In this section we overview the performance of our one-time preprocessing, publicly verifiable, PIR protocol. As discussed in Section 1.1, this work presents, to the best of our knowledge, the first instantiation of publicly verifiable PIR with one-time preprocessing.

For this evaluation, we use the GLWE parameters of Table 6. Comparatively to the parameters from Table 1, these parameters allow to pack more database information per matrix cell. This minimizes matrix size (and thus costs) for a given database size. The following measures are with respect to a 4 GiB database with a matrix of size $n = 2^{13} \times m = 2^{17.33}$, meaning each database element measures 24 KiB.

Computation Again, we use an AWS EC2 instance of type `Hpc7a.24xlarge` equipped with 48 CPU cores and 768 GB of memory to measure the client and server performance of our PIR protocol. The server time for a matrix of dimension $2^{13} \times 2^{18}$ is of ≈ 14 minutes to retrieve one 24 KiB database element. As per our previous analysis, more than half of this latency comes from the server computing the evaluation proof using the WHIR PCS.

On the client side, encryption, verification and decryption respectively take ≈ 20 ms, ≈ 650ms and ≈ 500ms. Therefore, the total client time is around 1.2s on a single thread.

Communication In the offline phase, the server simply commits to the matrix $\mathbf{M}$. This cost is entirely dependent on the PCS used. In our implementation using WHIR, the commitment size is 64 bytes.

Table 6: GLWE Parameters for the PIR use case

q	N	κ	$\log_2 \sigma$	p
$2^{64} - 2^{32} + 1$	2048	1	6.4	2^{24}

In the online phase, the client sends 1.26 MiB for the b-component of the input ciphertexts. The compressed $\mathbf{a}$-component measures only 8 bytes using the technique of [FBK+25]. At the end of the protocol, the client receives 256 MiB of ciphertext data, plus the proof size which is well under 1 MiB. Total online communication size is thus ≈ 258 MiB. These numbers follow from the generic communication costs analysis given in the previous section.

Acknowledgments

The authors would like to thank Petar Ivanov for insightful discussions and help in optimizing the implementation.

References

[AAB+24] Marius A Aardal, Diego F Aranha, Katharina Boudgoust, Sebastian Kolby, and Akira Takahashi. Aggregating falcon signatures with labrador. In *Annual International Cryptology Conference*, pages 71–106. Springer, 2024.

[ABPS24] Shahla Atapoor, Karim Baghery, Hilder V. L. Pereira, and Jannik Spiessens. Verifiable FHE via lattice-based snarks. *IACR Commun. Cryptol.*, 1(1):24, 2024.

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed-solomon proximity testing with super-fast verification. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 214–243, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland. doi:10.1007/978-3-031-91134-7_8.

[ACGS24] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part V*, volume 15488 of *Lecture Notes in Computer Science*, pages 302–334, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore. doi:10.1007/978-981-96-0935-2_10.

[AGHV22] Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. In *TCC (2)*, volume 13748 of *Lecture Notes in Computer Science*, pages 70–99. Springer, 2022.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[BC25] Dan Boneh and Binyi Chen. Latticefold+: faster, simpler, shorter lattice-based folding for succinct proof systems. In *Annual International Cryptology Conference*, pages 327–361. Springer, 2025.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 677–706, Cham, 2020. Springer International Publishing.

[BS23] Ward Beullens and Gregor Seiler. Labrador: compact proofs for r1cs from module-sis. In *Annual International Cryptology Conference*, pages 518–548. Springer, 2023.

[CCC+25] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable computation for approximate homomorphic encryption schemes. In *CRYPTO (7)*, volume 16006 of *Lecture Notes in Computer Science*, pages 643–677. Springer, 2025.

[CGG16] Ilaria Chillotti, Nicolas Gama, and Louis Goubin. Attacking FHE-based applications by software fault injections. Cryptology ePrint Archive, Report 2016/1164, 2016. URL: https://eprint.iacr.org/2016/1164.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, page 41, USA, 1995. IEEE Computer Society.

[CKP+23] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. Poster: Verifiable encodings for maliciously-secure homomorphic encryption evaluation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 3525–3527, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3576915.3624403.

[CS25] Elizabeth Crites and Alistair Stewart. On reed–solomon proximity gaps conjectures. Cryptology ePrint Archive, Paper 2025/2046, 2025. URL: https://eprint.iacr.org/2025/2046.

[dCL24] Leo de Castro and Keewoo Lee. Verisimplepir: Verifiability in simplepir at no online cost for honest servers. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/de-castro.

[DG25] Benjamin E. Diamond and Angus Gruen. On the distribution of the distances of random words. Cryptology ePrint Archive, Paper 2025/2010, 2025. URL: https://eprint.iacr.org/2025/2010.

[FBK+25] Jordan Frery, Roman Bredehoft, Jakub Klemsa, Arthur Meyre, and Andrei Stoian. Private lora fine-tuning of open-source llms with homomorphic encryption, 2025. URL: https://arxiv.org/abs/2505.07329, arXiv:2505.07329.

[FG12] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *CCS*, pages 501–512. ACM, 2012.

[GBK+25] Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zkSNARKs: for private proof delegation and verifiable computation over encrypted data. *IACR Communications in Cryptology*, 2(3), 2025. doi:10.62056/akgyl8n4e.

[GGW24] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-68403-6_14.

[GNSV23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *Journal of Cryptology*, 36(4):41, 2023.

[GQH+24] Yang Gao, Gang Quan, Soamar Homsi, Wujie Wen, and Liqiang Wang. Secure and efficient general matrix multiplication on cloud using homomorphic encryption. *The Journal of Supercomputing*, 80(18):26394–26434, 2024.

[HLHD22] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *USENIX Security Symposium*, pages 809–826. USENIX Association, 2022.

[HLMZ24] Mi-Ying Miryam Huang, Baiyu Li, Xinyu Mao, and Jiapeng Zhang. Fully homomorphic encryption with efficient public verification. *Cryptology ePrint Archive*, 2024.

[HMZ25] Mi-Ying Miryam Huang, Xinyu Mao, and Jiapeng Zhang. Sublinear proofs over polynomial rings. *IACR Cryptol. ePrint Arch.*, page 199, 2025.

[JAHT25] Sohyun Jeon, Calvin Abou Haidar, and Mehdi Tibouchi. Lastrings: Lattice-based scalable threshold ring signatures. *Cryptology ePrint Archive*, 2025.

[KVMH24] Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. vfhe: Verifiable fully homomorphic encryption. In *WAHC@CCS*, pages 11–22. ACM, 2024.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

[LLZ+25] Fengrun Liu, Haofei Liang, Tianyu Zhang, Yuncong Hu, Xiang Xie, Haisheng Tan, and Yu Yu. Hasteboots: Proving FHE bootstrapping in seconds. *IACR Cryptol. ePrint Arch.*, page 261, 2025.

[LS18] Wen-jie Lu and Jun Sakuma. More practical privacy-preserving machine learning as a service via efficient secure matrix multiplication. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 25–36, 2018.

[LZ22] Jing Liu and Liang Feng Zhang. Privacy-preserving and publicly verifiable matrix multiplication. *IEEE Transactions on Services Computing*, 16(3):2059–2071, 2022.

[MK25] Hannah Mahon and Shane Kosieradzki. Encrypted matrix multiplication using 3-dimensional rotations. *Cryptology ePrint Archive*, 2025.

[MMJG24] Xirong Ma, Chuan Ma, Yali Jiang, and Chunpeng Ge. Improved privacy-preserving pca using optimized homomorphic matrix multiplication. *Computers & Security*, 138:103658, 2024.

[MN24] Mark Manulis and Jérôme Nguyen. Fully homomorphic encryption beyond ind-cca1 security: Integrity through verifiability. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 63–93, Cham, 2024. Springer Nature Switzerland.

[NS24] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: fast polynomial commitments from lattices. In *Annual International Cryptology Conference*, pages 243–275. Springer, 2024.

[OANWO21] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. Blake3: One function, fast everywhere. 2021. URL: https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf.

[PLFT25] Kabir Peshawaria, Zeyu Liu, Ben Fisch, and Eran Tromer. Laminate: Succinct SIMD-friendly verifiable FHE. Cryptology ePrint Archive, Paper 2025/2285, 2025. URL: https://eprint.iacr.org/2025/2285.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.

[RLP26] Mayank Rathee, Keewoo Lee, and Raluca Ada Popa. Verifiable PIR with Small Client Storage . In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1423–1442, Los Alamitos, CA, USA, May 2026. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/SP63933.2026.00024, doi:10.1109/SP63933.2026.00024.

[SXZ+25] Muhammad Husni Santriaji, Jiaqi Xue, Yancheng Zhang, Qian Lou, and Yan Solihin. Dataseal: Ensuring the verifiability of private computation on encrypted data. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2378–2394. IEEE, 2025.

[TW25] Louis Tremblay Thibault and Michael Walter. Towards verifiable FHE in practice: Proving correct execution of tfhe's bootstrapping using plonky2. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pages 1113–1126. ACM, 2025. doi:10.1145/3719027.3765097.

[ZLW+25] Zhelei Zhou, Yun Li, Yuchen Wang, Zhaomin Yang, Bingsheng Zhang, Cheng Hong, Tao Wei, and Wenguang Chen. Zhe: Efficient zero-knowledge proofs for he evaluations. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3328–3346. IEEE, 2025.

[ZWL+25a] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. Phalanx: An FHE-friendly SNARK for verifiable computation on encrypted data. Cryptology ePrint Archive, Report 2025/302, 2025. URL: https://eprint.iacr.org/2025/302.

[ZWL+25b] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. Phalanx: An fhe-friendly snark for verifiable computation on encrypted data. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 4035–4048, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3719027.3765226.